# Bilgisayar Olimpiyatlarına Hazırlık

**Dost Seferoğlu**

**Bilgisayar Olimpiyatlarına Hazırlık**
Dost Seferoğlu

**e-ISBN:** 978-625-00-5243-3

T.C. Kültür ve Turizm Bakanlığı
Kütüphaneler ve Yayımlar Genel Müdürlüğü (ISBN Ajansı)

Yayın türü: Elektronik materyal (PDF)
Yayın yılı: 2026

© 2026 Dost Seferoğlu · Tüm hakları saklıdır.
Bu kitap ücretsiz olarak dağıtılır.

Kitabın güncel sürümü, çalışma rehberi ve her bölümle ilişkili
çıkmış sınav sorularının listesi:
https://bilgisayar-olimpiyatlari.com

# Önsöz

Lisedeki son iki yılımda alt dönemlerimden en çok duyduğum cümle, *"Abi sınava az kaldı ve ne yapmam gerektiğini bilmiyorum"* oldu. Bu soruyu soranlar tembel ya da isteksiz değil, aksine son derece parlak öğrencilerdi. 1. Aşama Sınavı kendine has yapısıyla özel bir hazırlık gerektirirken öğrencilerin önünde ne doğru dürüst bir yol haritası ne de doğrudan bu sınava odaklanan derli toplu bir kaynak vardı. Yıllarca bu arkadaşlarıma en fazla bir-iki ay yetecek dağınık notlar önerebilmenin burukluğunu yaşadım.

Bilgisayar olimpiyatı salt kod yazmaktan ibaret değildir; bir problemi matematiksel olarak modelleme ve bulunan fikri hayata geçirme işidir. Başarılı olanların ortak noktası soru kalıplarını ezberlemek değil, çözdükleri her sorudan bir ders çıkarıp bunu bir sonraki probleme uyarlayabilmeleridir. Bu kitap da yarışma hilelerinden ziyade sonlu matematik, sayılar teorisi ve algoritmaların kavramsal temellerini sağlam atmayı hedefler.

Bilgisayar olimpiyatı ağır bir zihinsel spordur. Saatlerce tek bir soruyu çözemediğiniz, bazı soyut konuları günlerce kafanızda oturtamadığınız zamanlar olacaktır. Önemli olan bu zorlanma anlarında pes etmemek ve kavramların zihninizde yer etmesi için kendinize zaman tanımaktır. Doğru bir yönlendirme ve düzenli çalışmayla geliştirilen problem çözme sezgisi, ileride hangi alana yönelirseniz yönelin karşılığını fazlasıyla verecektir.

Üniversiteye başlamadan önceki yaz tatilimde, henüz lisedeki tecrübelerim tazeyken bu kitabı hazırlamak istedim. Günümüzdeki yapay zekâ araçları dizgi ve kaynak tarama gibi rutin işleri oldukça kolaylaştırdı; ben de zamanımı müfredatı kurgulamaya, soruları titizlikle ayıklamaya ve adımları kontrol etmeye ayırabildim. Yaklaşık 600 sayfa olan bu kitabı acele etmeden, örnekleri kendi kâğıdınızda çözerek ve zamana yayarak sindirmenizi öneririm.

Herkesten çok destek olan aileme; birlikte saatlerce soru çözdüğümüz okul arkadaşlarıma, alt dönemlerime, TÜBİTAK Fen Lisesi öğretmenlerime ve okul müdürüm Dr. Hacı Ahmet Yıldırım'a teşekkür ederim. Destekleri için TÜBİTAK ve BİDEB'e, TÜBİTAK Bilgisayar Olimpiyatları komite başkanı Prof. Dr. Muhammed Fatih Demirci'ye, kamplardaki rehberlerime, desteğini hissettiğim Ahmet Kaan Avcı ağabeyime, özverili BİDEB personeli Dr. Burcu Çalışkan'a, üretkenlik konusunda bana rol model olan Dr. Fatih Kürşat Cansu'ya, beni bu projeye başlamaya YouTube'daki videolarıyla cesaretlendiren Prof. Dr. Oğuz Ergin'e ve derslerime katılan öğrencilerime şükranlarımı sunarım.

# Hakkında

Ben Dost Seferoğlu. TÜBİTAK Fen Lisesi'nin ilk mezunlarındanım ve Koç Üniversitesi Elektrik-Elektronik Mühendisliği Bölümü'nde lisans eğitimime başlıyorum.

Ortaokulda yazılımla hiçbir ilgim yoktu; bilgisayar olimpiyatlarına lise hazırlık sınıfının sonunda başladım. Okulun ilk öğrencileri olduğumuz için önümüzde bize yol gösterecek bir üst dönem yoktu. İlk bir buçuk yılımı tek başıma, ne kadar ilerlediğimi kestiremeden ve gereksiz yere yüzlerce kolay soru çözerek geçirdim. Doğru bir yönlendirme alamadığım için de ilk senemde 1. Aşama Sınavı'nı geçemedim.

Benim için asıl dönüm noktası, alt dönemlerin okula gelişiyle başladı. Yurdun ortak alanında tek başıma problem çözerken yanıma geldiler ve birlikte çalışmaya başladık. Ders çıkışlarında ve gece etütlerinde saatlerce süren bu ortak çalışma, zamanla okulumuzda bir kültüre dönüştü; birbirimize bilmediklerimizi anlatıp eksiklerimizi kapatacak bir ortam kurduk. Bir süre sonra okuldaki alt dönemlerime ders vermeye, lisenin son yılında ise TÜBİTAK kamplarında eğitmenlik yapmaya başladım. Bu süreçte IOI, Balkan ve Asya-Pasifik Bilgisayar Olimpiyatları gibi yarışmalarda ülkemizi temsil etme fırsatı buldum.

Kendimi hiçbir zaman bu işi kolayca kavrayan o "parlak" öğrencilerden biri olarak görmedim; aksine çoğu kavramı zihnimde oturtabilmek için çok uğraşmam gerekti. Ancak bir konuyu zor öğrenmiş olmak, anlatırken karşımdakinin nerede ve neden takılacağını sezmemi kolaylaştırdı. Bu kitap, o yıllarda bocalarken eksikliğini derinden hissettiğim sağlam ve derli toplu zemini benden sonrakilere sunma çabasıdır.

Bu kitabı ben yazmadım; yapay zekâ araçlarından yararlanarak derledim ve düzenledim. Müfredat, öncelikler ve kapsam benim tecrübeme dayanıyor; yazım ve dizgi işlerinde de bu araçlardan yararlandım. Bu araçların ürettiği metnin belirlediğim kalite çizgisinin üstünde kalmasını denetledim ve gerekli düzeltmeleri yaptım.

Kitapla ilgili her türlü eleştiri ve geri bildiriminiz için bana `dseferoglu26@ku.edu.tr` adresinden ulaşabilirsiniz.

# İçindekiler

# Bölüm 1

# Matematiğin Dili

Matematiksel düşüncenin ve bilgisayar biliminin temeli, düşünceleri hiçbir belirsizliğe yer bırakmayacak açıklıkta ifade etmektir. Günlük konuşma dili; sezgileri ve gündelik durumları aktarmakta zengin olsa da kesin mantıksal çıkarımlar söz konusu olduğunda çoğunlukla muğlaktır. Örneğin, "Herkes bu soruyu çözemez" cümlesi, hiç kimsenin soruyu çözemediğini mi anlatır, yoksa bazı kişilerin çözüp bazılarının çözemediğini mi? Günlük dilde bağlamdan ne kastedildiğini anlarız; fakat bir algoritmanın doğruluğunu kanıtlarken veya bir sayma problemini modellerken bu tür belirsizlikler hataya yol açar.

Olimpiyatlarda ve teorik bilgisayar biliminde karşılaştığımız problemler; girdi koşullarının net tanımlanmasını, durumların eksiksiz incelenmesini ve her adımın mantıksal olarak gerekçelendirilmesini gerektirir. Burada; ilerleyen bölümlerde kombinatorik, sayılar teorisi ve algoritmalar inşa edilirken başvuracağımız temel dili kuracağız: kümeler, mantıksal önermeler ve niceleyiciler.

## 1.1 Kümeler ve Gösterim

Matematikte en yalın kavramlar genellikle tanımlanması en güç olanlardır. Bir sepetteki elmalar, bir okuldaki öğrenciler veya 1 ile 100 arasındaki tek sayılar... Zihnimiz, belirli ortak özelliklere sahip nesneleri bir araya getirip tek bir bütün olarak düşünmeye yatkındır. Küme kavramı, bu sezgisel gruplamanın matematiksel olarak kesinleştirilmiş hâlidir.

Modern küme kuramının öncüsü Georg Cantor, kümeyi "iyi tanımlanmış, birbirinden farklı nesneler topluluğu" olarak ifade etmiştir. Buradaki en belirleyici ölçüt **iyi tanımlanmış** olma koşuludur. Bir topluluğun küme sayılabilmesi için, evrendeki herhangi bir nesnenin o topluluğa ait olup olmadığının nesnel bir kurala bağlı olması gerekir. Örneğin "TÜBİTAK sınavına hazırlanan çalışkan öğrenciler" ifadesi bir küme oluşturmaz; çünkü "çalışkanlık" kişiden kişiye değişen öznel bir nitelemedir. Buna karşılık "TÜBİTAK 1. Aşama sınavında 60 ve üzeri puan alan öğrenciler" kesin bir küme tanımlar.

**Tanım 1.1** (Küme, Eleman ve Aidiyet)**.** *Bir kümeyi oluşturan nesnelerin her birine o kümenin bir* ***elemanı*** *denir. Kümeler genellikle $A, B, C, S$ gibi büyük harflerle, elemanlar ise $a, b, x, y$ gibi küçük harflerle gösterilir.*

- *$x$ nesnesi $A$ kümesinin bir elemanı ise bu durum $x \in A$ biçiminde yazılır ve "$x$ elemanıdır $A$" diye okunur.*
- *$x$ nesnesi $A$ kümesine ait değilse bu durum $x \notin A$ biçiminde yazılır ve "$x$ elemanı değildir $A$" diye okunur.*

Kümeleri yazarken iki temel yöntem kullanılır:

1. **Liste Yöntemi:** Eleman sayısı az olduğunda elemanlar süslü parantez içine virgülle ayrılarak açıkça yazılır:

   $$A = \{1, 3, 5, 7, 9\}$$

   Bir kümede aynı elemanın birden fazla yazılması kümeyi değiştirmez; yani $\{1, 2, 2, 3\} = \{1, 2, 3\}$ kabul edilir. Ayrıca elemanların yazılış sırası önemsizdir: $\{1, 2\} = \{2, 1\}$.

2. **Ortak Özellik Yöntemi:** Elemanları sağladıkları bir kural ile ifade etmektir:

   $$S = \{x \mid P(x)\} \quad \text{veya} \quad S = \{x : P(x)\}$$

   Bu gösterim, "$P(x)$ koşulunu sağlayan tüm $x$ nesnelerinin kümesi" anlamına gelir.

Matematikte sıkça başvurulan standart sayı kümeleri için özel harfler kullanılır:

- $\mathbb{N} = \{0, 1, 2, 3, \dots\}$: Doğal sayılar kümesi (bu kitapta aksi belirtilmedikçe 0 doğal sayı kabul edilir). Pozitif doğal sayılar kümesi $\mathbb{N}^+ = \{1, 2, 3, \dots\}$ ile gösterilir.
- $\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$: Tam sayılar kümesi.
- $\mathbb{Q}$: Rasyonel sayılar kümesi; $a, b \in \mathbb{Z}$ ve $b \neq 0$ olmak üzere $\frac{a}{b}$ biçiminde yazılabilen sayılardır.
- $\mathbb{R}$: Reel (gerçel) sayılar kümesi; sayı doğrusundaki tüm noktaları temsil eder.

**Tanım 1.2** (Boş Küme ve Kümenin Büyüklüğü)**.** *Hiçbir elemanı bulunmayan kümeye* ***boş küme*** *denir ve $\emptyset$ simgesiyle (veya $\{\}$ biçiminde) gösterilir.*

*Sonlu bir $A$ kümesinin içerdiği farklı elemanların sayısına o kümenin* ***eleman sayısı*** *(veya kardinalitesi) denir ve $|A|$ ile gösterilir. Boş kümenin hiçbir elemanı olmadığından $|\emptyset| = 0$'dır.*

**Dikkat edilecek bir ayrıntı:** $\emptyset$ ile $\{\emptyset\}$ aynı kavramlar değildir. $\emptyset$ boş bir kümedir; $\{\emptyset\}$ ise içinde boş kümeyi barındıran tek elemanlı bir kümedir. Dolayısıyla $|\emptyset| = 0$ iken $|\{\emptyset\}| = 1$'dir.

**Tanım 1.3** (Alt Küme ve Küme Eşitliği)**.** *$A$ ve $B$ iki küme olsun. $A$'nın her elemanı aynı zamanda $B$'nin de bir elemanı oluyorsa, $A$'ya $B$'nin bir* ***alt kümesi*** *denir ve bu ilişki*

$$A \subseteq B$$

*biçiminde gösterilir. Eğer $A$ kümesi $B$'nin bir alt kümesi değilse (yani $A$'da olup $B$'de olmayan en az bir eleman bulunabiliyorsa) $A \not\subseteq B$ yazılır.*

*Eğer $A \subseteq B$ ve $A \neq B$ ise (yani $B$'de bulunup $A$'da yer almayan en az bir eleman varsa), $A$'ya $B$'nin bir* ***öz alt kümesi*** *denir ve $A \subsetneq B$ veya $A \subset B$ ile gösterilir.*

*İki kümenin birbirine eşit olması ($A = B$), her iki kümenin de tam olarak aynı elemanlara sahip olması demektir. Bu durum iki taraflı kapsama ile ifade edilir:*

$$A = B \iff (A \subseteq B \text{ ve } B \subseteq A)$$

**Önerme 1.4** (Boş Kümenin Evrenselliği)**.** *Herhangi bir $A$ kümesi için, $\emptyset \subseteq A$ ve $A \subseteq A$'dır.*

*Kanıt.* $A \subseteq A$ olduğu açıktır; çünkü $A$'dan alınan her eleman tanım gereği $A$'nın bir elemanıdır. $\emptyset \subseteq A$ olduğunu görmek için alt küme tanımının olumsuzuna bakalım: Eğer $\emptyset \not\subseteq A$ olsaydı, boş kümede bulunup $A$'da bulunmayan en az bir eleman gösterebilmemiz gerekirdi. Ancak boş kümenin hiçbir elemanı yoktur; dolayısıyla böyle bir karşı örnek bulmak imkânsızdır. O hâlde iddia zorunlu olarak doğrudur (bu mantıksal duruma *boş gerektirme* ya da *vacuous truth* denir). □

### 1.1.1 Küme İşlemleri ve Venn Şemaları

Yeni kümeler oluşturmak ve kümeleri birleştirmek için dört temel işlem kullanılır:

**Tanım 1.5** (Birleşim, Kesişim, Fark ve Tümleyen)**.** *$A$ ve $B$ iki küme, $U$ ise incelenen tüm nesneleri kapsayan evrensel küme olsun.*

1. ***Birleşim ($A \cup B$):*** *$A$'da veya $B$'de (veya her ikisinde) bulunan tüm elemanların kümesidir:*

$$A \cup B = \{x \mid x \in A \text{ veya } x \in B\}$$

2. ***Kesişim ($A \cap B$):*** *Hem $A$'da hem de $B$'de aynı anda bulunan ortak elemanların kümesidir:*

$$A \cap B = \{x \mid x \in A \text{ ve } x \in B\}$$

*Eğer iki kümenin hiçbir ortak elemanı yoksa ($A \cap B = \emptyset$), bu kümelere* ***ayrık kümeler*** *denir.*

3. ***Fark*** *($A \setminus B$* ***veya*** *$A - B$)**:** A'da olup B'de olmayan elemanların kümesidir:*

$$A \setminus B = \{x \mid x \in A \text{ ve } x \notin B\}$$

4. ***Tümleyen*** *($A^c$* ***veya*** *$\overline{A}$)**:** Evrensel kümede bulunup A'da yer almayan elemanların kümesidir ($A^c = U \setminus A$):*

$$A^c = \{x \in U \mid x \notin A\}$$

Küme işlemlerini görselleştirmenin en pratik yolu **Venn şemaları**dır. Düzlem üzerinde evrensel küme bir dikdörtgenle, alt kümeler ise bu dikdörtgenin içine çizilen kapalı eğrilerle (genellikle dairelerle) temsil edilir. Dairelerin kesişen bölgesi ortak elemanları, kapladıkları toplam alan birleşimi, dairenin dışında kalan bölge ise tümleyeni gösterir.

**Teorem 1.6** (De Morgan Kuralları)**.** *A ve B iki küme olmak üzere:*

$$(A \cup B)^c = A^c \cap B^c$$

$$(A \cap B)^c = A^c \cup B^c$$

*Kanıt.* İlk eşitliği ispatlayalım. İki kümenin eşit olduğunu göstermek için her birinin diğerini kapsadığını, yani $(A \cup B)^c \subseteq A^c \cap B^c$ ve $A^c \cap B^c \subseteq (A \cup B)^c$ olduğunu doğrulamalıyız.

Önce $x \in (A \cup B)^c$ olsun. Tanım gereği $x \notin (A \cup B)$'dir. Bir eleman birleşimde değilse, ne $A$'da ne de $B$'de bulunabilir; yani $x \notin A$ ve $x \notin B$ olmalıdır. Buradan $x \in A^c$ ve $x \in B^c$ çıkar. Her iki kümede de yer aldığı için $x \in (A^c \cap B^c)$ bulunur. Böylece $(A \cup B)^c \subseteq A^c \cap B^c$ doğrulanmış olur.

Tersine, $y \in (A^c \cap B^c)$ olsun. O hâlde $y \in A^c$ ve $y \in B^c$'dir; bu da $y \notin A$ ve $y \notin B$ demektir. $y$ ne $A$'da ne de $B$'de olduğuna göre, $y \notin (A \cup B)$ olmalıdır. Dolayısıyla $y \in (A \cup B)^c$ elde edilir. Bu da $A^c \cap B^c \subseteq (A \cup B)^c$ kapsamasını tamamlar. İki yönlü kapsama sağlandığından eşitlik geçerlidir. İkinci kural da benzer adımlarla gösterilir. □

### 1.1.2 Çözümlü Örnekler

**Örnek 1.7.** *Evrensel küme $U = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$ olsun. Bu evrende şu iki kümeyi tanımlayalım:*

$$A = \{x \in U \mid x \text{ çift sayıdır}\} = \{2, 4, 6, 8, 10\}$$

$$B = \{x \in U \mid x \geq 6\} = \{6, 7, 8, 9, 10\}$$

*Aşağıdaki kümeleri liste yöntemiyle bulunuz: a) $A \cap B$ b) $A \cup B$ c) $A \setminus B$ d) $B \setminus A$ e) $(A \cup B)^c$*

*Çözüm.* Tanımları doğrudan uygulayalım:

- a) $A \cap B$, her iki kümede ortak bulunan elemanlardır: $A \cap B = \{6, 8, 10\}$.
- b) $A \cup B$, iki kümedeki tüm elemanların tekrarsız listesidir: $A \cup B = \{2, 4, 6, 7, 8, 9, 10\}$.
- c) $A \setminus B$, $A$'da olup $B$'de olmayan (yani 6'dan küçük çift) elemanlardır: $A \setminus B = \{2, 4\}$.
- d) $B \setminus A$, $B$'de olup $A$'da olmayan (yani 6'dan büyük tek) elemanlardır: $B \setminus A = \{7, 9\}$.
- e) $(A \cup B)^c$, evrensel kümede bulunup $A \cup B$'de yer almayan elemanlardır. $U$'dan $\{2, 4, 6, 7, 8, 9, 10\}$ kümesini çıkardığımızda geriye $\{1, 3, 5\}$ kalır. De Morgan kuralı uyarınca $A^c = \{1, 3, 5, 7, 9\}$ ve $B^c = \{1, 2, 3, 4, 5\}$ kesiştirildiğinde de aynı sonuç, $A^c \cap B^c = \{1, 3, 5\}$, elde edilir.

□

**Örnek 1.8.** *$n$ elemanlı sonlu bir $S$ kümesinin toplam kaç farklı alt kümesi vardır? Küçük değerler için inceleyip genel kuralı bulunuz.*

*Çözüm.* Küçük değerleri listeleyerek örüntüyü görelim:

- $n = 0$ ise: $S = \emptyset$. Yalnızca 1 alt kümesi vardır: $\emptyset$. ($2^0 = 1$)
- $n = 1$ ise: $S = \{a_1\}$. Alt kümeler: $\emptyset, \{a_1\}$. Toplam 2 tane. ($2^1 = 2$)
- $n = 2$ ise: $S = \{a_1, a_2\}$. Alt kümeler: $\emptyset, \{a_1\}, \{a_2\}, \{a_1, a_2\}$. Toplam 4 tane. ($2^2 = 4$)
- $n = 3$ ise: $S = \{a_1, a_2, a_3\}$. Alt kümeler:

  $$\emptyset, \{a_1\}, \{a_2\}, \{a_3\}, \{a_1, a_2\}, \{a_1, a_3\}, \{a_2, a_3\}, \{a_1, a_2, a_3\}$$

  Toplam 8 alt küme oluşur. ($2^3 = 8$)

Genel kuralı şöyle kurabiliriz: Bir alt küme oluşturmak, $S$'deki her bir eleman için bağımsız bir seçim yapmak demektir. $a_1$ elemanı oluşturacağımız alt kümede ya yer alacaktır ya da almayacaktır (2 seçenek). Benzer biçimde $a_2$ için 2, $a_3$ için 2 ve nihayet $a_n$ için 2 seçenek bulunur.

Bölüm 3'te ele alacağımız çarpma kuralı uyarınca, her eleman için yapılan bu bağımsız seçimlerin toplam sonucu çarpımla belirlenir:

$$\underbrace{2 \times 2 \times \cdots \times 2}_{n \text{ tane}} = 2^n$$

Demek ki $n$ elemanlı bir kümenin tam $2^n$ alt kümesi vardır. (Bu yaklaşım, Bölüm 32'de göreceğimiz üzere bilgisayar biliminde alt kümelerin $n$ bitlik ikili sayılarla temsil edilmesinin de doğrudan temelidir). □

**Örnek 1.9.** *Bir okulun bilgisayar kulübünde* 25 *öğrenci vardır. Bu öğrencilerden* 16*'sı Python,* 13*'ü C dilini bilmektedir.* 4 *öğrenci ise her iki dili de bilmemektedir. Yalnızca C dilini bilen kaç öğrenci vardır?*

*Çözüm.* Evrensel küme kulüpteki tüm öğrenciler olsun: $|U| = 25$. Python bilenlerin kümesi $P$, C bilenlerin kümesi $C$ olsun. Her iki dili de bilmeyen 4 öğrenci olduğuna göre, en az bir dili bilenlerin sayısı:

$$|P \cup C| = |U| - 4 = 25 - 4 = 21$$

olmalıdır.

Birleşim formülü gereğince (bu formülü Bölüm 8'de içerme-dışarma ilkesi başlığı altında genelleştireceğiz):

$$|P \cup C| = |P| + |C| - |P \cap C|$$

Bilinen değerleri yerine yazalım:

$$21 = 16 + 13 - |P \cap C| \implies 21 = 29 - |P \cap C| \implies |P \cap C| = 8$$

Her iki dili de bilen 8 öğrenci vardır.

Yalnızca C bilenler $C \setminus P$ kümesidir:

$$|C \setminus P| = |C| - |P \cap C| = 13 - 8 = 5$$

Dolayısıyla yalnızca C dilini bilen 5 öğrenci bulunmaktadır. □

**Örnek 1.10.** *Herhangi iki $A$ ve $B$ kümesi için $(A \setminus B) \cap B = \emptyset$ olduğunu eleman bazında gösteriniz.*

*Çözüm.* İddiayı çelişki yöntemiyle gösterelim (bu yöntemin ayrıntılarını Bölüm 2'de ele alacağız). Kesişimin boş küme olmadığını, yani bu kesişime ait en az bir $x$ elemanı bulunduğunu varsayalım:

$$x \in (A \setminus B) \cap B$$

Kesişim tanımı gereği:

1. $x \in (A \setminus B)$
2. $x \in B$

Birinci maddedeki $x \in (A \setminus B)$ ifadesini fark kümesi tanımına göre açarsak:

$$x \in A \quad \text{ve} \quad x \notin B$$

elde edilir. Bu durumda hem $x \notin B$ hem de ikinci maddeden gelen $x \in B$ aynı anda sağlanmalıdır. Bir eleman bir kümenin hem elemanı olup hem elemanı olmamazlık edemez; bu açık bir mantıksal çelişkidir.

Bu çelişki, kesişimin boş küme olmadığı varsayımından doğmuştur. O hâlde varsayım yanlıştır ve $(A \setminus B) \cap B = \emptyset$ olmak zorundadır. □

## 1.2 Mantık: Önermeler ve Bağlaçlar

Kümeler dünyasında nesneleri tanımladıktan sonra, bu nesneler hakkında kesin yargılarda bulunabilmek için **önermeler mantığı**ndan yararlanırız. Bir programda yer alan `if` koşullarından algoritmaların değişmezlerine kadar birçok yapı mantık kuralları üzerine oturur.

**Tanım 1.11** (Önerme ve Doğruluk Değeri)**.** *Doğru ya da yanlış kesin bir hüküm bildiren ifadelere* ***önerme*** *denir. Önermeler genellikle $p, q, r, s$ gibi harflerle gösterilir.*

- *Bir önerme ya* ***doğru*** *(D veya 1) ya da* ***yanlış*** *(Y veya 0) değerini alır. Bu değere önermenin* ***doğruluk değeri*** *denir.*
- *Soru, emir, ünlem veya kişisel yargı belirten cümleler önerme değildir.*

Örneğin:

- "$2 + 3 = 5$" bir önermedir ve doğruluk değeri 1'dir.
- "7 bir çift sayıdır" bir önermedir ve doğruluk değeri 0'dır.
- "Lütfen kapıyı kapat" bir önerme değildir (emir cümlesidir).
- "Bu kitap çok sürükleyici" bir önerme değildir (öznel beğeni bildirir).

### 1.2.1 Mantıksal Bağlaçlar ve Doğruluk Tabloları

Tekil önermeleri birbirine bağlayarak bileşik önermeler elde ederiz. Bu bağlaçların davranışı **doğruluk tabloları** ile kesinleşir.

**Tanım 1.12** (Değil, Ve, Veya)**.** *$p$ ve $q$ iki önerme olsun.*

1. ***Değil ($\neg p$ veya $p'$):*** *$p$ doğru ise $\neg p$ yanlış, $p$ yanlış ise $\neg p$ doğrudur.*
2. ***Ve ($\wedge$):*** *$p \wedge q$ bileşik önermesi, yalnızca hem $p$ hem de $q$ doğru olduğunda doğrudur; diğer tüm durumlarda yanlıştır.*
3. ***Veya ($\vee$):*** *$p \vee q$ bileşik önermesi, $p$ veya $q$'dan en az biri doğru olduğunda doğrudur; yalnızca her iki önerme de yanlış olduğunda yanlıştır.*

Aşağıdaki tablo, bu üç temel işlemin doğruluk değerlerini özetler:

**Dikkat:** Günlük dildeki "veya" sözcüğü bazen dışlayıcı anlamda ("çay veya kahve alabilirsiniz" dendiğinde yalnızca birinin seçilmesi kastedilir) kullanılır. Ancak matematikteki $\vee$ bağlacı daima **kapsayıcı veya**dır; her iki durumun birlikte doğru olması da kabul edilir.

| $p$ | $q$ | $\neg p$ | $p \wedge q$ | $p \vee q$ |
|---|---|---|---|---|
| 1 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 |

Tablo 1.1: Temel Mantık Bağlaçlarının Doğruluk Tablosu

### 1.2.2 Koşullu Önerme (Gerektirme) ve Karşıt Pozitif

Matematiksel savların büyük bir kısmı "Eğer ... ise ..." yapısındadır. Bu ilişkiyi koşullu önerme temsil eder.

**Tanım 1.13** (Koşullu Önerme / Gerektirme: $p \Rightarrow q$)**.** *$p$ ve $q$ iki önerme olsun. "$p$ ise $q$" önermesi $p \Rightarrow q$ sembolüyle gösterilir. Burada $p$'ye **hipotez** (varsayım), $q$'ya ise **hüküm** (sonuç) denir.*

*$p \Rightarrow q$ önermesi, yalnızca hipotezin doğru (1) fakat hükmün yanlış (0) olduğu durumda **yanlış** kabul edilir. Diğer tüm durumlarda **doğru**dur.*

Hipotez yanlışken ($p = 0$), $p \Rightarrow q$ önermesinin neden doğru kabul edildiğini anlamak için bir **taahhüt** benzetmesi yapılabilir: Bir öğretmen öğrencisine, "Eğer bu sınavdan 100 alırsan ($p$), sana bir kitap hediye edeceğim ($q$)" demiş olsun. Öğretmen hangi durumda sözünü tutmamış sayılır?

- Öğrenci 100 alır ($p = 1$) ve öğretmen kitabı verirse ($q = 1$): Söz yerine getirilmiştir (1).
- Öğrenci 100 alır ($p = 1$) fakat öğretmen kitabı vermezse ($q = 0$): Söz çiğnenmiştir (0).
- Öğrenci 100 alamazsa ($p = 0$): Öğretmen öğrenciye yine de bir kitap hediye etse ($q = 1$) veya etmese ($q = 0$), verdiği sözü bozmuş sayılmaz; çünkü sözün önkoşulu gerçekleşmemiştir. Dolayısıyla her iki durumda da koşul ihlal edilmemiştir (1).

**Tanım 1.14** (Karşıt, Ters ve Karşıt Pozitif)**.** *Verilen bir $p \Rightarrow q$ koşullu önermesi için:*

- *$q \Rightarrow p$ önermesine bu koşulun **karşıtı** denir.*
- *$\neg p \Rightarrow \neg q$ önermesine bu koşulun **tersi** denir.*
- *$\neg q \Rightarrow \neg p$ önermesine bu koşulun **karşıt pozitifi** (contrapositive) denir.*

Bu tablo incelendiğinde temel bir teorem ortaya çıkar:

| $p$ | $q$ | $p \Rightarrow q$ | Karşıt: $q \Rightarrow p$ | Ters: $\neg p \Rightarrow \neg q$ | Karşıt Pozitif: $\neg q \Rightarrow \neg p$ |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 | 1 |

Tablo 1.2: Koşullu Önerme Türevlerinin Doğruluk Tablosu

**Teorem 1.15** (Karşıt Pozitifin Denkliği)**.** *Bir koşullu önerme ile onun karşıt pozitifi mantıksal olarak birbirine denktir:*

$$(p \Rightarrow q) \equiv (\neg q \Rightarrow \neg p)$$

*Buna karşılık bir önerme, kendi karşıtına veya tersine denk olmak zorunda DEĞİLDİR:*

$$(p \Rightarrow q) \not\equiv (q \Rightarrow p)$$

Bu denklik, ispat yöntemlerinde sıkça kullanılır. Bir iddianın doğrudan ispatlanması zor olduğunda, karşıt pozitifini ele almak aynı sonucu verir. Örneğin "$n^2$ tek sayı ise $n$ tek sayıdır" önermesini doğrudan ispatlamak yerine, onun karşıt pozitifi olan "$n$ çift sayı ise $n^2$ çift sayıdır" iddiasını göstermek çok daha pratiktir (Bölüm 2'de bunu ayrıntısıyla ele alacağız).

**Tanım 1.16** (İki Yönlü Koşullu Önerme / Ancak ve Ancak: $p \Leftrightarrow q$)**.** *$p$ ile $q$ önermelerinin birbirini karşılıklı gerektirmesi durumudur:*

$$(p \Leftrightarrow q) \equiv (p \Rightarrow q) \wedge (q \Rightarrow p)$$

*$p \Leftrightarrow q$ önermesi; $p$ ile $q$ aynı doğruluk değerine sahipken doğru, farklı değerlere sahipken yanlıştır.*

**Teorem 1.17** (Mantıksal De Morgan Kuralları ve Dağılma Özellikleri)**.** *Herhangi $p, q, r$ önermeleri için aşağıdaki mantıksal denklikler geçerlidir:*

$$\begin{aligned}
\neg(p \wedge q) &\equiv \neg p \vee \neg q \\
\neg(p \vee q) &\equiv \neg p \wedge \neg q \\
p \wedge (q \vee r) &\equiv (p \wedge q) \vee (p \wedge r) \\
p \vee (q \wedge r) &\equiv (p \vee q) \wedge (p \vee r) \\
p \Rightarrow q &\equiv \neg p \vee q
\end{aligned}$$

$p \Rightarrow q \equiv \neg p \vee q$ denkliği, koşullu önermenin temel bağlaçlarla nasıl ifade edilebileceğini gösterir.

### 1.2.3 Çözümlü Örnekler

**Örnek 1.18.** *$\neg(p \Rightarrow q) \equiv p \wedge \neg q$ denkliğini doğruluk tablosuyla gösteriniz.*

*Çözüm.* $p$ ve $q$'nun alabileceği dört farklı doğruluk değeri kombinasyonu için değerleri adım adım hesaplayalım:

| $p$ | $q$ | $p \Rightarrow q$ | $\neg(p \Rightarrow q)$ | $\neg q$ | $p \wedge \neg q$ |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 |

Tablonun 4. sütunu ($\neg(p \Rightarrow q)$) ile 6. sütunundaki ($p \wedge \neg q$) doğruluk değerlerinin her satırda aynı $(0, 1, 0, 0)$ olduğu görülür. Dolayısıyla iki ifade birbirine denktir.

Bu sonucun günlük dildeki karşılığı şudur: "Eğer yağmur yağarsa yerler ıslanır" iddiasını çürütmek için yağmurun yağdığını fakat yerlerin ıslanmadığını göstermek gerekir ($p \wedge \neg q$). □

**Örnek 1.19.** *Aşağıdaki önermelerin karşıtını, tersini ve karşıt pozitifini yazınız:*

1. *"Bir sayı 4'e tam bölünüyorsa, o sayı çifttir."*
2. *"$x > 3$ ise $x^2 > 9$'dur ($x \in \mathbb{R}$)."*

*Çözüm.* İlk önermeyi bileşenlerine ayıralım: $p$: "Sayı 4'e tam bölünür", $q$: "Sayı çifttir".

- **Karşıtı ($q \Rightarrow p$):** "Bir sayı çift ise, o sayı 4'e tam bölünür." (Görüleceği gibi bu iddia yanlıştır; örneğin 6 çifttir ama 4'e bölünmez).
- **Tersi ($\neg p \Rightarrow \neg q$):** "Bir sayı 4'e tam bölünmüyorsa, o sayı tek sayıdır." (Bu da yanlıştır; 6 sayısı 4'e bölünmez fakat çifttir).
- **Karşıt Pozitifi ($\neg q \Rightarrow \neg p$):** "Bir sayı tek ise (çift değilse), o sayı 4'e tam bölünmez." (Bu ifade doğrudur ve ana önermeye denktir).

İkinci önerme için: $p$: "$x > 3$", $q$: "$x^2 > 9$".

- **Karşıtı ($q \Rightarrow p$):** "$x^2 > 9$ ise $x > 3$'tür." (Yanlıştır; $x = -4$ için $x^2 = 16 > 9$ olur fakat $-4 > 3$ sağlanmaz).
- **Tersi ($\neg p \Rightarrow \neg q$):** "$x \leq 3$ ise $x^2 \leq 9$'dur." (Yanlıştır; $x = -5 \leq 3$ iken $x^2 = 25 \leq 9$ değildir).
- **Karşıt Pozitifi ($\neg q \Rightarrow \neg p$):** "$x^2 \leq 9$ ise $x \leq 3$'tür." (Doğrudur; $x^2 \leq 9 \iff -3 \leq x \leq 3$ olduğundan $x \leq 3$ zorunlu olarak sağlanır).

□

**Örnek 1.20** (Klasik Mantık Bulmacası: Şövalyeler ve Yalancılar)**.** *Yalnızca iki tür insanın yaşadığı bir adadasınız:* ***Şövalyeler*** *daima doğruyu söyler,* ***Yalancılar*** *ise daima yalan söyler. Adada iki yerliyle (A ve B) karşılaşıyorsunuz.*

*A kişisi şöyle bir cümle kuruyor: "İkimizden en az biri yalancıdır."*

*A ve B'nin kimliklerini mantıksal olarak belirleyiniz.*

*Çözüm.* Durum analizi yaparak ilerleyelim. $A$'nın olası iki durumunu inceleyelim:

1. **Durum:** $A$ bir yalancı olsun.

- $A$ yalancı ise söylediği cümle mantıksal olarak **yanlış** (0) olmalıdır.
- İfade şudur: "$A$ yalancıdır VEYA $B$ yalancıdır."
- Bu ifadenin yanlış olabilmesi için her iki bileşenin de yanlış olması gerekir; yani hem $A$ hem de $B$ yalancı olmamalıdır.
- Buradan $A$'nın şövalye olması gerektiği çıkar.
- Fakat başlangıç varsayımı $A$'nın yalancı olduğuydu. Bir kişi aynı anda hem yalancı hem şövalye olamayacağından bu bir çelişkidir.
- O hâlde $A$ yalancı olamaz.

2. **Durum:** $A$ bir şövalye olsun.

- $A$ şövalye ise söylediği cümle mantıksal olarak **doğru** (1) olmalıdır.
- İfadenin doğru olması için "$A$ yalancıdır VEYA $B$ yalancıdır" önermesi doğru olmalıdır.
- $A$ şövalye olduğundan birinci kısım ("$A$ yalancıdır") yanlıştır (0).
- VEYA bağlacında sonucun 1 çıkabilmesi için ikinci kısmın zorunlu olarak doğru olması gerekir; yani "$B$ yalancıdır" önermesi doğru (1) olmalıdır.
- Buradan $B$'nin yalancı olduğu sonucuna varılır.

Sonuç olarak $A$ kesinlikle bir **şövalye**, $B$ ise kesinlikle bir **yalancı**dır. □

**Örnek 1.21.** *Bir yazılımda iki tamsayı değişkeni x ve y olsun. Şu kontrol bloğunu ele alalım:*

```
if (!(x > 0 && y <= 5)) {
    // Calisacak kod blogu
}
```

*Bu `if` koşulunu, başında olumsuzlama işareti (`!`) kalmayacak biçimde mantıksal De Morgan kurallarını kullanarak sadeleştiriniz.*

*Çözüm.* Önermeleri adlandıralım:

- $p$: `x > 0`
- $q$: `y <= 5`

Koşul ifadesi $\neg(p \wedge q)$ biçimindedir. De Morgan kuralı uyarınca:

$$\neg(p \wedge q) \equiv \neg p \vee \neg q$$

Tekil önermelerin olumsuzları:

- $\neg p$: `x > 0` ifadesinin olumsuzu `x <= 0`'dır.
- $\neg q$: `y <= 5` ifadesinin olumsuzu `y > 5`'tir.

Böylece koşul doğrudan şuna dönüşür:

```
if (x <= 0 || y > 5) {
    // Calisacak kod blogu
}
```

Bu iki koşul tüm olası $x$ ve $y$ değerleri için tamamen aynı mantıksal sonucu üretir. □

## 1.3 Niceleyiciler

Önermeler mantığında sabit önermelerle $(p, q)$ çalıştık. Ancak matematikte sıklıkla bir değişkene bağlı olan ve o değişkenin değerine göre doğruluğu değişen ifadeler yer alır. Örneğin "$x > 0$" cümlesi tek başına bir önerme belirtmez; çünkü $x$'in değeri bilinmeden doğruluktan söz edilemez. Bu tür ifadelere **açık önerme** denir ve $P(x)$ ile gösterilir. $x = 5$ için $P(5)$ doğru iken, $x = -2$ için $P(-2)$ yanlıştır.

Bir açık önermeyi kesin bir önermeye dönüştürmek için değişkenin hangi kapsamda ele alındığını belirten **niceleyiciler** kullanılır.

**Tanım 1.22** (Evrensel ve Varlıksal Niceleyiciler)**.** *$P(x)$ bir $A$ evrensel kümesi üzerinde tanımlı açık bir önerme olsun.*

1. ***Evrensel Niceleyici ($\forall$, "her" ya da "bütün"):***

   $$\forall x \in A, P(x)$$

   *ifadesi, "$A$ kümesindeki her $x$ elemanı için $P(x)$ doğrudur" anlamına gelir. Bu ifadenin doğru sayılması için istisnasız tüm elemanların koşulu sağlaması gerekir. Tek bir aykırı eleman önermeyi yanlış kılar.*

2. ***Varlıksal Niceleyici (∃, "en az bir", "bazı" ya da "vardır"):***

$$\exists x \in A, P(x)$$

*ifadesi, "A kümesinde $P(x)$ koşulunu sağlayan en az bir $x$ elemanı vardır" anlamına gelir. Bu ifadenin doğru olması için koşulu sağlayan tek bir örnek bulmak yeterlidir. Yalnızca hiçbir eleman koşulu sağlamadığında önerme yanlış olur.*

### 1.3.1 Niceleyicilerin Olumsuzlanması

Matematiksel ispatlarda bir savın olumsuzunu doğru kurmak esastır. Bir iddiayı çürütmek, onun olumsuzunun doğru olduğunu göstermekle eşdeğerdir.

**Teorem 1.23** (Niceleyicilerin Olumsuzlanması Kuralı)**.** *$P(x)$ bir açık önerme olmak üzere:*

$$\neg(\forall x \in A, P(x)) \equiv \exists x \in A, \neg P(x)$$

$$\neg(\exists x \in A, P(x)) \equiv \forall x \in A, \neg P(x)$$

*Kanıt.* İlk denkliği inceleyelim: "Bir sınıftaki tüm öğrenciler gözlüklüdür" $(\forall x, P(x))$ iddiasını çürütmek için bütün öğrencilerin gözlüksüz olduğunu göstermek gerekmez. Sınıfta gözlük takmayan tek bir öğrenci bulmak $(\exists x, \neg P(x))$ iddianın yanlış olduğunu kanıtlamaya yeterlidir. Bu tek aykırı örneğe matematikte **karşı örnek** (counterexample) denir (karşı örnekle çürütme tekniği Bölüm 2'de ele alınacaktır).

İkinci denklik de benzer biçimde çalışır: "Bu odada bir yabancı var" $(\exists x, P(x))$ iddiasının olumsuzu, odadaki herkesin tanıdık olduğunu; yani odadaki her bireyin yabancı olmadığını $(\forall x, \neg P(x))$ ifade etmektir. □

### 1.3.2 Çoklu Niceleyiciler ve Sıralamanın Önemi

Birden fazla değişken içeren açık önermelerde ($P(x, y)$ gibi) niceleyicilerin yazılış sırası cümlenin anlamını bütünüyle değiştirir. Bu ayrım, niceleyicileri anlamadaki en önemli noktalardan biridir.

İnsanlar evreninde tanımlı şu iki önermeyi karşılaştıralım:

1. $\forall x, \exists y, \text{Anne}(y, x)$: "Her $x$ insanı için, $x$'in annesi olan bir $y$ insanı vardır." (Bu önerme **doğrudur**; her insanın bir annesi bulunur).

2. $\exists y, \forall x, \text{Anne}(y, x)$: "Öyle bir $y$ insanı vardır ki, herkesin $(x)$ annesidir." (Bu önerme **yanlıştır**; tüm insanların ortak tek bir biyolojik annesi yoktur).

Niceleyicilerin sırasının değişmesi önermenin anlamını tamamen başkalaştırır:

- $\forall x \exists y$ ifadesinde seçilecek $y$ elemanı, $x$'e **bağlı olabilir** (her $x$ için farklı bir $y$ belirlenebilir).
- $\exists y \forall x$ ifadesinde ise tek ve sabit bir $y$ bulunmalıdır; bu $y$, istisnasız tüm $x$'ler için koşulu aynı anda sağlamalıdır.

### 1.3.3 Çözümlü Örnekler

**Örnek 1.24.** *Aşağıdaki matematiksel ifadelerin doğruluk değerlerini belirleyiniz ve her birinin olumsuzunu alınız:*

1. $\forall x \in \mathbb{R}, x^2 \geq 0$
2. $\exists x \in \mathbb{Z}, x + 5 = 2$
3. $\forall x \in \mathbb{N}, x - 1 \in \mathbb{N}$

*Çözüm.* Sırasıyla inceleyelim:

1. $\forall x \in \mathbb{R}, x^2 \geq 0$: Her reel sayının karesi sıfırdan büyük veya eşittir (pozitif sayıların karesi pozitif, negatiflerin karesi pozitif, sıfırın karesi sıfırdır). Dolayısıyla doğruluk değeri 1 **(doğru)**dur.

   Olumsuzu: $\neg(\forall x \in \mathbb{R}, x^2 \geq 0) \equiv \exists x \in \mathbb{R}, x^2 < 0$. ("Karesi sıfırdan küçük en az bir reel sayı vardır").

2. $\exists x \in \mathbb{Z}, x + 5 = 2$: Denklem çözüldüğünde $x = 2 - 5 = -3$ bulunur. $-3$ bir tam sayı ($\mathbb{Z}$) olduğundan koşulu sağlayan bir eleman mevcuttur. Dolayısıyla doğruluk değeri 1 **(doğru)**dur.

   Olumsuzu: $\neg(\exists x \in \mathbb{Z}, x + 5 = 2) \equiv \forall x \in \mathbb{Z}, x + 5 \neq 2$. ("Her $x$ tam sayısı için $x + 5 \neq 2$'dir").

3. $\forall x \in \mathbb{N}, x - 1 \in \mathbb{N}$: Doğal sayılar kümesi $\mathbb{N} = \{0, 1, 2, \dots\}$ biçimindedir. $x = 0$ doğal sayısı için $0 - 1 = -1$ elde edilir. Fakat $-1 \notin \mathbb{N}$'dir. Tek bir karşı örnek iddiayı geçersiz kılmaya yeterlidir. Dolayısıyla doğruluk değeri 0 **(yanlış)**dır.

   Olumsuzu: $\neg(\forall x \in \mathbb{N}, x - 1 \in \mathbb{N}) \equiv \exists x \in \mathbb{N}, x - 1 \notin \mathbb{N}$. ($x = 0$ karşı örneği nedeniyle bu olumsuz ifade doğrudur).

□

**Örnek 1.25.** *Reel sayılar evreninde ($\mathbb{R}$) şu iki önermeyi ele alalım:*

1. $P_1 : \forall x \in \mathbb{R}, \exists y \in \mathbb{R}, (x + y = 10)$
2. $P_2 : \exists y \in \mathbb{R}, \forall x \in \mathbb{R}, (x + y = 10)$

*Bu iki önermenin doğruluk değerlerini belirleyip aralarındaki farkı açıklayınız.*

*Çözüm.* $P_1$ **önermesi:** Verilen herhangi bir $x$ reel sayısı için, onunla toplandığında 10 sonucunu veren bir $y$ reel sayısı seçilebilir mi? $x$ ne olursa olsun $y = 10 - x$ seçilirse:

$$x + y = x + (10 - x) = 10$$

sağlanır. $x \in \mathbb{R}$ olduğundan $10 - x$ de daima bir reel sayıdır. Her $x$ için uygun bir $y$ üretilebildiği için $P_1$ önermesi **doğru** (1)dur. Burada $y$'nin değeri seçilen $x$'e bağlı olarak değişir.

$P_2$ **önermesi:** Öyle sabit bir $y$ reel sayısı var mıdır ki, hangi $x$ reel sayısı seçilirse seçilsin $x + y = 10$ eşitliği korunsun? Böyle bir $y$ sayısı olduğunu varsayalım. Bu eşitlik $x = 0$ için de sağlanmalıdır:

$$0 + y = 10 \implies y = 10$$

Fakat aynı $y = 10$ değeri $x = 1$ için de geçerli olmalıdır:

$$1 + 10 = 10 \implies 11 = 10$$

Bu açık bir çelişkidir. Tüm $x$'ler için aynı anda çalışan tek bir $y$ reel sayısı bulunamaz. Dolayısıyla $P_2$ önermesi **yanlış** (0)dır. □

**Örnek 1.26.** *Bir algoritma tasarımcısı, $N$ elemanlı bir tamsayı dizisinin $(A[0], A[1], \ldots, A[N-1])$ sıralı olduğunu iddia etmektedir.*

1. *Bu iddiayı niceleyicilerle matematiksel olarak ifade ediniz.*
2. *İddianın olumsuzunu yazınız.*
3. *Bu iddiayı doğrulayan bir algoritmanın neden tüm diziyi incelemek zorunda olduğunu, iddiayı çürütmek isteyen bir testin ise neden bazen tek adımda sonlanabileceğini açıklayınız.*

*Çözüm.* **1. İddianın Niceleyici ile İfadesi:** Dizinin küçükten büyüğe sıralı olması, ardışık her eleman çiftinde soldakinin sağdakinden küçük ya da ona eşit olması demektir:

$$\forall i \in \{0, 1, \ldots, N-2\}, \quad A[i] \leq A[i+1]$$

**2. İddianın Olumsuzu:** Niceleyicinin olumsuzunu alma kuralını uygulayalım ($\neg\forall \to \exists$):

$$\exists i \in \{0, 1, \ldots, N-2\}, \quad A[i] > A[i+1]$$

Yani dizinin sıralı olmadığını göstermek, soldaki elemanın sağdakinden büyük olduğu en az bir $i$ indisi bulmak anlamına gelir.

**3. Algoritmik Çıkarım:** Evrensel iddiaların ($\forall$) doğrulanması ile çürütülmesi arasındaki yapısal fark, Bölüm 21'de göreceğimiz algoritma karmaşıklığının da temel nedenlerinden biridir:

- Dizinin sıralı olduğunu **doğrulamak** istiyorsak, tek bir ihlalin dahi bulunmadığından emin olmak zorundayız. Bu nedenle algoritma $i = 0$'dan $N - 2$'ye kadar tüm indisleri denetlemeli, yani diziyi bütünüyle taramalıdır ($O(N)$ işlem).
- Buna karşılık iddiayı **çürütmek** için tek bir karşı örnek ($\exists$) yeterlidir. Örneğin $A[0] = 9$ ve $A[1] = 2$ ise, algoritma ilk adımda $A[0] > A[1]$ ihlalini yakalar ve dizinin kalan $N - 2$ elemanına bakmadan doğrudan "dizi sıralı değildir" yanıtını verebilir.

Bu mantık, programlamadaki kısa devre değerlendirmesi (short-circuit evaluation) ve arama algoritmalarındaki erken sonlanma (early exit) stratejilerinin temelini oluşturur. □

# Bölüm 2

# İspat Nasıl Yapılır

Bir yazılımcı yazdığı kodun doğru çalıştığını anlamak için onlarca, bazen yüzlerce test girdisi dener. Girdiler başarıyla geçtikçe programın hatasız olduğuna dair güveni artar. Ancak bilgisayar biliminin ve matematiğin kalbinde yatan teorik problemler söz konusu olduğunda, "bin farklı değerde denedim, hepsinde çalıştı" demek asla yeterli değildir. Çünkü incelenen evren çoğunlukla sonsuzdur ve henüz denemediğimiz bir sonraki değer bütün iddiamızı geçersiz kılabilir.

İşte **ispat**, bir iddianın yalnızca denenen birkaç durumda değil, istisnasız **tüm** geçerli durumlarda doğru olduğunu şüpheye yer bırakmayacak mantıksal adımlarla güvenceye alma yöntemidir. İspat yapmak, sağlam kanıtlardan hareketle gerçeğe ulaşan bir araştırmacı gibi çalışmaktır: Başlangıçta kabul edilen tanımlardan ve aksiyomlardan yola çıkılır; Bölüm 1'de kurduğumuz mantık kuralları zinciriyle ilerlenerek hedeflenen sonuca varılır.

Olimpiyat matematiğinde ve algoritmik düşüncede her iddianın arkasındaki "neden?" sorusunu yanıtlamak için dört temel ispat tekniğinden yararlanırız: doğrudan ispat, çelişkiyle ispat, karşı örnekle çürütme ve matematiksel tümevarım.

## 2.1 Doğrudan İspat

Matematiksel iddiaların en yaygın biçimi $p \Rightarrow q$ yapısındaki koşullu önermelerdir. Doğrudan ispat, bu koşulun mantığına en sade ve doğal yaklaşımdır: Hipotez olan $p$'nin doğru olduğunu kabul ederiz; ardından tanımları, bilinen cebirsel kuralları ve daha önce kanıtlanmış gerçekleri ardışık adımlarla uygulayarak hüküm olan $q$'ya ulaşırız.

Doğrudan ispatın mantığını kavramak için Bölüm 11 ve 12'de bölünebilme ve asal sayıları incelerken de sıkça başvuracağımız teklik ve çiftlik tanımlarıyla başlayalım.

**Tanım 2.1** (Çift ve Tek Sayılar)**.** *$n$ bir tam sayı olsun ($n \in \mathbb{Z}$):*

- *Eğer $n = 2k$ olacak biçimde bir $k \in \mathbb{Z}$ tam sayısı varsa, $n$'ye bir* ***çift sayı*** *denir.*
- *Eğer $n = 2k + 1$ olacak biçimde bir $k \in \mathbb{Z}$ tam sayısı varsa, $n$'ye bir* ***tek sayı*** *denir.*

Bu tanımlar yalnızca birer sınıflandırma değil, ispatlarımızda doğrudan yerlerine koyabileceğimiz cebirsel eşitliklerdir.

**Teorem 2.2.** *İki tek sayının toplamı daima bir çift sayıdır.*

*Kanıt.* Adım adım doğrudan ispat zincirini kuralım:

1. **Varsayım (Hipotez):** $x$ ve $y$ iki tek sayı olsun.
2. **Tanımı Uygulama:** Tek sayı tanımı gereği, öyle $a, b \in \mathbb{Z}$ tam sayıları vardır ki:
$$x = 2a + 1 \quad \text{ve} \quad y = 2b + 1$$
3. **Cebirsel İşlem:** Bu iki sayıyı toplayalım:
$$x + y = (2a + 1) + (2b + 1) = 2a + 2b + 2$$
4. **Hedef Formu Yakalama:** Elde ettiğimiz ifadeyi 2 ortak parantezine alabiliriz:
$$x + y = 2(a + b + 1)$$
5. **Sonuç (Hüküm):** $a$ ve $b$ tam sayı olduğundan $k = a + b + 1$ sayısı da bir tam sayıdır. O hâlde $x + y = 2k$ biçiminde yazılabilmektedir. Çift sayı tanımı gereğince $x + y$ zorunlu olarak bir çift sayıdır.

İspat böylece doğrudan tamamlanmış olur. □

### 2.1.1 Karşıt Pozitifle İspat

Bölüm 1'de ele aldığımız temel mantıksal denkliklerden biri, bir koşullu önermenin kendi karşıt pozitifine denk olduğuydu:

$$(p \Rightarrow q) \equiv (\neg q \Rightarrow \neg p)$$

Kimi zaman $p$'den yola çıkıp $q$'ya ulaşmak cebirsel olarak güç veya dolambaçlı olabilir. Böyle durumlarda $\neg q$ ifadesini hipotez kabul edip doğrudan adımlarla $\neg p$'ye ulaşmak çok daha kolaydır. Karşıt pozitifi doğrudan ispatlamak, ana iddiayı ispatlamakla tamamen aynı güce sahiptir.

**Teorem 2.3.** *$n$ bir tam sayı olsun. Eğer $n^2$ çift sayı ise, $n$ de çift sayıdır.*

*Kanıt.* Bu iddiayı doğrudan ispatlamaya çalışalım: "$n^2$ çift ise $n^2 = 2k$'dır; buradan $n = \sqrt{2k}$ olur." Fakat $\sqrt{2k}$ ifadesinin neden bir tam sayı olduğunu ve neden 2'ye bölündüğünü açıklamak köklü ifadeler nedeniyle doğrudan adımlarla kolay değildir.

Bunun yerine önermemizin karşıt pozitifini kuralım:

- $p$: "$n^2$ çifttir", $q$: "$n$ çifttir".
- $\neg q$: "$n$ tektir", $\neg p$: "$n^2$ tektir".

Karşıt pozitif iddiamız şudur: **"Eğer $n$ tek sayı ise, $n^2$ de tek sayıdır."** Şimdi bu iddiayı doğrudan ispatlayalım:

1. $n$ bir tek sayı olsun. O hâlde bir $k \in \mathbb{Z}$ için $n = 2k + 1$'dir.
2. $n^2$'yi hesaplayalım:
$$n^2 = (2k+1)^2 = 4k^2 + 4k + 1 = 2(2k^2 + 2k) + 1$$
3. $k$ tam sayı olduğundan $m = 2k^2 + 2k$ de bir tam sayıdır.
4. Dolayısıyla $n^2 = 2m + 1$ biçimindedir; bu da tanım gereği $n^2$'nin tek sayı olduğunu gösterir.

Karşıt pozitif $(\neg q \Rightarrow \neg p)$ doğrudan kanıtlandığı için, ana teoremimiz $(p \Rightarrow q)$ de ispatlanmış olur. □

### 2.1.2 Çözümlü Örnekler

**Örnek 2.4.** *Ardışık iki tam sayının çarpımının daima bir çift sayı olduğunu doğrudan ispatlayınız.*

*Çözüm.* Herhangi bir $n \in \mathbb{Z}$ tam sayısı alalım. Ardışık iki sayı $n$ ve $n + 1$ olur. Çarpımları $P = n(n + 1)$'dir. Bir tam sayı ya çifttir ya da tektir. Bu iki durumu ayrı ayrı inceleyelim (durum analizi ile doğrudan ispat):

- **1. Durum ($n$ çift olsun):** $n = 2k$ $(k \in \mathbb{Z})$ yazılabilir. Buradan:
$$P = n(n+1) = 2k(2k+1) = 2[k(2k+1)]$$
$k(2k + 1)$ bir tam sayı olduğundan $P$ sayısı 2'nin tam katıdır, yani çifttir.
- **2. Durum ($n$ tek olsun):** $n = 2k + 1$ $(k \in \mathbb{Z})$ yazılabilir. O hâlde bir sonraki sayı:
$$n + 1 = (2k + 1) + 1 = 2k + 2 = 2(k + 1)$$
olur. Çarpım ise:
$$P = n(n+1) = (2k+1) \cdot 2(k+1) = 2[(2k+1)(k+1)]$$
biçimindedir. Yine 2 ortak çarpanı elde edildiğinden $P$ çifttir.

Her iki durumda da $n(n+1)$ çift çıktığından iddia tüm tam sayılar için doğrudur. □

**Örnek 2.5.** *$x$ ve $y$ iki tek tam sayı ise, $x^2+y^2$ toplamının çift olduğunu fakat 4 ile bölünemediğini doğrudan ispatlayınız.*

*Çözüm.* $x$ ve $y$ tek sayı olduklarından $x = 2a+1$ ve $y = 2b+1$ $(a, b \in \mathbb{Z})$ yazabiliriz. Karelerini alıp toplayalım:

$$\begin{aligned} x^2 + y^2 &= (2a+1)^2 + (2b+1)^2 \\ &= (4a^2 + 4a + 1) + (4b^2 + 4b + 1) \\ &= 4a^2 + 4a + 4b^2 + 4b + 2 \end{aligned}$$

Bu ifadeyi iki farklı biçimde gruplayabiliriz:

1. 2 ortak parantezine alırsak:
$$x^2 + y^2 = 2(2a^2 + 2a + 2b^2 + 2b + 1)$$
Bu ifade 2'nin katı olduğundan toplam kesinlikle bir **çift sayı**dır.

2. Şimdi aynı ifadeyi 4 paranteziyle inceleyelim:
$$x^2 + y^2 = 4(a^2 + a + b^2 + b) + 2$$
$a^2+a+b^2+b$ bir tam sayıdır; buna $K$ diyelim. O hâlde $x^2+y^2 = 4K+2$'dir. Bir sayının 4 ile tam bölünebilmesi için kalanın 0 olması gerekir; oysa burada kalan daima 2'dir.

Demek ki iki tek sayının kareleri toplamı çifttir fakat asla 4'e tam bölünemez. □

**Örnek 2.6.** *$n$ bir tam sayı olsun. Eğer $3n+2$ tek sayı ise, $n$'nin tek sayı olduğunu karşıt pozitif yöntemiyle ispatlayınız.*

*Çözüm.* Önermemiz şudur: "$3n+2$ tek $\Rightarrow$ $n$ tek". Bu önermenin karşıt pozitifi:

"$n$ çift sayı ise $3n+2$ çift sayıdır."

Bu yeni iddiayı doğrudan ispatlayalım:

1. $n$ bir çift sayı olsun. O hâlde $n = 2k$ olacak biçimde bir $k \in \mathbb{Z}$ vardır.
2. $3n+2$ ifadesinde $n$ yerine $2k$ yazalım:
$$3n + 2 = 3(2k) + 2 = 6k + 2$$
3. İfadeyi 2 parantezine alalım:
$$3n + 2 = 2(3k + 1)$$
4. $k$ tam sayı olduğundan $3k+1$ de bir tam sayıdır.
5. Dolayısıyla $3n+2$ bir çift sayıdır.

Karşıt pozitif doğru olduğundan, ana iddia da zorunlu olarak doğrudur. □

## 2.2 Çelişkiyle İspat

Bazen bir sonucun neden doğru olduğunu doğrudan inşa etmek güç olabilir; buna karşılık onun **yanlış olduğunu varsaymanın** açık bir mantıksal tutarsızlık doğurduğunu göstermek çok daha pratiktir. Bu yönteme **çelişkiyle ispat** (Latince adıyla *reductio ad absurdum*) denir.

Çelişkiyle ispatın mantıksal adımları şöyledir:

1. İspatlamak istediğimiz önerme $P$ olsun.
2. $P$'nin **yanlış** olduğunu, yani $\neg P$'nin doğru olduğunu varsayarız.
3. Bu varsayımdan yola çıkarak kurala uygun mantıksal çıkarımlar yaparız.
4. En sonunda mantıksal olarak olanaksız bir durumla karşılaşırız: Örneğin bir sayının aynı anda hem tek hem çift çıkması, $0 = 1$ eşitliği ya da bilinen bir teoremin ihlali.
5. Geçerli çıkarım adımları hata üretmeyeceğine göre, bu çelişkinin tek kaynağı en başta yaptığımız "$P$ yanlıştır" varsayımıdır.
6. Dolayısıyla varsayımımız geçersizdir ve $P$ önermesinin **doğru** olduğu kanıtlanmış olur.

Matematik tarihinin en bilinen çelişkiyle ispatlarından biri, antik Yunan'da geliştirilen $\sqrt{2}$'nin irrasyonelliği teoremidir.

**Teorem 2.7.** *$\sqrt{2}$ bir rasyonel sayı değildir (yani irrasyoneldir).*

*Kanıt.* İddianın tersini varsayalım: $\sqrt{2}$ bir **rasyonel sayı** olsun.

1. Rasyonel sayı tanımı gereği, $\sqrt{2}$ sayısı aralarında asal iki pozitif tam sayının oranı olarak yazılabilir:

$$\sqrt{2} = \frac{a}{b} \quad (a, b \in \mathbb{Z}^+)$$

   Buradaki temel kabulümüz, kesrin **en sade** hâlinde olmasıdır; yani $a$ ve $b$'nin 1'den büyük ortak böleni yoktur (özellikle ikisi birden çift sayı olamaz).

2. Eşitliğin her iki tarafının karesini alalım:

$$2 = \frac{a^2}{b^2} \implies a^2 = 2b^2$$

3. $a^2 = 2b^2$ eşitliği, $a^2$'nin bir **çift sayı** olduğunu söyler. Bölüm 2.1'de kanıtladığımız teorem uyarınca, bir tam sayının karesi çift ise kendisi de çifttir. O hâlde $a$ bir çift sayıdır.

4. $a$ çift olduğuna göre, bir $k \in \mathbb{Z}^+$ için $a = 2k$ yazabiliriz. Bunu bir önceki denklemde yerine koyalım:

$$(2k)^2 = 2b^2 \implies 4k^2 = 2b^2 \implies b^2 = 2k^2$$

5. Şimdi bu son eşitliğe bakalım: $b^2 = 2k^2$ olduğuna göre $b^2$ de bir **çift sayı**dır. Yine aynı kuraldan ötürü $b$ de zorunlu olarak bir çift sayı olmalıdır.

6. Elde ettiğimiz sonuçları birleştirelim: Hem $a$'nın hem de $b$'nin çift olduğunu bulduk. İkisi de çift ise, her ikisi de 2'ye tam bölünür.

7. Bu durum, en başta yaptığımız "$a/b$ kesri en sade hâldedir, $a$ ve $b$'nin 1'den büyük ortak böleni yoktur" kabulüyle **çelişir**.

Bu çelişki, $\sqrt{2}$'nin rasyonel olduğu yönündeki varsayımın yanlış olduğunu kanıtlar. Demek ki $\sqrt{2}$ bir rasyonel sayı olamaz; irrasyoneldir. □

### 2.2.1 Çözümlü Örnekler

**Örnek 2.8.** *Toplamları* 100 *olan* 5 *tam sayının en az birinin* 20*'den büyük veya ona eşit olması gerektiğini çelişkiyle ispatlayınız.*

*Çözüm.* Sayılarımız $x_1, x_2, x_3, x_4, x_5 \in \mathbb{Z}$ olsun ve $x_1 + x_2 + x_3 + x_4 + x_5 = 100$ verilsin.

İspatlamak istediğimiz iddia: "$\exists i \in \{1, 2, 3, 4, 5\}, x_i \geq 20$".
Bu iddianın olumsuzunu varsayalım: Sayıların hiçbiri 20'ye eşit veya büyük olmasın. Yani her bir sayı 20'den kesinlikle küçük olsun:

$$\forall i \in \{1, 2, 3, 4, 5\}, \quad x_i < 20 \implies x_i \leq 19$$

Şimdi bu 5 sayının toplamını üstten sınırlandıralım:

$$x_1 + x_2 + x_3 + x_4 + x_5 \leq 19 + 19 + 19 + 19 + 19 = 95$$

Fakat bize verilen başlangıç koşulu toplamın tam olarak 100 olduğuydu:

$$100 \leq 95$$

Bu durum açık bir çelişkidir. Demek ki varsayımımız yanlıştır; sayılardan en az biri 20 veya daha büyük olmak zorundadır. (Bu akıl yürütme, Bölüm 7'de göreceğimiz Güvercin Yuvası İlkesi'nin de temelini oluşturur). □

**Örnek 2.9.** *$r$ bir rasyonel sayı ($r \in \mathbb{Q}$) ve $x$ bir irrasyonel sayı olsun. $r + x$ toplamının bir irrasyonel sayı olduğunu ispatlayınız.*

*Çözüm.* Çelişki yöntemini uygulayalım. Toplamın irrasyonel **olmadığını**, yani bir rasyonel sayı olduğunu varsayalım:

$$r + x = q \quad (q \in \mathbb{Q})$$

Eşitlikte $x$'i yalnız bırakalım:

$$x = q - r$$

$q$ ve $r$ iki rasyonel sayı olduğundan, $q = \frac{a}{b}$ ve $r = \frac{c}{d}$ $(a, c \in \mathbb{Z}$ ve $b, d \in \mathbb{Z}^+)$ biçiminde yazılabilir. Farklarını alalım:

$$x = \frac{a}{b} - \frac{c}{d} = \frac{ad - bc}{bd}$$

Tam sayılar kümesi çarpma ve çıkarma altında kapalı olduğundan $ad - bc$ bir tam sayıdır. Benzer biçimde $bd$ de sıfırdan farklı bir tam sayıdır.

Dolayısıyla $x$, iki tam sayının oranı biçiminde yazılabilmiştir; yani $x \in \mathbb{Q}$ (rasyonel) olmalıdır.

Fakat problemin başında $x$'in bir **irrasyonel** sayı olduğu verilmişti. Bir sayı aynı anda hem rasyonel hem irrasyonel olamaz. Bu çelişki varsayımımızı geçersiz kılar. O hâlde $r + x$ toplamı kesinlikle irrasyoneldir. □

**Örnek 2.10.** *Karesi bir çift sayı olan en küçük pozitif tek sayının var olmadığını çelişkiyle ispatlayınız.*

*Çözüm.* İddia şudur: Karesi çift olan hiçbir pozitif tek sayı yoktur. Tersini varsayalım: Karesi çift olan en az bir $n$ pozitif tek sayısı var olsun.

1. $n$ tek olduğuna göre $n = 2k + 1$ $(k \in \mathbb{N})$ biçimindedir.

2. Karesini alalım:

$$n^2 = (2k + 1)^2 = 4k^2 + 4k + 1 = 2(2k^2 + 2k) + 1$$

3. Bu eşitlik $n^2$'nin kesinlikle bir tek sayı olduğunu gösterir.

4. Ancak varsayımımız $n^2$'nin bir çift sayı olduğuydu.

5. Bir tam sayı aynı anda hem tek hem çift olamaz.

Bu çelişki varsayımı geçersiz kılar. Dolayısıyla karesi çift olan tek bir pozitif tek sayı bile bulunamaz. □

## 2.3 Karşı Örnekle Çürütme

Matematikte genel bir kuralı doğrulamak ile çürütmek arasında belirgin bir asimetri bulunur. Bir iddianın doğruluğunu göstermek için olası tüm durumları kapsayan bir kanıt sunmak gerekirken, evrensel bir iddiayı $(\forall x, P(x))$ çürütmek için kurala uymayan **tek bir örnek** göstermek yeterlidir. Bu örneğe **karşı örnek** (counterexample) denir.

Bölüm 1'de incelediğimiz mantıksal denklik bu durumu açıkça gösterir:

$$\neg(\forall x \in A, P(x)) \equiv \exists x \in A, \neg P(x)$$

Yani "her eleman koşulu sağlar" iddiasının değili, "koşulu sağlamayan en az bir eleman vardır" ifadesidir.

### 2.3.1 Öğrenci Tümevarımı Tuzağı

Yarışmalara hazırlanan öğrencilerin sıkça düştüğü yanılgılardan biri, bir iddianın ilk birkaç değer için sağlandığını görünce onun genel bir kural olduğuna hükmetmektir: "$n = 1$ için doğru, $n = 2$ için doğru, $n = 3$ için doğru; demek ki her $n$ için geçerlidir." Bu çıkarım biçimi matematikte kanıt sayılmaz.

Tarihte önde gelen matematikçiler dahi bu tuzağa dair çarpıcı örnekler bırakmıştır:

- **Euler'in Asal Üreten Polinomu:** Leonhard Euler, $f(n) = n^2 + n + 41$ polinomunu incelemiştir.

$$\begin{aligned} n = 0 &\implies 41 \text{ (asal)} \\ n = 1 &\implies 43 \text{ (asal)} \\ n = 2 &\implies 47 \text{ (asal)} \\ &\dots \\ n = 39 &\implies 39^2 + 39 + 41 = 1601 \text{ (asal)} \end{aligned}$$

  Bu formül $n = 0$'dan $n = 39$'a kadar tam 40 adım boyunca kesintisiz olarak asal sayı üretir. İlk 40 denemeyi gören biri formülün daima asal ürettiğine kolayca inanabilir.

  Fakat $n = 40$ koyduğumuzda:

  $$f(40) = 40^2 + 40 + 41 = 40(40+1) + 41 = 40 \times 41 + 41 = 41 \times (40+1) = 41^2$$

  elde edilir. $41^2 = 1681$ sayısı 41'e tam bölünür ve asal değildir. $n = 40$ tek başına bu genel iddiayı çürüten bir karşı örnektir.

- **Fermat Sayıları:** Pierre de Fermat, $F_n = 2^{2^n} + 1$ biçimindeki sayıların her $n \geq 0$ için daima asal olduğunu öne sürmüştür. İlk birkaç değere bakalım:

$$F_0 = 3, \quad F_1 = 5, \quad F_2 = 17, \quad F_3 = 257, \quad F_4 = 65537$$

Bu sayıların hepsi gerçekten asaldır. Ancak yaklaşık bir asır sonra Euler, $n = 5$ için:

$$F_5 = 2^{32} + 1 = 4294967297 = 641 \times 6700417$$

olduğunu göstererek Fermat'nın iddiasını tek bir karşı örnekle çürütmüştür.

### 2.3.2 Çözümlü Örnekler

**Örnek 2.11.** *Aşağıdaki iddiaların yanlış olduğunu birer karşı örnek göstererek çürütünüz:*

1. *"Bütün tek sayılar asal sayıdır."*
2. *"İki irrasyonel sayının toplamı daima irrasyoneldir."*
3. *"Herhangi $a, b, c \in \mathbb{R}$ için $a \cdot b = a \cdot c$ ise $b = c$'dir."*

*Çözüm.* Her iddiayı tek bir karşı örnekle geçersiz kılalım:

1. İddia: $\forall n \in \mathbb{Z}, (n \text{ tek} \Rightarrow n \text{ asal})$.
   **Karşı örnek:** $n = 9$ seçelim. $9 = 2 \times 4 + 1$ olduğundan bir tek sayıdır; ancak $9 = 3 \times 3$ olduğundan asal değildir. $n = 9$ iddiayı çürütmeye yeterlidir.

2. İddia: $x, y \notin \mathbb{Q} \Rightarrow x + y \notin \mathbb{Q}$.
   **Karşı örnek:** $x = \sqrt{2}$ ve $y = -\sqrt{2}$ seçelim. Her ikisi de irrasyoneldir. Fakat toplamları:
   $$x + y = \sqrt{2} + (-\sqrt{2}) = 0$$
   olur. $0 = \frac{0}{1}$ bir rasyonel sayıdır. Dolayısıyla iddia genel olarak doğru değildir.

3. İddia: $a \cdot b = a \cdot c \Rightarrow b = c$.
   **Karşı örnek:** $a = 0, b = 2, c = 5$ seçelim.
   $$a \cdot b = 0 \cdot 2 = 0 \quad \text{ve} \quad a \cdot c = 0 \cdot 5 = 0$$
   Burada $a \cdot b = a \cdot c$ eşitliği sağlanır; ancak $2 \neq 5$'tir ($b \neq c$). Sıfır çarpanı iddiayı geçersiz kılar.

□

**Örnek 2.12.** *Bir öğrenci "n bir pozitif tam sayı ise $n^2 - n + 11$ sayısı daima asaldır" iddiasında bulunmaktadır. Bu iddiayı çürüten en küçük n pozitif tam sayısını bulunuz.*

*Çözüm.* İfademiz $f(n) = n^2 - n + 11$'dir. Küçük değerleri inceleyelim:

- $n = 1 \implies 1 - 1 + 11 = 11$ (asal)

- $n = 2 \implies 4 - 2 + 11 = 13$ (asal)
- $n = 3 \implies 9 - 3 + 11 = 17$ (asal)
- $n = 4 \implies 16 - 4 + 11 = 23$ (asal)
- $n = 5 \implies 25 - 5 + 11 = 31$ (asal)

Denemelere devam etmek yerine ifadenin cebirsel yapısına bakalım: İfadenin 11'in katı olmasını nasıl sağlarız? Eğer $n^2 - n$ kısmı 11'in bir katı olursa, toplamın tamamı 11'e bölünür ve sayı asal olmaktan çıkar:

$$n^2 - n = n(n-1)$$

Eğer $n = 11$ seçersek:

$$f(11) = 11^2 - 11 + 11 = 11^2 = 121$$

elde edilir. $121 = 11 \times 11$ olduğundan asal değildir.

Peki 11'den daha küçük bir karşı örnek var mıdır? $n(n-1)$ çarpımının 11'e bölünmesi için $n - 1 = 11 \implies n = 12$ gerekir. $n = 1, \ldots, 10$ değerleri için $n^2 - n + 11$ asal kalır. O hâlde iddiayı çürüten en küçük pozitif tam sayı $n = 11$'dir. □

## 2.4 Tümevarım ve Toplam Gösterimi

Doğal sayılar üzerine kurulu iddiaları ($\forall n \in \mathbb{N}^+, P(n)$) ispatlamanın en etkili ve sistematik yolu **matematiksel tümevarım** (mathematical induction) yöntemidir.

Tümevarımın çalışma mantığını yan yana dizilmiş sonsuz sayıda domino taşı üzerinden düşünebiliriz:

1. **Taban Adımı (Base Case):** İlk domino taşını devirebiliyor muyuz? Yani $P(1)$ doğru mudur?
2. **Tümevarım Adımı (Inductive Step):** Herhangi bir $k$. taş devrildiğinde, bir sonraki $(k+1)$. taşı da mutlaka deviriyor mu? Yani $P(k) \Rightarrow P(k+1)$ gerektirmesi genel olarak geçerli midir?

Bu iki koşul sağlandığında, ilk taş devrildiğinde ikinciyi, ikinci üçüncüyü devirecek ve zincirleme bir biçimde tüm taşlar devrilecektir.

**Teorem 2.13** (Matematiksel Tümevarım İlkesi)**.** *$P(n)$, pozitif tam sayılar ($n \in \mathbb{Z}^+$) üzerinde tanımlı bir açık önerme olsun. Eğer:*

1. *$P(1)$ doğru ise, ve*
2. *Herhangi bir $k \geq 1$ tam sayısı için $P(k)$'nin doğruluğu $P(k+1)$'in doğruluğunu gerektiriyorsa ($P(k) \Rightarrow P(k+1)$),*

*o hâlde her $n \in \mathbb{Z}^+$ için $P(n)$ doğrudur.*

### 2.4.1 Toplam Gösterimi ($\sum$)

Ardışık terimlerin toplamını uzun uzadıya $a_1 + a_2 + \cdots + a_n$ biçiminde yazmak yerine matematikte büyük Yunan harfi Sigma ($\sum$) ile gösterilen **toplam sembolü** kullanılır:

$$\sum_{i=1}^{n} a_i = a_1 + a_2 + a_3 + \cdots + a_n$$

Burada $i$ değişkenine **toplam indisi** denir; 1 indisin başlangıç değerini, $n$ ise bitiş değerini belirtir.

Tümevarımın en bilinen uygulamalarından biri olan Gauss toplam formülünü ele alalım.

**Teorem 2.14** (Gauss Toplam Formülü)**.** *İlk $n$ pozitif tam sayının toplamı:*

$$\sum_{i=1}^{n} i = 1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$$

*formülüyle verilir.*

*Kanıt.* Tümevarım ilkesinin iki adımını uygulayalım:

**1. Taban Adımı ($n = 1$):** Sol taraf: Yalnızca ilk terim vardır, yani 1. Sağ taraf: Formülde $n = 1$ yazalım:

$$\frac{1 \cdot (1+1)}{2} = \frac{1 \cdot 2}{2} = 1$$

Sol taraf ile sağ taraf birbirine eşittir ($1 = 1$). Taban adımı doğrulanmıştır.

**2. Tümevarım Adımı:** Bir $k \geq 1$ tam sayısı için formülün doğru olduğunu varsayalım (**tümevarım hipotezi**):

$$1 + 2 + \cdots + k = \frac{k(k+1)}{2}$$

Amacımız, bu hipotezi kullanarak eşitliğin $n = k + 1$ için de sağlandığını, yani:

$$1 + 2 + \cdots + k + (k+1) = \frac{(k+1)((k+1)+1)}{2} = \frac{(k+1)(k+2)}{2}$$

sonucuna vardığını göstermektir.

$k + 1$ terimli toplamın sol tarafını yazalım ve ilk $k$ terimi tümevarım hipotezimizle değiştirelim:

$$\underbrace{1 + 2 + \cdots + k}_{=\frac{k(k+1)}{2}} + (k+1) = \frac{k(k+1)}{2} + (k+1)$$

Ortak payda oluşturalım:

$$= \frac{k(k+1) + 2(k+1)}{2}$$

Pay kısmını $(k+1)$ ortak parantezine alalım:

$$= \frac{(k+1)(k+2)}{2}$$

Bu, $n = k+1$ için elde etmek istediğimiz ifadedir.

Taban adımı ve tümevarım adımı sağlandığından, matematiksel tümevarım ilkesi gereğince formül tüm $n \geq 1$ tam sayıları için geçerlidir. □

### 2.4.2 Çözümlü Örnekler

**Örnek 2.15.** *İlk $n$ pozitif tek tam sayının toplamının $n^2$ olduğunu, yani:*

$$\sum_{i=1}^{n}(2i-1) = 1 + 3 + 5 + \cdots + (2n-1) = n^2$$

*eşitliğini tümevarım yöntemiyle ispatlayınız.*

*Çözüm.* **1. Taban Adımı ($n = 1$):** Sol taraf: $2(1) - 1 = 1$. Sağ taraf: $1^2 = 1$. Eşitlik $n = 1$ için doğrudur.

**2. Tümevarım Adımı:** İddianın $n = k$ için doğru olduğunu varsayalım:

$$1 + 3 + 5 + \cdots + (2k-1) = k^2$$

$n = k+1$ durumunu inceleyelim. Bu durumda toplama bir sonraki tek sayı olan $2(k+1) - 1 = 2k + 1$ eklenir:

$$\underbrace{1 + 3 + 5 + \cdots + (2k-1)}_{=k^2} + (2k+1) = k^2 + 2k + 1$$

Cebirsel tam kare açılımından biliyoruz ki:

$$k^2 + 2k + 1 = (k+1)^2$$

Bu da formülün $n = k+1$ için öngördüğü değerdir.

Tümevarım tamamlanmıştır; ilk $n$ tek sayının toplamı daima $n^2$'dir. □

**Örnek 2.16.** *Her $n \geq 1$ tam sayısı için $3^{2n} - 1$ sayısının 8 ile tam bölündüğünü tümevarımla ispatlayınız.*

*Çözüm.* **1. Taban Adımı ($n = 1$):**

$$3^{2(1)} - 1 = 3^2 - 1 = 9 - 1 = 8$$

8 sayısı 8'e tam bölünür ($8 = 8 \times 1$). Taban adımı sağlanır.

**2. Tümevarım Adımı:** $n = k$ için ifadenin 8'e bölündüğünü varsayalım; yani bir $m \in \mathbb{Z}$ için:

$$3^{2k} - 1 = 8m \implies 3^{2k} = 8m + 1$$

olsun. Şimdi $n = k + 1$ için ifadeyi hesaplayalım:

$$3^{2(k+1)} - 1 = 3^{2k+2} - 1 = 3^{2k} \cdot 3^2 - 1 = 9 \cdot 3^{2k} - 1$$

Tümevarım hipotezimiz olan $3^{2k} = 8m + 1$ değerini yerine yazalım:

$$\begin{aligned} 9 \cdot (8m + 1) - 1 &= 72m + 9 - 1 \\ &= 72m + 8 \\ &= 8(9m + 1) \end{aligned}$$

$m$ bir tam sayı olduğundan $9m + 1$ de bir tam sayıdır. Elde edilen ifade 8'in katı olduğundan, $n = k + 1$ için de 8 ile tam bölünür.

Tümevarım ilkesi uyarınca iddia her $n \geq 1$ için doğrudur. □

**Örnek 2.17.** *Her $n \geq 4$ tam sayısı için $2^n \geq n^2$ eşitsizliğinin sağlandığını tümevarımla ispatlayınız.*

*Çözüm.* Burada taban adımımızın 1 değil 4 olduğuna dikkat ediniz ($n = 3$ için $2^3 = 8 < 3^2 = 9$ olurdu).

**1. Taban Adımı ($n = 4$):** Sol taraf: $2^4 = 16$. Sağ taraf: $4^2 = 16$. $16 \geq 16$ sağlandığından taban adımı geçerlidir.

**2. Tümevarım Adımı:** $k \geq 4$ için $2^k \geq k^2$ olduğunu varsayalım. $n = k + 1$ için $2^{k+1} \geq (k + 1)^2$ olduğunu göstermek istiyoruz. Sol taraftan başlayalım:

$$2^{k+1} = 2 \cdot 2^k \geq 2k^2 \quad \text{(tümevarım hipotezinden)}$$

Şimdi $2k^2 \geq (k+1)^2$ olduğunu gösterebilirsek zincir tamamlanacaktır. Farkı inceleyelim:

$$2k^2 - (k + 1)^2 = 2k^2 - (k^2 + 2k + 1) = k^2 - 2k - 1 = k(k - 2) - 1$$

$k \geq 4$ olduğunu biliyoruz. O hâlde:

$$k(k - 2) - 1 \geq 4(4 - 2) - 1 = 4(2) - 1 = 7 > 0$$

Fark pozitif olduğuna göre $2k^2 \geq (k + 1)^2$'dir. Buradan:

$$2^{k+1} \geq 2k^2 \geq (k + 1)^2$$

elde edilir. İspat tamamlanmıştır. □

## 2.5 İspatlı Küçük Örnekler

Öğrendiğimiz ispat tekniklerini yarışmalarda karşılaşılan klasik problemlere ve algoritmik düşünceye uygulayalım.

### 2.5.1 Rekürsiyon ve Tümevarım Kardeşliği

Bilgisayar biliminde bir problemin çözümünü kendi alt problemlerine indirgeyerek tanımlamaya **rekürsiyon** (recursion) denir (bu kavram Bölüm 26'da ayrıntılı biçimde ele alınacaktır). Rekürsiyon ile matematiksel tümevarım birbirini doğrudan tamamlar:

- Tümevarım çözümü tabandan yukarıya doğru inşa eder ($1 \to 2 \to 3 \dots$).
- Rekürsiyon ise problemi adım adım taban duruma (*base case*) indirger.

Aşağıdaki C kodunu ele alalım:

```c
// n pozitif tam sayisinin faktoriyelini hesaplar
int faktoriyel(int n) {
    if (n <= 1) {
        return 1; // Taban adimi (base case)
    }
    return n * faktoriyel(n - 1); // Ozyinelemeli adim
}
```

Bu fonksiyonun her $n \geq 1$ için doğru çalıştığının doğrulaması doğrudan matematiksel tümevarımla yapılır:

1. $n = 1$ için kod `if` bloğuna girer ve 1 döndürür. $1! = 1$ olduğundan taban adımı geçerlidir.
2. Fonksiyonun $k$ için doğru çalıştığını varsayarsak (`faktoriyel(k)` $= k!$), $k+1$ çağrıldığında `(k+1) * faktoriyel(k)` $= (k+1) \cdot k! = (k+1)!$ değerini üretecektir. Tümevarım adımı tamamlanır.

### 2.5.2 Klasik Çözümlü Problemler

**Örnek 2.18** (L-Tromino ile Tahta Döşeme)**.** *Bir kenarı $2^n$ birim olan ($2^n \times 2^n$) kare bir satranç tahtasından rastgele **tek bir birim kare** kesilip çıkartılıyor. Geriye kalan $2^{2n}-1$ adet birim karenin, her biri 3 birim kareden oluşan L biçimindeki taşlarla (L-tromino) boşluksuz ve taşma olmadan tamamen döşenebileceğini her $n \geq 1$ için ispatlayınız.*

*Çözüm.* Bu geometri problemini matematiksel tümevarımla çözebiliriz:

**1. Taban Adımı ($n = 1$):** Tahtamız $2^1 \times 2^1 = 2 \times 2$ boyutundadır ve toplam 4 karesi vardır. Herhangi 1 kare çıkarıldığında geriye tam olarak 3 kare kalır. Bu

3 kare kendiliğinden tam bir $L$-tromino şeklindedir. Dolayısıyla 1 taş ile tahta döşenir. Taban adımı sağlanır.

**2. Tümevarım Adımı:** İddianın $n = k$ için doğru olduğunu kabul edelim: 1 karesi eksik herhangi bir $2^k \times 2^k$ tahta $L$-trominolarla döşenebilsin.

Şimdi $2^{k+1} \times 2^{k+1}$ boyutundaki tahtayı ele alalım. Bu tahtayı tam ortadan yatay ve dikey çizgilerle bölerek **dört adet** $2^k \times 2^k$ **boyutunda çeyrek tahtaya** ayıralım.

Eksik olan tek kare, bu dört çeyrekten **yalnızca birinin** içinde yer alır.

- Eksik kareyi barındıran çeyrek tahta, tümevarım hipotezimiz gereğince tek başına döşenebilir.
- Eksik karesi olmayan diğer üç çeyrek tahtaya gelince: Büyük tahtanın tam merkezine, eksik karesi **olmayan** üç çeyreğin kesiştiği köşeleri örtecek biçimde tek bir $L$-tromino yerleştirelim.
- Bu hamleyle, diğer üç çeyrek tahtanın da merkezdeki birer karesi örtülmüş olur. Artık o üç çeyrek tahta da "birer karesi eksilmiş $2^k \times 2^k$ tahtalara" dönüşür.
- Tümevarım hipotezimize göre bu üç çeyrek de bağımsız olarak $L$-trominolarla tamamen döşenebilir.

Böylece $2^{k+1} \times 2^{k+1}$ tahtanın tamamı döşenmiş olur. Tümevarım ilkesi gereği iddia her $n \geq 1$ için geçerlidir. □

**Örnek 2.19.** *Ardışık üç tam sayının toplamının daima* 3 *ile tam bölündüğünü doğrudan ispatlayınız.*

*Çözüm.* Ardışık üç tam sayıyı temsil etmenin iki yolu vardır:

- Birinci yol: $n, n+1, n+2$. Toplamları:

$$n + (n+1) + (n+2) = 3n + 3 = 3(n+1)$$

- İkinci (daha simetrik) yol: Ortadaki sayıya $m$ dersek sayılar $m-1, m, m+1$ olur. Toplamları:

$$(m-1) + m + (m+1) = 3m$$

Her iki durumda da toplamın 3 ortak çarpanına sahip olduğu görülür. Sayılar tam sayı olduğundan toplam daima 3'ün tam katıdır. □

**Örnek 2.20.** *Bir çember etrafına* $1, 2, 3, \ldots, 6$ *sayıları rastgele bir sırayla dizilmiştir. Çember üzerinde yan yana bulunan ardışık üç sayının toplamının en az* 11 *olması gerektiğini çelişki yöntemiyle ispatlayınız.*

*Çözüm.* Çember üzerindeki sayıları saat yönünde sırasıyla $a_1, a_2, a_3, a_4, a_5, a_6$ olarak adlandıralım. Bölüm 4'te ele alacağımız permütasyon kavramıyla ifade edersek, bu sayılar $1, 2, 3, 4, 5, 6$'nın bir dizilimidir; dolayısıyla toplamları:

$$\sum_{i=1}^{6} a_i = 1 + 2 + 3 + 4 + 5 + 6 = 21$$

olur.

Çemberi ardışık üçerli iki ayrık gruba ayıralım:

$$S_1 = a_1 + a_2 + a_3 \quad \text{ve} \quad S_2 = a_4 + a_5 + a_6$$

Bu iki grubun toplamı tüm sayıları kapsar:

$$S_1 + S_2 = (a_1 + a_2 + a_3) + (a_4 + a_5 + a_6) = 21$$

Şimdi iddiamızın tersini varsayalım: Çember üzerindeki ardışık hiçbir üç sayının toplamı 11 veya daha fazla olmasın. Yani yan yana her üç sayının toplamı en fazla 10 olsun:

$$S_1 \leq 10 \quad \text{ve} \quad S_2 \leq 10$$

Bu iki eşitsizliği taraf tarafa toplayalım:

$$S_1 + S_2 \leq 10 + 10 = 20$$

Fakat biz $S_1 + S_2 = 21$ olduğunu biliyoruz:

$$21 \leq 20$$

Bu bir çelişkidir. O hâlde varsayımımız yanlıştır; çember üzerindeki ardışık üç sayının toplamı en az bir yerde 11'e eşit veya 11'den büyük olmak zorundadır. □

# Bölüm 3

# Saymanın Temel İlkeleri

Bir algoritmanın çalışma süresini analiz ederken, bir şifrenin güvenliğini değerlendirirken ya da bir veri yapısındaki olası durumları incelerken temel bir soruyla karşılaşırız: "Burada kaç farklı durum vardır?"

Kombinatorik, sonlu ya da ayrık yapıların eleman sayılarını belirleyen matematik dalıdır. İlk bakışta saymak, nesneleri birer birer göstermek kadar yalın bir eylem gibi görünebilir. Ancak incelenen nesnelerin sayısı binleri, milyonları ya da milyarları aştığında nesneleri tek tek listelemek mümkün olmaz. Örneğin 64 karelik bir satranç tahtasına taşların dizilme biçimlerinin sayısı evrendeki atomların sayısından çok daha fazladır.

Bu nedenle nesneleri tek tek saymak yerine, kurallara dayalı sistematik sayma yöntemleri kullanılır. Kombinatorik, temelde iki ilke üzerine kuruludur: **çarpma kuralı** ve **toplama kuralı**. Bu bölümde, sayma kuramının bu iki temel yapı taşını somut örneklerle ele alıp genel formüllerine ulaşacağız.

## 3.1 Çarpma Kuralı

Gündelik bir problemle başlayalım: Gardırobunuzda 3 farklı gömlek (Mavi, Beyaz, Çizgili) ve 4 farklı pantolon (Siyah, Gri, Kot, Bej) bulunuyor. Bir gömlek **ve** bir pantolondan oluşan bir kıyafeti kaç farklı şekilde seçebilirsiniz?

Seçenekleri listeleyelim:

- Mavi gömlek seçilirse yanına 4 farklı pantolondan biri giyilebilir: (Mavi, Siyah), (Mavi, Gri), (Mavi, Kot), (Mavi, Bej) $\rightarrow$ 4 seçenek.
- Beyaz gömlek seçilirse yine 4 farklı pantolon seçeneği vardır: (Beyaz, Siyah), (Beyaz, Gri), (Beyaz, Kot), (Beyaz, Bej) $\rightarrow$ 4 seçenek.
- Çizgili gömlek seçilirse yine 4 farklı pantolon seçeneği vardır: (Çizgili, Siyah), (Çizgili, Gri), (Çizgili, Kot), (Çizgili, Bej) $\rightarrow$ 4 seçenek.

Her gömlek seçimi için pantolon seçeneklerinin sayısı değişmez ve her biri için 4 yeni durum ortaya çıkar. Toplam seçenek sayısı:

$$4 + 4 + 4 = 3 \times 4 = 12$$

olur. Bu kombine bir de 2 farklı ayakkabı çifti (Spor, Klasik) eklenseydi, bu 12 durumun her biri için 2 farklı ayakkabı seçilebilecek ve toplam seçenek sayısı $12 \times 2 = 24$ olacaktı.

Bu gözlem bizi saymanın ilk temel ilkesine götürür:

**Tanım 3.1** (Çarpma Kuralı / Bağımsız Adımlar İlkesi)**.** *Bir işlem birbiri ardına gerçekleştirilen $k$ adımdan oluşsun:*

- *1. adım $n_1$ farklı yolla yapılabiliyorsa,*
- *1. adım nasıl yapılırsa yapılsın, 2. adım $n_2$ farklı yolla yapılabiliyorsa,*
- *1. ve 2. adımlar nasıl yapılırsa yapılsın, 3. adım $n_3$ farklı yolla yapılabiliyorsa,*
- *…*
- *İlk $(k-1)$ adım nasıl yapılırsa yapılsın, sonuncu $k$. adım $n_k$ farklı yolla yapılabiliyorsa;*

*bu işlemin tamamı sırasıyla*

$$n_1 \times n_2 \times n_3 \times \cdots \times n_k$$

*farklı yolla gerçekleştirilebilir.*

### 3.1.1 Kümeler Kuramı Açısından Çarpma Kuralı

Bölüm 1'de kurduğumuz küme kavramlarını hatırlayalım. $A$ ve $B$ iki sonlu küme olsun. Bu iki kümenin elemanlarından oluşturulan tüm sıralı ikililerin $((a, b))$ kümesine $A$ ile $B$'nin **Kartezyen çarpımı** denir ve $A \times B$ ile gösterilir:

$$A \times B = \{(a, b) \mid a \in A \text{ ve } b \in B\}$$

**Teorem 3.2.** *$A$ ve $B$ sonlu iki küme olmak üzere, Kartezyen çarpımın eleman sayısı kümelerin eleman sayılarının çarpımına eşittir:*

$$|A \times B| = |A| \cdot |B|$$

*Kanıt.* $A = \{a_1, a_2, \ldots, a_m\}$ ve $B = \{b_1, b_2, \ldots, b_n\}$ olsun. Dolayısıyla $|A| = m$ ve $|B| = n$'dir. $A \times B$'nin tüm elemanlarını satır ve sütunlardan oluşan bir tablo biçiminde düzenleyelim:

| | $b_1$ | $b_2$ | $\dots$ | $b_n$ |
|---|---|---|---|---|
| $a_1$ | $(a_1, b_1)$ | $(a_1, b_2)$ | $\dots$ | $(a_1, b_n)$ |
| $a_2$ | $(a_2, b_1)$ | $(a_2, b_2)$ | $\dots$ | $(a_2, b_n)$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\ddots$ | $\vdots$ |
| $a_m$ | $(a_m, b_1)$ | $(a_m, b_2)$ | $\dots$ | $(a_m, b_n)$ |

Bu tablonun $m$ satırı ve her satırında $n$ elemanı vardır. Tablodaki hiçbir eleman birbiriyle aynı değildir ve olası tüm $(a_i, b_j)$ çiftleri bu tabloda yer alır. O hâlde toplam eleman sayısı satır sayısı ile sütun sayısının çarpımıdır:

$$|A \times B| = m \cdot n = |A| \cdot |B|$$

Böylece eşitlik gösterilmiş olur. □

**Önemli Bir Ayrım:** Çarpma kuralında adımların birbirinden bağımsız olması, her adımda seçilen nesnelerin aynı kümeden gelmesi gerektiği anlamına gelmez. Önemli olan, **seçenek sayısının sabit kalmasıdır**. Örneğin 5 kişi arasından bir başkan ve bir başkan yardımcısı seçecek olalım:

- Başkan için 5 aday vardır ($n_1 = 5$).
- Başkan kim seçilirse seçilsin, seçilen kişi başkan yardımcısı olamayacağından geriye 4 aday kalır ($n_2 = 4$).
- İkinci adımdaki adayların kimler olduğu birinci adımdaki seçime bağlıdır; ancak adayların **sayısı** her durumda 4'tür.
- Dolayısıyla çarpma kuralı geçerlidir: Toplam $5 \times 4 = 20$ farklı seçim yapılabilir.

### 3.1.2 Çözümlü Örnekler

**Örnek 3.3.** $\{1, 2, 3, 4, 5\}$ *kümesinin elemanları kullanılarak:*

1. *Üç basamaklı kaç farklı doğal sayı yazılabilir?*
2. *Rakamları birbirinden farklı üç basamaklı kaç farklı doğal sayı yazılabilir?*

*Çözüm.* Üç basamaklı bir sayıyı oluşturmak, yüzler, onlar ve birler basamağını belirlemekten oluşan 3 adımlı bir işlemdir.

1. **Rakam tekrarı serbestken:**
   - Yüzler basamağı için $\{1, 2, 3, 4, 5\}$ kümesinden 5 seçenek vardır.
   - Onlar basamağı için yine 5 seçenek vardır (kullanılan rakam tekrar seçilebilir).

- Birler basamağı için yine 5 seçenek vardır.

Çarpma kuralı uyarınca:

$$5 \times 5 \times 5 = 5^3 = 125$$

farklı sayı yazılabilir.

2. **Rakamları farklıyken:**

   - Yüzler basamağı için 5 seçenekten biri seçilir.
   - Onlar basamağı için, yüzler basamağında kullanılan rakam hariç kalan 4 seçenekten biri seçilir.
   - Birler basamağı için, ilk iki basamakta kullanılan rakamlar hariç kalan 3 seçenekten biri seçilir.

   Çarpma kuralı uyarınca:

$$5 \times 4 \times 3 = 60$$

   farklı sayı yazılabilir.

□

**Örnek 3.4.** *Bilgisayar biliminde temel bilgi birimi bit'tir (*0 *veya* 1*). Sekiz bitten oluşan bir dizgiye bir* ***bayt*** *(byte) denir (örneğin* `10110010`*). Bir bayt ile kaç farklı bilgi (karakter ya da sayı) temsil edilebilir?*

*Çözüm.* Bir bayt oluşturmak, art arda 8 pozisyonun her birine bir bit (0 veya 1) yerleştirmek anlamına gelir:

$$\_ \quad \_ \quad \_ \quad \_ \quad \_ \quad \_ \quad \_ \quad \_$$

Her pozisyon için 2 farklı seçenek vardır (0 veya 1):

- 1. bit: 2 seçenek
- 2. bit: 2 seçenek
- …
- 8. bit: 2 seçenek

Çarpma kuralı gereğince toplam farklı dizgi sayısı:

$$\underbrace{2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2}_{8 \text{ adım}} = 2^8 = 256$$

olur. Bu nedenle 1 baytlık bellekte 0 ile 255 arasındaki tam sayılar ya da standart ASCII tablosundaki 256 farklı karakter saklanabilir. □

**Örnek 3.5.** 4 *farklı mektup,* 3 *farklı posta kutusuna her kutuda istenen sayıda mektup bulunabilecek şekilde kaç farklı biçimde atılabilir?*

*Çözüm.* Bu tip sorularda sık karşılaşılan tereddüt, cevabın $3^4$ mü yoksa $4^3$ mü olduğudur. Bu ayrımı yapmanın pratik yolu şudur: **"Kararı veren kimdir?"**

- Posta kutuları mektupları seçemez; kutu seçim yapmaya kalkarsa hangi mektubun hangi kutuya gideceği belirsizleşir.
- Karar mektuplar üzerinden verilir: Her mektup sırayla ele alınır ve hangi kutuya atılacağı belirlenir.

Adımları tanımlayalım:

- 1. mektup için 3 farklı kutu seçeneği vardır.
- 2. mektup için 3 farklı kutu seçeneği vardır.
- 3. mektup için 3 farklı kutu seçeneği vardır.
- 4. mektup için 3 farklı kutu seçeneği vardır.

Her mektubun seçimi bağımsız olduğundan çarpma kuralı uygulanır:

$$3 \times 3 \times 3 \times 3 = 3^4 = 81$$

farklı dağıtım yapılabilir. □

**Örnek 3.6.** *A şehrinden B şehrine* 3 *farklı kara yolu, B şehrinden C şehrine ise* 4 *farklı kara yolu bulunmaktadır.*

1. *A'dan C'ye B üzerinden kaç farklı yoldan gidilebilir?*
2. *A'dan C'ye gidip tekrar A'ya dönecek olan bir sürücü, dönerken gittiği yolları* ***aynen kullanmamak*** *şartıyla kaç farklı gidiş-dönüş rotası izleyebilir?*

*Çözüm.*

1. $A$'dan $C$'ye gitmek 2 adımlı bir işlemdir: Önce $A \to B$ (3 seçenek), ardından $B \to C$ (4 seçenek). Çarpma kuralı uyarınca:

   $$3 \times 4 = 12$$

   farklı rota vardır.

2. Gidiş-dönüş işlemi toplam 4 adımdan oluşur:
   - $A \to B$ gidiş: 3 yol seçeneği.
   - $B \to C$ gidiş: 4 yol seçeneği.
   - $C \to B$ dönüş: Giderken kullanılan yol kullanılamayacağından geriye kalan $4 - 1 = 3$ seçenek.

- $B \to A$ dönüş: Giderken kullanılan yol kullanılamayacağından geriye kalan $3 - 1 = 2$ seçenek.

Çarpma kuralı gereğince toplam rota sayısı:

$$3 \times 4 \times 3 \times 2 = 72$$

farklı biçimde belirlenebilir.

□

## 3.2 Toplama Kuralı

Çarpma kuralında adımlar birbirini izliyordu ("önce bunu yap **ve** sonra şunu yap"). Seçenekler ardışık adımlar yerine birbirinin alternatifi olan ayrık yollardan oluşuyorsa toplama kuralı devreye girer.

Örneğin Ankara'dan İstanbul'a seyahat etmek istediğinizi varsayalım. İki ulaşım türü bulunmaktadır: Uçak veya hızlı tren. Gün içinde 3 farklı uçak seferi ve 5 farklı hızlı tren seferi vardır. Bu yolculuk için kaç farklı sefer seçeneği bulunur?

Burada bir uçak seferi ve bir tren seferi aynı anda seçilemez; ya bir uçak seferi **veya** bir tren seferi seçilir. Bu iki seçenek grubu birbirini dışlar (ayrıktır). Dolayısıyla toplam seçenek sayısı:

$$3 + 5 = 8$$

olur. Bu mantık **toplama kuralı**dır.

**Tanım 3.7** (Toplama Kuralı / Ayrık Durumlar İlkesi)**.** *Bir işlem, birbirini dışlayan $k$ farklı durumdan biriyle yapılabilsin:*

- *1. durum $n_1$ farklı yolla,*
- *2. durum $n_2$ farklı yolla,*
- *…*
- *$k$. durum $n_k$ farklı yolla gerçekleşebiliyorsa;*

*ve bu durumların hiçbiri aynı anda gerçekleşemiyorsa (yani durumlar ikili olarak ayrıksa), bu işlem toplam*

$$n_1 + n_2 + \cdots + n_k$$

*farklı yolla yapılabilir.*

### 3.2.1 Kümeler Kuramı Açısından Toplama Kuralı

Bölüm 1'de ayrık kümeleri $A \cap B = \emptyset$ olarak tanımlamıştık. Toplama kuralı, ayrık kümelerin birleşiminin eleman sayısını verir.

**Teorem 3.8.** *$A_1, A_2, \ldots, A_k$ sonlu kümeler olsun. Bu kümeler ikili olarak ayrıksa (yani her $i \neq j$ için $A_i \cap A_j = \emptyset$), birleşimlerinin eleman sayısı kümelerin eleman sayıları toplamına eşittir:*

$$|A_1 \cup A_2 \cup \cdots \cup A_k| = |A_1| + |A_2| + \cdots + |A_k|$$

*Kanıt.* Kümeler ikili olarak ayrık olduğundan, hiçbir eleman birden fazla kümede yer almaz. Dolayısıyla birleşimdeki tüm elemanları saymak, her kümenin elemanlarını bağımsız olarak sayıp bu değerleri toplamaktan ibarettir. □

**Önemli Bir Uyarı (Ayrık Olmama Durumu):** Toplama kuralını doğrudan uygulayabilmek için durumların **kesişimsiz** olması şarttır. İki kümenin ortak elemanları varsa ($A \cap B \neq \emptyset$), doğrudan toplama işlemi ortak elemanların ikişer kez sayılmasına yol açar.

Bölüm 1'de gördüğümüz gibi genel durumda:

$$|A \cup B| = |A| + |B| - |A \cap B|$$

bağıntısı geçerlidir. İncelenen durumların ayrık olduğundan emin olunmadıkça doğrudan toplama yapılamaz; kesişim kümesinin çıkarılması gerekir (bu yaklaşım Bölüm 8'de İçerme-Dışarma İlkesi başlığı altında ayrıntılı biçimde incelenecektir).

### 3.2.2 Çözümlü Örnekler

**Örnek 3.9.** *Bir öğrencinin kitaplığında 6 farklı matematik, 4 farklı fizik ve 5 farklı kimya kitabı bulunmaktadır. Bu öğrenci masasına koyup çalışmak üzere bu kitaplar arasından **tek bir kitap** seçecektir. Kaç farklı seçim yapabilir?*

*Çözüm.* Öğrenci tek bir kitap seçecektir. Bu kitap:

- Ya bir matematik kitabı (6 seçenek),
- Ya bir fizik kitabı (4 seçenek),
- Ya da bir kimya kitabı (5 seçenek) olur.

Bir kitap aynı anda birden fazla kategoriye ait olamayacağından bu seçenekler ikili olarak ayrıktır. Toplama kuralı uyarınca:

$$6 + 4 + 5 = 15$$

farklı kitap seçimi yapılabilir. □

**Örnek 3.10.** *İki standart zar birlikte atılıyor. Zarların üst yüzüne gelen sayıların toplamının* 7 *veya* 11 *olma durumlarının sayısını bulunuz.*

*Çözüm.* Zarlar atıldığında toplam aynı anda hem 7 hem de 11 olamaz; bu iki durum birbirini dışlar. Durumları ayrı ayrı ele alalım:

- **1. Durum (Toplamın 7 olması):** Zarların gelme çiftlerini $(z_1, z_2)$ olarak listeleyelim:

$$(1,6),(2,5),(3,4),(4,3),(5,2),(6,1) \implies 6 \text{ farklı durum.}$$

- **2. Durum (Toplamın 11 olması):**

$$(5,6),(6,5) \implies 2 \text{ farklı durum.}$$

İki durum ayrık olduğundan toplama kuralı uygulanır:

$$6 + 2 = 8$$

farklı durumda toplam 7 veya 11 gelir. □

**Örnek 3.11.** 1 *ile* 100 *arasındaki (1 ve 100 dahil) tam sayılardan kaç tanesi* 3*'e veya* 5*'e tam bölünür?*

*Çözüm.* Durumların ayrık olup olmadığını kontrol edelim:

- 3'e bölünen sayılar kümesi $A$ olsun: $3, 6, 9, \ldots, 99$. $99 = 3 \times 33$ olduğundan $|A| = 33$'tür.
- 5'e bölünen sayılar kümesi $B$ olsun: $5, 10, 15, \ldots, 100$. $100 = 5 \times 20$ olduğundan $|B| = 20$'dir.

Hem 3'e hem 5'e bölünen sayılar (Bölüm 11 ve 13'te ele alınan bölünebilme ve EKOK kavramları uyarınca) 15'in katlarıdır ($A \cap B$):

$$15, 30, 45, 60, 75, 90 \implies |A \cap B| = 6$$

Kümeler ayrık olmadığından doğrudan $33 + 20$ toplanamaz; 15'in katları iki kez sayılmış olur. Birleşim formülü uygulanır:

$$|A \cup B| = |A| + |B| - |A \cap B| = 33 + 20 - 6 = 47$$

Buna göre bu aralıkta 3'e veya 5'e bölünen 47 sayı vardır.

Alternatif olarak, problem **ayrık alt kümelere ayrılarak** da çözülebilir:

- Yalnızca 3'e bölünüp 5'e bölünmeyenler: $33 - 6 = 27$
- Yalnızca 5'e bölünüp 3'e bölünmeyenler: $20 - 6 = 14$

- Hem 3'e hem 5'e bölünenler: 6

Bu üç grup ikili olarak ayrıktır. Toplama kuralı uygulanırsa:

$$27 + 14 + 6 = 47$$

aynı sonuca ulaşılır. □

## 3.3 Ağaç Diyagramları ve Listeleme

Çarpma kuralı son derece kullanışlıdır; ancak her adımda var olan seçenek sayısının önceki seçimlerden bağımsız olarak **sabit** kalmasını gerektirir.

Sonraki adımın seçenekleri önceki adımlardaki tercihe göre değişiyorsa, bazı seçimler süreci sonlandırıp bazıları yeni dallar açıyorsa çarpma formülü doğrudan uygulanamaz. Böyle durumlarda **ağaç diyagramları** (tree diagrams) ve sistematik listeleme yöntemlerine başvurulur.

**Tanım 3.12** (Ağaç Diyagramı)**.** *Bir sürecin tüm aşamalarını görselleştiren dallanmış yapıya* ***ağaç diyagramı*** *denir:*

- *Süreç en tepedeki tek bir noktadan (**kök**) başlar.*
- *Her karar veya olay, o noktadan çıkan **dallar** ile gösterilir.*
- *Dallar boyunca ilerleyerek sürecin tamamlandığı uç noktalara **yaprak** denir.*
- *Kökten herhangi bir yaprağa uzanan her kesintisiz yol, sürecin olası **bir sonucunu** temsil eder.*

Ağaç diyagramları; bilgisayar biliminde karar ağaçlarının (*decision trees*), oyun kuramı algoritmalarının ve sözdizim çözümleyicilerinin (*parsers*) temel modelini oluşturur.

### 3.3.1 Sistematik Listeleme ve Sözlük Sırası

Elemanları elle listelerken bazı durumları atlamamak ya da aynı durumu birden fazla kez saymamak için **sözlük sırası** (lexicographical order) kullanılır. Sözlükte kelimelerin alfabetik dizilmesinde olduğu gibi, sayma durumları da belirli bir öncelik kuralına göre düzenlenir.

### 3.3.2 Çözümlü Örnekler

**Örnek 3.13** (Klasik Turnuva Problemi)**.** *İki takım (A ve B) bir dizi maç yapacaktır. Kurallar şöyledir:*

- *Beraberlik yoktur; her maçı ya A ya da B kazanır.*

- ***Üst üste*** 2 ***maç kazanan*** *takım turnuva şampiyonu olur ve turnuva biter.*
- *Veya takımlardan biri* ***toplam*** 3 ***maç kazandığında*** *turnuva sona erer.*

*Bu turnuva kaç farklı maç dizilimi ile sonuçlanabilir?*

*Çözüm.* Turnuvanın kaç maç süreceği önceden sabit değildir; süreç 2 maçta da bitebilir, 5 maça da uzayabilir.

İlk maçı $A$'nın kazandığı senaryoları ($A$ ile başlayan dalları) adım adım izleyelim:

- **2. Maç:**
  - $A$ kazanırsa: Dizi $AA$ olur. $A$ üst üste 2 maç kazandığı için turnuva biter (**1. Senaryo:** $AA$ – 2 maç).
  - $B$ kazanırsa: Dizi $AB$ olur, turnuva sürer.
- **3. Maç ($AB$'den sonra):**
  - $B$ kazanırsa: Dizi $ABB$ olur. $B$ üst üste 2 maç kazandığından turnuva biter (**2. Senaryo:** $ABB$ – 3 maç).
  - $A$ kazanırsa: Dizi $ABA$ olur, turnuva sürer.
- **4. Maç ($ABA$'dan sonra):**
  - $A$ kazanırsa: Dizi $ABAA$ olur. $A$ üst üste 2 maç kazandığı için turnuva biter (**3. Senaryo:** $ABAA$ – 4 maç).
  - $B$ kazanırsa: Dizi $ABAB$ olur, turnuva sürer.
- **5. Maç ($ABAB$'den sonra):**
  - $A$ kazanırsa: Dizi $ABABA$ olur. $A$ toplam 3 galibiyete ulaştığı için turnuva biter (**4. Senaryo:** $ABABA$ – 5 maç).
  - $B$ kazanırsa: Dizi $ABABB$ olur. $B$ hem üst üste 2 maç kazanmış hem de toplam 3 galibiyete ulaşmış olur (**5. Senaryo:** $ABABB$ – 5 maç).

İlk maçı $A$'nın kazandığı durumda 5 farklı bitiş senaryosu elde edilir:

$$AA,\ \ ABB,\ \ ABAA,\ \ ABABA,\ \ ABABB$$

Takımların rolleri simetriktir (bu tür simetri argümanlarını Bölüm 17'de daha ayrıntılı inceleyeceğiz). Dolayısıyla ilk maçı $B$'nin kazandığı dal da simetrik biçimde 5 farklı senaryo üretir:

$$BB,\ \ BAA,\ \ BABB,\ \ BABAB,\ \ BABAA$$

Toplama kuralı gereğince turnuva toplam:

$$5 + 5 = 10$$

farklı şekilde sonuçlanabilir. □

**Örnek 3.14.** *Bir madeni para havaya atılıyor. Oyun şu iki koşuldan biri sağlandığında derhâl durmaktadır:*

- *Art arda iki kez Tura ($T$) gelmesi, ya da*
- *Toplamda üç kez Yazı ($Y$) gelmesi.*

*Bu oyun en fazla kaç atış sürebilir ve kaç farklı atış dizilimi ile sonlanabilir?*

*Çözüm.* Oyunun olası bitiş dizilimlerini, Yazı ($Y$) sayısına göre ayrık durumlara ayırarak listeleyelim:

- **0 Yazı içeren bitiş:** $TT$ (2 atış).
- **1 Yazı içeren bitişler:** $YTT, TYTT$ (3 ve 4 atış).
- **2 Yazı içeren bitişler:** $YYTT, YTYTT, TYYTT, TYTYTT$ (4, 5 ve 6 atış).
- **3 Yazı içeren bitişler (3 Yazı'ya ulaşıldığı an biter):** Art arda iki $T$ gelmeden 3 adet $Y$ biriktiren dizilimlerdir: $YYY$, $TYYY$, $YTYY$, $YYTY$, $TYTYY$, $TYYTY$, $YTYTY$, $TYTYTY$.

Ağaç diyagramının tüm dalları incelendiğinde, art arda iki Tura gelmeden en uzun süre devam edebilen salınım dizilimi $TYTYTY$ olup 6. atışta 3. Yazı kuralıyla biter; bu yüzden en uzun bitiş tam olarak 6 atıştır. Toplam bitiş dizilimi sayısı incelendiğinde, 2 Yazı'dan az içeren bitişler 7 tanedir ($TT$, $YTT$, $TYTT$, $YYTT$, $YTYTT$, $TYYTT$, $TYTYTT$), 3 Yazı içeren bitişler 8 tanedir; dolayısıyla toplam $7 + 8 = 15$ farklı dizilim vardır ve süreç sonlu sayıda yaprak üretir. □

**Örnek 3.15.** $S = \{1, 2, 3\}$ *kümesinin elemanlarıyla yazılabilecek rakamları farklı tüm iki basamaklı sayıları sözlük sırasıyla listeleyiniz ve sayısını çarpma kuralıyla doğrulayınız.*

*Çözüm.* Sözlük sırasıyla listeleme yapalım (önce onlar basamağı küçük olanlar):

- Onlar basamağı 1 olanlar: $12, 13$
- Onlar basamağı 2 olanlar: $21, 23$
- Onlar basamağı 3 olanlar: $31, 32$

Toplam 6 sayı listelenmiştir.

Çarpma kuralı ile doğrulama:

- Onlar basamağı için 3 seçenek vardır.
- Birler basamağı için geriye kalan 2 seçenek vardır.

Toplam $3 \times 2 = 6$ sayıdır. Doğrudan listeleme sonucu ile formül örtüşmektedir. □

## 3.4 İki İlkeyi Birlikte Kullanma

Birçok sayma probleminde çarpma ve toplama kuralları birlikte kullanılır. Karmaşık bir problem önce **birbirini dışlayan ana senaryolara (ayrık durumlara)** ayrılır; her senaryodaki seçenek sayısı **çarpma kuralı** ile hesaplanır ve elde edilen sonuçlar **toplama kuralı** ile birleştirilir:

$$\text{Toplam Sonuç} = \text{Senaryo}_1 + \text{Senaryo}_2 + \cdots + \text{Senaryo}_k$$

Bunun yanı sıra, doğrudan sayım yerine **tüm durumların sayısından istenmeyen durumların sayısını çıkarmak** işlem yükünü belirgin biçimde azaltabilir. Bu yaklaşıma **tümleyen yoluyla sayma** (komplementer sayma) denir.

### 3.4.1 Çözümlü Örnekler

**Örnek 3.16.** $\{0,1,2,3,4,5\}$ *kümesinin elemanları kullanılarak rakamları birbirinden farklı, dört basamaklı ve* 5 ***ile tam bölünebilen*** *kaç farklı doğal sayı yazılabilir?*

*Çözüm.* Burada sıfır (0) rakamının iki farklı kısıtlaması vardır: Dört basamaklı bir sayının en başına gelemez; ancak 5'e bölünebilme gereği son basamakta bulunabilir.

Sıfırın son basamakta kullanılıp kullanılmaması binler basamağındaki seçenek sayısını değiştirdiğinden, çarpma kuralı tek adımda uygulanamaz. Problem, **son basamağa göre iki ayrık duruma** ayrılarak çözülür:

**1. Durum: Son basamağın** $0$ **olması (** $\overline{a\,b\,c\,0}$ **):**

- Birler basamağı için tek seçenek vardır: $\{0\}$ (1 seçenek).
- Binler basamağı için: 0 birler basamağında kullanıldığından geriye kalan $\{1,2,3,4,5\}$ kümesinden 5 rakam da seçilebilir (5 seçenek).
- Yüzler basamağı için: Kalan 4 rakamdan biri (4 seçenek).
- Onlar basamağı için: Kalan 3 rakamdan biri (3 seçenek).

Çarpma kuralı gereğince 1. durumdaki sayı adedi:

$$5 \times 4 \times 3 \times 1 = 60$$

olur.

**2. Durum: Son basamağın** $5$ **olması (** $\overline{a\,b\,c\,5}$ **):**

- Birler basamağı için tek seçenek vardır: $\{5\}$ (1 seçenek).

- Binler basamağı için: 5 kullanılmıştır ve 0 da başa gelemez; dolayısıyla $\{1, 2, 3, 4\}$ arasından seçim yapılır (4 seçenek).
- Yüzler basamağı için: Bu basamakta 0 kullanılabilir. Toplam 6 rakamdan 2'si kullanıldığından geriye $6 - 2 = 4$ seçenek kalır (4 seçenek).
- Onlar basamağı için: Kalan 3 seçenekten biri (3 seçenek).

Çarpma kuralı gereğince 2. durumdaki sayı adedi:

$$4 \times 4 \times 3 \times 1 = 48$$

olur.

**Birleştirme:** Bu iki durumun son basamakları farklı olduğundan durumlar ayrıktır. Toplama kuralı gereği:

$$60 + 48 = 108$$

farklı sayı yazılabilir. □

**Örnek 3.17.** $\{0, 1, 2, 3, 4, 5\}$ *kümesinin elemanları ile rakamları tekrarsız, üç basamaklı ve* 300***'den büyük tek sayılar*** *yazılmak isteniyor. Kaç farklı sayı yazılabilir?*

*Çözüm.* Koşulları inceleyelim:

1. Üç basamaklı sayı: $\overline{abc}$.
2. 300'den büyük olması için $a \in \{3, 4, 5\}$ olmalıdır.
3. Tek sayı olması için $c \in \{1, 3, 5\}$ olmalıdır.

3 ve 5 rakamları hem yüzler hem birler basamağı için adaydır. Yüzler basamağında çift rakam olan 4 seçilirse birler basamağına 3 tek sayı seçeneği kalır. Ancak yüzler basamağında 3 veya 5 seçilirse birler basamağının seçeneği 2'ye düşer.

Bu çakışmayı yönetmek için yüzler basamağının tek veya çift olma durumuna göre iki ayrık durum incelenir:

**1. Senaryo: Yüzler basamağının çift olması ($a = 4$):**

- Yüzler basamağı ($a$): Yalnızca $\{4\}$ seçilebilir (1 seçenek).
- Birler basamağı ($c$): Tek sayılardan $\{1, 3, 5\}$ herhangi biri seçilebilir (3 seçenek).
- Onlar basamağı ($b$): Toplam 6 rakamdan 2'si kullanılmıştır; geriye kalan 4 rakamdan biri seçilir (4 seçenek).

1. senaryodaki sayı adedi:

$$1 \times 4 \times 3 = 12$$

**2. Senaryo: Yüzler basamağının tek olması ($a \in \{3, 5\}$):**

- Yüzler basamağı ($a$): $\{3, 5\}$ kümesinden biri seçilir (2 seçenek).
- Birler basamağı ($c$): $\{1, 3, 5\}$ kümesinden yüzler basamağında kullanılan rakam hariç kalan 2 seçenekten biri seçilir (2 seçenek).
- Onlar basamağı ($b$): Kalan 4 rakamdan biri seçilir (4 seçenek).

2. senaryodaki sayı adedi:

$$2 \times 4 \times 2 = 16$$

Bu iki senaryo yüzler basamağının değerine göre ayrık olduğundan, toplama kuralı uyarınca:

$$12 + 16 = 28$$

farklı sayı yazılabilir. □

**Örnek 3.18** (Tümleyen Yoluyla Sayma: Güvenli Şifre Problemi)**.** *Bir banka sistemi için* 4 *haneli şifreler üretilecektir. Her hanede* 0*'dan* 9*'a kadar olan* 10 *rakamdan biri kullanılabilir (rakam tekrarı serbesttir).*

*İçinde* ***en az bir adet*** 7 ***rakamı*** *barındıran kaç farklı şifre oluşturulabilir?*

*Çözüm.* Bu problemi doğrudan çözmek için tam 1, tam 2, tam 3 ve 4 tane 7 içeren durumları ayrı ayrı hesaplamak gerekir.

Bunun yerine **tümleyen yoluyla sayma** kullanılır:

$$\text{İstenen Durumlar} = \text{Tüm Durumlar} - \text{İstenmeyen Durumlar}$$

Burada:

- **Tüm Durumlar:** Hiçbir kısıtlama olmaksızın yazılabilecek tüm 4 haneli şifreler. Her hanede 10 rakam seçeneği olduğundan:

$$10 \times 10 \times 10 \times 10 = 10^4 = 10000$$

- **İstenmeyen Durumlar:** "En az bir tane 7 bulunması" koşulunun tersi, şifrede **hiç 7 bulunmamasıdır**. Hiç 7 kullanmamak, her hane için 7 hariç kalan 9 rakamdan birini seçmek demektir:

$$9 \times 9 \times 9 \times 9 = 9^4 = 6561$$

O hâlde en az bir 7 içeren şifrelerin sayısı:

$$10000 - 6561 = 3439$$

olarak tek bir çıkarma işlemiyle elde edilir. □

**Örnek 3.19.** $8 \times 8$ *boyutlarındaki standart bir satranç tahtasına, birbirini tehdit edemeyecek (yani aynı satırda veya aynı sütunda yer almayacak) biçimde biri beyaz biri siyah iki kale kaç farklı şekilde yerleştirilebilir?*

*Çözüm.* İki aşamalı bir seçim süreci kuralım:

1. **1. Aşama (Beyaz kalenin yerleştirilmesi):** Satranç tahtasında toplam $8 \times 8 = 64$ kare vardır. Beyaz kale bu 64 kareden herhangi birine yerleştirilebilir (64 seçenek).

2. **2. Aşama (Siyah kalenin yerleştirilmesi):** Beyaz kale bir kareye konulduğunda, siyah kalenin konulamayacağı kareler şunlardır:

   - Beyaz kalenin durduğu kare (1 kare).
   - Beyaz kalenin bulunduğu satırdaki diğer kareler (7 kare).
   - Beyaz kalenin bulunduğu sütundaki diğer kareler (7 kare).

   Bu satır ve sütun yalnızca beyaz kalenin bulunduğu karede kesiştiğinden, toplam yasaklı kare sayısı:

   $$1 + 7 + 7 = 15$$

   olur.

   Buna göre siyah kale için geriye kalan kare sayısı:

   $$64 - 15 = 49$$

   olur. (Alternatif olarak: Siyah kale beyazın bulunduğu satırın dışındaki 7 satırdan birinde ve beyazın bulunduğu sütunun dışındaki 7 sütundan birinde bulunmalıdır; yani $7 \times 7 = 49$ karedir).

Çarpma kuralı gereğince iki kalenin yerleştirilme sayısı:

$$64 \times 49 = 3136$$

farklı şekilde hesaplanır. □

### 3.4.2 C Kodu ile Doğrulama ve Döngü Sezgisi

Algoritma analizinde iç içe döngülerin adım sayısını hesaplarken çarpma ve toplama kurallarından yararlanırız.

Aşağıdaki C kod bloklarını inceleyelim:

```
// Ornek A: Carpmali yapi (bagimsiz ic ice donguler)
int sayacA = 0;
for (int i = 0; i < M; i++) {
    for (int j = 0; j < N; j++) {
        sayacA++;
    }
}
// sayacA degeri tam olarak M * N olur.

// Ornek B: Toplamali yapi (ardisik ayrik donguler)
int sayacB = 0;
for (int i = 0; i < M; i++) {
    sayacB++;
}
for (int j = 0; j < N; j++) {
    sayacB++;
}
// sayacB degeri tam olarak M + N olur.
```

Bir algoritmada işlemler ardışık bloklar hâlindeyse zaman maliyeti toplanır $(M + N)$. İşlemler iç içe geçmiş döngülerle yürütülüyorsa adım sayıları çarpılır $(M \times N)$. Saymanın temel ilkeleri, algoritma analizinde $O(N)$ ve $O(N^2)$ gibi karmaşıklık sınıflarını (Bölüm 21) belirlemenin de zeminini oluşturur.

# Bölüm 4

# Permütasyon

Bölüm 3'te nesneleri ve durumları saymanın iki temel dayanağını kurmuştuk: bağımsız adımların seçenek sayılarını çarpmak ve birbirini dışlayan ayrık durumları toplamak. Bu iki temel ilkeyi arkamıza alarak kombinatoriğin en temel yapı taşlarından birine geçiyoruz: **Sıralama**.

Gerek günlük yaşamda gerekse bilgisayar algoritmalarında nesnelerin diziliş sırası belirleyici bir fark yaratır:

- Bir kelimede harflerin diziliş sırası anlamı bütünüyle değiştirir: "KARA" ile "ARAK" tamamen aynı harflerden oluşur, ancak iki farklı kelimedir.
- Bir koşu yarışında ilk üç dereceye giren atletler belirlenirken 1. olmak ile 3. olmak aynı şey değildir; derece sırası esastır.
- Bir işletim sisteminin işlemciye gönderdiği görevlerin (task) çalışma sırası, sistemin kilitlenip kilitlenmeyeceğini veya kaynakların ne kadar verimli kullanılacağını belirler.

Nesnelerin belirli bir sıraya göre dizilmesine matematikte **permütasyon** (sıralı dizilim) denir. Bu bölümde faktöriyel kavramını çarpma kuralından türetecek; $n$ nesneden $r$ tanesinin sıralı seçimini ($P(n, r)$), özdeş nesnelerin yer aldığı tekrarlı permütasyonları ve dönme simetrisine sahip dairesel dizilimleri ele alacağız.

## 4.1 Faktöriyel

Küçük değerlerle hesaplayabileceğimiz somut bir soruyla başlayalım: Birbirinden farklı $n$ adet kitabı düz bir rafa yan yana kaç farklı sırada dizebiliriz?

Örüntüyü görmek için $n$'nin küçük değerlerini inceleyelim:

- $n = 1$ kitap ($A$): Yalnızca 1 dizilim vardır: $A$.
- $n = 2$ kitap ($A, B$): İki dizilim mümkündür: $AB, BA \rightarrow 2$ dizilim.

- $n = 3$ kitap $(A, B, C)$: Kitapları tek tek listeleyelim:

$$ABC,\ \ ACB,\ \ BAC,\ \ BCA,\ \ CAB,\ \ CBA \implies 6 \text{ dizilim.}$$

- $n = 4$ kitap $(A, B, C, D)$: 1. sıraya bu 4 kitaptan herhangi birini koyabiliriz. 1. sıraya hangi kitabı koyarsak koyalım, geriye kalan 3 kitap kalan 3 pozisyona yukarıda bulduğumuz gibi 6 farklı şekilde dizilir. Çarpma kuralı gereğince:

$$4 \times 6 = 24 \text{ dizilim.}$$

Şimdi genel duruma bakalım: Birbirinden farklı $n$ nesneyi yan yana $n$ adet boş pozisyona yerleştirmek, $n$ adımlı bir karar sürecidir:

$$\underline{\quad 1. \quad}\ \ \underline{\quad 2. \quad}\ \ \underline{\quad 3. \quad}\ \ \ldots\ \ \underline{\quad n. \quad}$$

1. En baştaki 1. pozisyon için elimizde $n$ farklı nesne seçeneği vardır.
2. 2. pozisyon için, ilk pozisyona konulan nesne hariç geriye kalan $n - 1$ nesneden biri seçilir.
3. 3. pozisyon için geriye kalan $n - 2$ nesneden biri seçilir.
4. Bu şekilde devam edildiğinde, sonuncu $n$. pozisyon için geriye yalnızca 1 nesne kalır.

Bölüm 3'te ele aldığımız çarpma kuralı uyarınca, bu adımların seçenek sayılarını çarparız:

$$n \times (n-1) \times (n-2) \times \cdots \times 2 \times 1$$

Matematikte bu ardışık çarpım o kadar sık karşımıza çıkar ki kendine ait özel bir gösterimi vardır.

**Tanım 4.1** (Faktöriyel)**.** *$n$ pozitif bir tam sayı olmak üzere, 1'den $n$'ye kadar olan tüm pozitif tam sayıların çarpımına $n$* ***faktöriyel*** *denir ve $n!$ simgesiyle gösterilir:*

$$n! = n \times (n-1) \times (n-2) \times \cdots \times 3 \times 2 \times 1$$

*Özel bir kabul olarak:*

$$0! = 1$$

*olarak tanımlanır.*

**Neden $0! = 1$'dir?** Bu eşitlik ilk bakışta sezgilerle çelişiyor gibi görünebilir; oysa arkasında iki tutarlı gerekçe yatar:

1. **Kombinatorik Sezgi:** $n!$, $n$ elemanlı bir kümenin elemanlarını sıralama sayısını temsil eder. Boş kümede $(n = 0)$ sıralanacak hiçbir eleman yoktur; hiçbir elemanı sıralamamanın tam 1 yolu vardır (boş dizilim).

2. **Cebirsel Tutarlılık:** Faktöriyel fonksiyonu şu temel yineleme bağıntısını sağlar:

$$n! = n \times (n-1)! \implies (n-1)! = \frac{n!}{n}$$

Bu eşitlikte $n = 1$ yazarsak:

$$0! = \frac{1!}{1} = \frac{1}{1} = 1$$

elde edilir. Şayet $0! = 0$ kabul edilseydi, kombinatorikteki temel formüllerin tutarlılığı bozulurdu.

**Teorem 4.2.** *Birbirinden farklı $n$ nesnenin yan yana doğrusal dizilimlerinin toplam sayısı $n!$'dir.*

*Kanıt.* İspat, yukarıda kurduğumuz $n$ adımlı karar sürecine çarpma kuralının doğrudan uygulanmasıyla tamamlanmıştır. □

### 4.1.1 Kombinatorik Patlama (Combinatorial Explosion)

Faktöriyel fonksiyonu son derece hızlı büyür. İlk birkaç değere göz atalım:

$$\begin{aligned}
1! &= 1 \\
3! &= 6 \\
5! &= 120 \\
7! &= 5\,040 \\
10! &= 3\,628\,800 \\
13! &= 6\,227\,020\,800 \quad \text{(32-bit işaretli tamsayı sınırını aşar)} \\
20! &\approx 2.43 \times 10^{18} \quad \text{(64-bit standart tamsayı sınırına dayanır)} \\
52! &\approx 8.0658 \times 10^{67}
\end{aligned}$$

52! değerine dikkat ediniz: 52 kartlık standart bir iskambil destesini iyice karıştırdığınızda elde ettiğiniz kart dizilimi, neredeyse kesinlikle dünya tarihi boyunca daha önce hiç kimsenin eline geçmemiş benzersiz bir dizilimdir. Çünkü 52!, evrendeki tahmin edilen toplam atom sayısına ($10^{80}$) yaklaşacak denli devasa bir büyüklüktür.

Bilgisayar biliminde olası tüm sıralamaları deneyen (brute force / kaba kuvvet) algoritmaların karmaşıklığı $O(n!)$ olarak ifade edilir. Bu patlama nedeniyle $O(n!)$ karmaşıklığındaki programlar günümüz bilgisayarlarında bile yalnızca $n \leq 11-12$ gibi çok küçük sayılar için makul sürede tamamlanabilir; $n = 20$ olduğunda ise işlem süreleri pratik sınırların çok ötesine geçer.

### 4.1.2 Çözümlü Örnekler

**Örnek 4.3.** *Aşağıdaki cebirsel ifadelerin değerini hesaplayınız ve sadeleştiriniz:*

1. $\frac{10!}{8!\times 2!}$
2. $n \geq 1$ *için* $\frac{(n+2)!}{n!}$

*Çözüm.* Faktöriyelin temel özelliği büyük olanın küçük cinsinden yazılabilmesidir: $(n+1)! = (n+1)\cdot n!$.

1. $10! = 10 \times 9 \times 8!$ olarak yazılabilir:
$$\frac{10!}{8! \times 2!} = \frac{10 \times 9 \times 8!}{8! \times 2} = \frac{10 \times 9}{2} = \frac{90}{2} = 45$$
2. Benzer biçimde $(n+2)! = (n+2)(n+1)\cdot n!$ yazılır:
$$\frac{(n+2)!}{n!} = \frac{(n+2)(n+1)\cdot n!}{n!} = (n+2)(n+1) = n^2 + 3n + 2$$

□

**Örnek 4.4** (Blok / Paketleme Yöntemi)**.** *Anne, baba ve* 3 *çocuktan oluşan* 5 *kişilik bir aile düz bir sırada yan yana fotoğraf çektirecektir.*

1. *Anne ve baba daima yan yana olmak şartıyla kaç farklı şekilde sıralanabilirler?*
2. *Anne ve baba kesinlikle yan yana* ***olmamak*** *şartıyla kaç farklı şekilde sıralanabilirler?*

*Çözüm.* Bu problem saymadaki iki temel yaklaşımı örnekler:

1. **Blok (Paketleme) Yöntemi:** Anne ($A$) ve babayı ($B$) birbirinden hiç ayrılmayacak tek bir varlık, yani bir "blok" gibi düşünelim: $[AB]$.

   Artık elimizde sıralanacak şu nesneler vardır:
   $$[AB],\ \text{Çocuk}_1,\ \text{Çocuk}_2,\ \text{Çocuk}_3$$
   Toplam $1 + 3 = 4$ nesne vardır. Bu 4 nesne kendi arasında 4! farklı şekilde sıralanabilir.

   Ayrıca blok içindeki anne ve baba da kendi arasında yer değiştirebilir: $(A, B)$ veya $(B, A)$ olmak üzere 2! seçenek vardır.

   Çarpma kuralı gereğince toplam sıralama sayısı:
   $$4! \times 2! = 24 \times 2 = 48$$
   farklı şekilde gerçekleşebilir.

2. **Tümleyen Yoluyla Sayma:** Bölüm 3'te gördüğümüz gibi, "yan yana olmama" durumunu doğrudan saymak yerine tüm durumlardan istenmeyen durumu çıkarırız:

   $$\begin{aligned}\text{İstenen} = &\ \text{Tüm Sıralamalar} \\ &- \text{Anne ve Babanın Yan Yana Olduğu Sıralamalar}\end{aligned}$$

   - Hiçbir kısıtlama olmaksızın 5 kişi $5! = 120$ farklı şekilde sıralanır.
   - Anne ve babanın yan yana olduğu durumların sayısını az önce 48 olarak hesapladık.

   O hâlde anne ve babanın yan yana olmadığı sıralamaların sayısı:

   $$120 - 48 = 72$$

   olarak tek bir çıkarma işlemiyle bulunur.

□

**Örnek 4.5.** *Bir kitaplık rafına* 4 *farklı matematik kitabı ve* 3 *farklı fizik kitabı dizilecektir. Aynı dersin kitapları daima bir arada bulunmak koşuluyla bu kitaplar kaç farklı sırada dizilebilir?*

*Çözüm.* Yine blok yöntemini kullanalım:

- Matematik kitaplarını bir paket yapalım: $[M_1M_2M_3M_4]$ (1 blok).
- Fizik kitaplarını bir paket yapalım: $[F_1F_2F_3]$ (1 blok).

Adım adım çarpma kuralını uygulayalım:

1. **1. Adım (Blokların dizilişi):** Elimizde $[M]$ ve $[F]$ olmak üzere 2 blok vardır. Bu iki blok rafta $2! = 2$ farklı sırada durabilir (önce matematikler sonra fizikler ya da tam tersi).
2. **2. Adım (Matematik kitaplarının kendi içi):** Blok içinde 4 farklı matematik kitabı $4! = 24$ farklı sırada dizilebilir.
3. **3. Adım (Fizik kitaplarının kendi içi):** Blok içinde 3 farklı fizik kitabı $3! = 6$ farklı sırada dizilebilir.

Çarpma kuralı uyarınca toplam dizilim sayısı:

$$2! \times 4! \times 3! = 2 \times 24 \times 6 = 288$$

farklı şekilde yapılabilir. □

## 4.2 Sıralı Seçim: P(n, r)

Faktöriyel bahsinde elimizdeki $n$ nesnenin **tamamını** sıraladık. Uygulamada ise çoğu zaman $n$ nesnenin tamamını değil, yalnızca belirli bir $r$ tanesini ($r \leq n$) seçip sıralamak gerekir.

Örneğin 10 atletin katıldığı bir 100 metre koşusunu düşünelim. Yarış sonunda podyuma çıkacak ilk üç derece (Altın, Gümüş, Bronz madalya) belirlenecektir. Kaç farklı podyum sıralaması oluşabilir?

- 1.lik (Altın madalya) için 10 aday vardır.
- 1. olan atlet 2. olamayacağından, 2.lik (Gümüş madalya) için geriye kalan 9 aday vardır.
- 3.lük (Bronz madalya) için geriye kalan 8 aday vardır.

Çarpma kuralı gereğince toplam podyum sıralaması:

$$10 \times 9 \times 8 = 720$$

farklı şekilde gerçekleşebilir. Burada 10 atletin tamamını sıralamadık; yalnızca 3 tanesini sırayla seçtik.

**Tanım 4.6** (Permütasyon: $P(n,r)$)**.** *$n$ ve $r$ iki doğal sayı ve $0 \leq r \leq n$ olsun. $n$ elemanlı bir kümeden birbirinden farklı $r$ tanesinin belirli bir sıraya göre seçilip dizilmesine $n$'nin $r$'li **permütasyonu** denir ve bu sayı $P(n,r)$ ile gösterilir.*

**Teorem 4.7.** *Her $0 \leq r \leq n$ için:*

$$P(n,r) = n \times (n-1) \times (n-2) \times \cdots \times (n-r+1)$$

*Bu ifade faktöriyel cinsinden şu şekilde yazılabilir:*

$$P(n,r) = \frac{n!}{(n-r)!}$$

*Kanıt.* Sıralı $r$ adet pozisyon belirleyelim:

$$\underline{\quad 1. \quad} \quad \underline{\quad 2. \quad} \quad \underline{\quad 3. \quad} \quad \ldots \quad \underline{\quad r. \quad}$$

- 1. pozisyon için $n$ seçenek vardır.
- 2. pozisyon için $n-1$ seçenek vardır.
- 3. pozisyon için $n-2$ seçenek vardır.
- …
- $k$. pozisyon için $n-(k-1) = n-k+1$ seçenek kalır.

- Dolayısıyla sonuncu $r$. pozisyon için $n-(r-1)=n-r+1$ seçenek kalır.

Çarpma kuralı uyarınca:

$$P(n,r)=n(n-1)(n-2)\dots(n-r+1)$$

elde edilir. Bu çarpımın pay ve paydasını $(n-r)!$ ile çarparsak:

$$P(n,r)=\frac{[n(n-1)\dots(n-r+1)]\times[(n-r)\dots 2\cdot 1]}{(n-r)!}=\frac{n!}{(n-r)!}$$

formülüne ulaşırız. □

Uç durumları kontrol edelim:

- $r=n$ ise: $P(n,n)=\frac{n!}{(n-n)!}=\frac{n!}{0!}=\frac{n!}{1}=n!$ (Tüm nesneleri sıralama sonucuyla örtüşür).
- $r=1$ ise: $P(n,1)=\frac{n!}{(n-1)!}=n$.
- $r=0$ ise: $P(n,0)=\frac{n!}{n!}=1$ (0 nesne seçip sıralamanın 1 yolu vardır: boş dizilim).

### 4.2.1 Çözümlü Örnekler

**Örnek 4.8.** *Bir sınıfta 8 öğrenci vardır. Bu öğrenciler arasından bir başkan, bir başkan yardımcısı ve bir nöbetçi öğrenci kaç farklı şekilde seçilebilir?*

*Çözüm.* Burada roller birbirinden farklıdır; yani seçim sırası önemlidir (başkan seçilmek ile nöbetçi seçilmek aynı görev değildir). Dolayısıyla bu bir sıralı seçim problemidir:

$$P(8,3)=\frac{8!}{(8-3)!}=\frac{8!}{5!}=8\times 7\times 6=336$$

farklı şekilde seçim yapılabilir. □

**Örnek 4.9.** $S=\{1,2,3,4,5,6,7\}$ *kümesinin elemanları kullanılarak rakamları tekrarsız dört basamaklı:*

1. *Kaç farklı sayı yazılabilir?*
2. 5 *ile başlayan kaç farklı sayı yazılabilir?*
3. 5 *rakamını kesinlikle* ***içermeyen*** *kaç farklı sayı yazılabilir?*

*Çözüm.* 1. 7 farklı rakamdan 4 tanesi seçilip sıralanacaktır:

$$P(7,4)=7\times 6\times 5\times 4=840$$

2. Sayı 5 ile başlayacaktır ($\overline{5\ b\ c\ d}$): Binler basamağı için 5 sabittir (1 seçenek). Geriye kalan 3 basamağa, kalan 6 rakamdan ($\{1, 2, 3, 4, 6, 7\}$) seçim yapılacaktır:

$$1 \times P(6, 3) = 1 \times (6 \times 5 \times 4) = 120$$

3. 5 rakamı kullanılmayacaksa, seçim kümemizden 5'i çıkarırız. Geriye 6 rakam kalır: $\{1, 2, 3, 4, 6, 7\}$. Bu 6 rakamla yazılabilecek 4 basamaklı sayılar:

$$P(6, 4) = 6 \times 5 \times 4 \times 3 = 360$$

farklı şekilde yazılabilir.

□

**Örnek 4.10.** 6 *kişi arasından* 4 *kişilik sıralı bir bayrak koşusu takımı kurulacaktır. Adaylardan biri olan Can'ın bu takımda* ***mutlaka yer alması*** *şartıyla kaç farklı takım sıralaması oluşturulabilir?*

*Çözüm.* Bu problemi iki adımlı bir süreçle ele alalım:

1. **1. Adım (Can'ın pozisyonunu belirleme):** Can bayrak koşusunda 1., 2., 3. veya 4. koşucu olabilir. Can için 4 farklı pozisyon seçeneği vardır.

2. **2. Adım (Kalan pozisyonları doldurma):** Can yerleştirildikten sonra geriye 3 boş pozisyon kalır. Can dışındaki geriye kalan 5 aday arasından bu 3 pozisyona koşucular seçilip dizilecektir:

$$P(5, 3) = 5 \times 4 \times 3 = 60$$

Çarpma kuralı gereğince:

$$4 \times P(5, 3) = 4 \times 60 = 240$$

farklı takım kurulabilir.

*Alternatif Yol (Tümleyenle Çözüm):*

$$\text{Can'ın Olduğu} = \text{Tüm Takımlar} - \text{Can'ın Olmadığı Takımlar}$$

$$= P(6, 4) - P(5, 4) = (6 \times 5 \times 4 \times 3) - (5 \times 4 \times 3 \times 2) = 360 - 120 = 240$$

Her iki yol da aynı sonuca ulaştırır. □

## 4.3 Tekrarlı Permütasyon

Şu ana kadar incelediğimiz tüm durumlarda sıralanan nesneler birbirinden tamamen farklıydı. Peki sıralamak istediğimiz bazı nesneler birbirinin tıpatıp aynısı (özdeş) olursa durum nasıl değişir?

Somut bir soruyla başlayalım: "KARA" kelimesinin harfleriyle 4 harfli kaç farklı kelime yazılabilir?

Eğer bu iki $A$ harfi birbirinden farklı olsaydı (örneğin biri kırmızı $A_1$, diğeri mavi $A_2$ olsaydı), 4 farklı harf $4! = 24$ farklı şekilde dizilirdi. Bu 24 dizilimden ikisini yazalım:

$$K\ A_1\ R\ A_2 \quad \text{ve} \quad K\ A_2\ R\ A_1$$

Harflerin üzerindeki indisleri kaldırdığımızda her iki dizilim de "KARA" kelimesine dönüşür; çünkü iki $A$ harfi kendi arasında yer değiştirdiğinde yeni bir kelime oluşmaz.

Her bir kelime, $A$'ların kendi arasındaki $2! = 2$ farklı yer değişiminden ötürü tam 2 kez sayılmıştır. Dolayısıyla gerçek farklı kelime sayısını bulmak için toplam permütasyon sayısını 2!'e bölmemiz gerekir:

$$\frac{4!}{2!} = \frac{24}{2} = 12$$

İşte bu mantık **tekrarlı permütasyon**un temelini oluşturur.

**Teorem 4.11** (Tekrarlı Permütasyon Formülü)**.** *Toplam $n$ adet nesnenin; $n_1$ tanesi 1. türden özdeş, $n_2$ tanesi 2. türden özdeş, …, $n_k$ tanesi $k$. türden özdeş olsun ($n_1 + n_2 + \cdots + n_k = n$).*

*Bu $n$ nesnenin yan yana farklı doğrusal dizilimlerinin toplam sayısı:*

$$\frac{n!}{n_1! \times n_2! \times \cdots \times n_k!}$$

*formülüyle hesaplanır.*

*Kanıt.* Tüm nesneleri önce birbirinden farklıymış gibi düşünelim (her birine geçici bir numara verelim). Bu durumda $n!$ farklı dizilim elde edilir.

Şimdi bu numaraları kaldıralım. 1. türden $n_1$ adet özdeş nesne kendi arasında $n_1!$ farklı şekilde yer değiştirebilir; fakat bu yer değişimleri dizilimi değiştirmez. Benzer biçimde 2. türden nesneler $n_2!$ kez, …, $k$. türden nesneler $n_k!$ kez aynı dizilimi üretir.

Çarpma kuralı gereğince her bir özgün dizilim, bu yapay numaralandırma yüzünden tam olarak

$$n_1! \times n_2! \times \cdots \times n_k!$$

defa fazladan sayılmıştır. Gerçek dizilim sayısını bulmak için toplam sayıyı bu katsayıya böleriz. □

### 4.3.1 Izgara Üzerinde En Kısa Yollar (Kafes Yolları)

Tekrarlı permütasyonun sıkça karşımıza çıkan uygulamalarından biri koordinat düzlemindeki kafes (ızgara) yollarıdır.

Düzlemde $(0, 0)$ noktasında bulunan bir robot, her adımda yalnızca 1 birim **Sağa** ($S$) ya da 1 birim **Yukarı** ($Y$) hareket edebilmektedir. Bu robot $(m, n)$ noktasına en kısa yoldan kaç farklı rota izleyerek ulaşabilir?

*Sezgi:* Robotun $(0, 0)$'dan $(m, n)$'ye varabilmesi için toplamda tam $m$ defa Sağa ($S$) ve tam $n$ defa Yukarı ($Y$) adım atması şarttır.

Her rota, $m$ adet $S$ ve $n$ adet $Y$ harfinden oluşan $m + n$ uzunluğunda bir dizilimdir. Örneğin $(2, 3)$ noktasına gitmek için:

$$S\ S\ Y\ Y\ Y \quad \text{veya} \quad S\ Y\ S\ Y\ Y \quad \text{veya} \quad Y\ Y\ S\ S\ Y$$

rotalarından biri izlenir. Tüm $S$'ler birbiriyle özdeş, tüm $Y$'ler birbiriyle özdeştir. Dolayısıyla rota sayısı doğrudan bir tekrarlı permütasyondur:

$$\frac{(m+n)!}{m! \cdot n!}$$

(Bu önemli bağıntıyı Bölüm 5'te kombinasyon ve Bölüm 9'da kafes yolları ile yineleme bağıntıları bağlamında yeniden ele alacağız).

### 4.3.2 Çözümlü Örnekler

**Örnek 4.12.** *"MATEMATİK" kelimesinin harfleri yer değiştirilerek* 9 *harfli anlamlı ya da anlamsız kaç farklı kelime yazılabilir?*

*Çözüm.* Önce kelimedeki harflerin frekanslarını (tekrar sayılarını) çıkaralım:

- M harfi: 2 tane
- A harfi: 2 tane
- T harfi: 2 tane
- E harfi: 1 tane
- İ harfi: 1 tane
- K harfi: 1 tane

Toplam harf sayısı: $2+2+2+1+1+1 = 9$'dur. Tekrarlı permütasyon formülünü uygulayalım:

$$\frac{9!}{2! \times 2! \times 2! \times 1! \times 1! \times 1!} = \frac{362\,880}{2 \times 2 \times 2} = \frac{362\,880}{8} = 45\,360$$

farklı kelime yazılabilir. □

**Örnek 4.13.** $(0,0)$ *noktasından* $(4,3)$ *noktasına yalnızca sağa* $(S)$ *ve yukarı* $(Y)$ *adımlarla gitmek isteyen bir hareketli, yol üzerindeki* $(2,1)$ *noktasından* ***mutlaka geçmek*** *şartıyla kaç farklı en kısa rota izleyebilir?*

*Çözüm.* Hareketi iki ardışık etaba ayıralım (çarpma kuralı):

1. **1. Etap:** $(0,0) \to (2,1)$ **rotası:** Toplam 2 Sağa ve 1 Yukarı adım atılmalıdır (toplam 3 adım):
$$\frac{(2+1)!}{2! \times 1!} = \frac{3!}{2! \times 1!} = 3 \text{ rota.}$$
(Bu rotalar: $SSY, SYS, YSS$'dir).

2. **2. Etap:** $(2,1) \to (4,3)$ **rotası:** $x$ koordinatı 2'den 4'e (2 Sağa), $y$ koordinatı 1'den 3'e (2 Yukarı) gidecektir (toplam 4 adım):
$$\frac{(2+2)!}{2! \times 2!} = \frac{4!}{2! \times 2!} = \frac{24}{4} = 6 \text{ rota.}$$

Çarpma kuralı uyarınca $(2,1)$ üzerinden geçen toplam rota sayısı:

$$3 \times 6 = 18$$

farklı şekilde seçilebilir. □

**Örnek 4.14.** $2,2,0,3,3,5$ *rakamları kullanılarak altı basamaklı kaç farklı doğal sayı yazılabilir?*

*Çözüm.* Burada sıfır (0) rakamının en başa gelememesi kısıtı vardır (en başa sıfır gelirse sayı 5 basamaklı olur). İki yöntemle çözebiliriz:

**1. Yöntem (Tümleyenle Çözüm):**

- Hiçbir kısıtlama olmaksızın (0 başta da olabilseydi) 6 rakamın tekrarlı permütasyonu: Rakamlar: iki adet 2, iki adet 3, bir adet 0, bir adet 5.
$$\text{Tüm Durumlar} = \frac{6!}{2! \times 2! \times 1! \times 1!} = \frac{720}{4} = 180$$

- Sıfırın en başta olduğu istenmeyen durumlar ($\overline{0\ldots}$): İlk basamağa 0 sabitlenir; kalan 5 pozisyona $\{2,2,3,3,5\}$ rakamları dizilir:
$$\text{İstenmeyen Durumlar} = \frac{5!}{2! \times 2! \times 1!} = \frac{120}{4} = 30$$

Geçerli altı basamaklı sayıların sayısı:

$$180 - 30 = 150$$

olur.

**2. Yöntem (Oran / Simetri Sezgisi):** Toplam 6 rakamdan 5 tanesi sıfır dışındadır $(2, 2, 3, 3, 5)$. Rakamların simetrisinden ötürü, tüm dizilimlerin tam 5/6'sında ilk basamağa sıfırdan farklı bir rakam gelir:

$$180 \times \frac{5}{6} = 150$$

Her iki yöntem de aynı sonucu verir. □

## 4.4 Dairesel Permütasyon

Şu ana kadar tüm sıralamaları düz bir çizgi üzerinde yaptık. Doğrusal bir sırada belirgin bir başlangıç (en sol) ve bitiş (en sağ) noktası vardır.

Peki nesneleri **yuvarlak bir masa** etrafına dizersek ne değişir?

Küçük bir örnek üzerinden inceleyelim: 3 arkadaş $(A, B, C)$ yuvarlak bir masaya oturacaktır. Düz bir sırada bu 3 kişi $3! = 6$ farklı şekilde otururdu:

$$ABC,\ \ BCA,\ \ CAB,\ \ ACB,\ \ BAC,\ \ CBA$$

Şimdi yuvarlak masaya bakalım. $ABC$ diziliminde: $A$'nın sağında $B$, $B$'nin sağında $C$, $C$'nin sağında $A$ oturur. Herkesi aynı anda bir koltuk sağa kaydırırsak oturma düzeni $CAB$ olur; bir koltuk daha kaydırırsak $BCA$ olur.

Fakat dikkat edilirse kimsenin sağındaki ya da solundaki komşusu değişmemiştir. Masayı kendi ekseni etrafında döndürmek yeni bir oturma düzeni yaratmaz. Dairesel düzende mutlak koltuk numaraları değil, **kişilerin birbirine göre bağıl konumları** önemlidir.

Bu 3 doğrusal dizilim $(ABC, BCA, CAB)$ dairesel masa etrafında **aynı dizilime** karşılık gelir. Benzer biçimde diğer 3 dizilim $(ACB, BAC, CBA)$ de ikinci bir dairesel dizilim oluşturur. Dolayısıyla 3 kişinin yuvarlak masa etrafındaki farklı oturma düzeni sayısı:

$$\frac{3!}{3} = 2$$

olur.

**Teorem 4.15** (Dairesel Permütasyon Formülü)**.** *Birbirinden farklı $n$ adet nesnenin dairesel bir hat etrafındaki farklı dizilimlerinin toplam sayısı:*

$$(n-1)!$$

*formülüyle verilir.*

*Kanıt.* Bu teoremi iki farklı bakış açısıyla ispatlayabiliriz:

**1. İspat (Fazladan Sayımı Bölme):** $n$ farklı nesneyi önce düz bir sırada dizelim ($n!$ seçenek). Yuvarlak masa etrafında bu dizilimi her bir pozisyon döndürdüğümüzde (1 adım, 2 adım, ..., $n-1$ adım döndürme) toplam $n$ adet doğrusal dizilim birbirinin tıpatıp aynısı olan tek bir dairesel düzen üretir. Her düzen $n$ kez sayıldığı için:

$$\frac{n!}{n} = \frac{n \times (n-1)!}{n} = (n-1)!$$

bulunur.

**2. İspat (Referans Sabitleme Yöntemi):** Masaya ilk gelen kişiyi düşünelim. Masa henüz boştur ve dairesel simetriye sahiptir; dolayısıyla bu ilk kişinin hangi sandalyeye oturduğu fark yaratmaz (1 seçenek).

İlk kişi bir sandalyeye oturduğu anda dairesel simetri kırılır. Artık masada "o kişinin sağı", "o kişinin karşısı", "o kişinin iki koltuk solu" gibi sabit referans pozisyonları oluşmuştur. Geriye kalan $n-1$ kişi, bu referans noktasına göre sanki düz bir sıradaymış gibi doğrusal olarak yerleşir. Bu da $(n-1)!$ farklı şekilde yapılabilir. □

### 4.4.1 Dönen ve Ters Çevrilebilen Dizilimler (Bileklik ve Anahtarlık)

Yuvarlak bir masa iki boyutludur; masayı ters yüz edemezsiniz. Ancak bir **anahtarlık halkası** ya da boncuklardan oluşan bir **bileklik** üç boyutlu uzayda serbestçe ters çevrilebilir.

Bir bilekliği masanın üzerine koyup yukarıdan baktığınızda saat yönündeki sıralama, bilekliği arkasına çevirdiğinizde saat yönünün tersi hâline gelir. Yansıma (ters yüz etme) simetrisi nedeniyle her dairesel düzen bir de tersten kendisiyle eşleşir.

**Teorem 4.16** (Ters Çevrilebilen Dairesel Dizilimler)**.** *Birbirinden farklı $n$ nesnenin (örneğin anahtarların ya da boncukların), arkası çevrilebilen dairesel bir halka üzerindeki farklı dizilimlerinin sayısı:*

$$\frac{(n-1)!}{2}$$

*formülüyle hesaplanır ($n \geq 3$).*

### 4.4.2 Çözümlü Örnekler

**Örnek 4.17.** 6 *diplomat yuvarlak bir masa etrafında toplantı yapacaktır.*

1. *Hiçbir kısıtlama olmaksızın kaç farklı şekilde oturabilirler?*
2. *Belirli iki diplomatın (Ali ve Burak)* ***daima yan yana*** *oturması şartıyla kaç farklı şekilde oturabilirler?*

*Çözüm.* 1. 6 kişi dairesel masa etrafında:

$$(6-1)! = 5! = 120$$

farklı şekilde oturabilir.

2. Blok yöntemini uygulayalım: Ali ve Burak'ı tek bir blok yapalım: $[AB]$. Masada yer alacak varlıklar: $[AB]$ bloku ve geriye kalan 4 diplomat olmak üzere toplam $1 + 4 = 5$ nesnedir.

   - Bu 5 nesne yuvarlak masa etrafına $(5-1)! = 4! = 24$ farklı şekilde dizilir.
   - Blok içindeki Ali ve Burak kendi arasında $2! = 2$ farklı sırada oturabilir ($AB$ veya $BA$).

   Çarpma kuralı gereğince toplam oturma düzeni sayısı:

$$4! \times 2! = 24 \times 2 = 48$$

olur.

□

**Örnek 4.18.** 4 *kadın ve* 4 *erkek, yuvarlak bir masa etrafında* ***hiçbir iki kadın yan yana gelmeyecek*** *(yani bir kadın, bir erkek şeklinde ardışık) biçimde kaç farklı şekilde oturabilir?*

*Çözüm.* Bu tür ardışık dizilim problemlerinde iki aşamalı bir yerleştirme stratejisi izlenir:

1. **1. Aşama (Erkeklerin yerleşimi):** Önce 4 erkeği yuvarlak masaya oturtalım. Dairesel permütasyon kuralı gereğince 4 erkek masaya:

$$(4-1)! = 3! = 6$$

farklı şekilde oturur.

2. **2. Aşama (Kadınların yerleşimi):** Erkekler oturduktan sonra, aralarında tam 4 adet boş sandalye oluşur:

$$E_1 \ \_\_ \ E_2 \ \_\_ \ E_3 \ \_\_ \ E_4 \ \_\_$$

**Buradaki kritik nokta:** Erkekler masaya oturduğu anda dairesel simetri kırılmıştır. Kadınlar için bu 4 sandalye artık özdeş değildir; örneğin biri "$E_1$ ile $E_2$'nin arası", diğeri "$E_2$ ile $E_3$'ün arası"dır. Dolayısıyla kadınlar bu 4 sandalyeye dairesel değil, **doğrusal** olarak dizilirler:

$$4! = 24 \text{ seçenek.}$$

Çarpma kuralı uyarınca toplam oturma düzeni sayısı:

$$(4-1)! \times 4! = 6 \times 24 = 144$$

farklı şekilde gerçekleşebilir. □

**Örnek 4.19.** 7 *farklı anahtar, maskotsuz (halkası simetrik) bir anahtarlığa kaç farklı şekilde takılabilir?*

*Çözüm.* Anahtarlık ters çevrilebildiğinden yansıma simetrisi devreye girer:

$$\frac{(7-1)!}{2} = \frac{6!}{2} = \frac{720}{2} = 360$$

farklı şekilde takılabilir. □

# Bölüm 5

# Kombinasyon

Bölüm 4'te ele aldığımız permütasyon problemlerinde diziliş sırası belirleyiciydi: Bir podyumda birinci olmak ile üçüncü olmak arasındaki fark, bir kasanın şifresindeki rakamların dizilişi ya da bir yarışmadaki görev dağılımı doğrudan elemanların sırasına bağlıydı.

Matematikte ve bilgisayar biliminde ise çoğu zaman nesnelerin hangi sırada dizildiğiyle değil, **yalnızca hangi nesnelerin seçildiğiyle** ilgileniriz:

- 10 kişilik bir sınıftan okul meclisine 3 temsilci seçerken Ahmet, Ayşe ve Can'ın hangi sırayla belirlendiği temsilciler kurulunun yapısını değiştirmez. Kurul $\{\text{Ahmet}, \text{Ayşe}, \text{Can}\}$ kümesidir.

- Masadan çekilen 5 iskambil kartının çekiliş sırası elin değerini etkilemez; elde toplanan kartlar kümesi belirleyicidir.

- Bir bilgisayar ağında $n$ sunucu arasından birbirine doğrudan bağlanacak iki sunucu belirlemek, sırasız bir ikili seçmektir.

Nesnelerin seçilme sırasının hesaba katılmadığı bu tür seçimlere **kombinasyon** (sırasız seçim) denir. Burada kombinasyon kavramını permütasyondan türetecek, $\binom{n}{r}$ binom katsayısını, simetri ve Pascal bağıntısını, ardından özdeş nesnelerin dağıtımında başvurulan "Yıldızlar ve Bölücüler" yöntemini adım adım inceleyeceğiz.

## 5.1 Binom Katsayısı

Küçük bir örnekle başlayalım: Dört kişilik bir arkadaş grubundan $(A, B, C, D)$ iki kişilik bir proje ekibi kurmak istiyoruz. Kaç farklı ekip oluşturabiliriz?

Önce Bölüm 4'te gördüğümüz sıralı seçim (permütasyon) yaklaşımını uygulayalım:

$$P(4, 2) = 4 \times 3 = 12$$

Bu 12 sıralı ikiliyi açıkça yazalım:

$$(A,B),\ (A,C),\ (A,D),\ (B,A),\ (B,C),\ (B,D),$$
$$(C,A),\ (C,B),\ (C,D),\ (D,A),\ (D,B),\ (D,C)$$

Bir proje ekibi için $(A,B)$ ile $(B,A)$ arasında fark yoktur; her iki sıralı ikili de aynı iki kişiden oluşan $\{A,B\}$ ekibini belirtir. Benzer biçimde $(A,C)$ ile $(C,A)$ aynı ekiptir.

Her ekip, seçilen 2 kişinin kendi arasındaki $2! = 2$ farklı diziliminden ötürü listede tam 2 **defa** yer almıştır. Farklı ekip sayısını bulmak için sıralı seçimlerin sayısını 2!'e bölmeliyiz:

$$\frac{P(4,2)}{2!} = \frac{12}{2} = 6$$

Elde edilen 6 sırasız ekip şunlardır:

$$\{A,B\},\ \{A,C\},\ \{A,D\},\ \{B,C\},\ \{B,D\},\ \{C,D\}$$

Eğer bu 4 kişi arasından 3 kişilik bir ekip seçseydik, seçilen herhangi 3 kişi kendi arasında $3! = 6$ farklı sırada dizilebilirdi. Dolayısıyla her ekip $P(4,3) = 24$ sıralı üçlü listesinde tam 6 defa tekrarlanacaktı. Gerçek ekip sayısı $\frac{P(4,3)}{3!} = \frac{24}{6} = 4$ olurdu.

Bu gözlem bizi kombinasyonun formülüne ulaştırır:

**Tanım 5.1** (Kombinasyon ve Binom Katsayısı)**.** *$n$ ve $r$ iki doğal sayı ve $0 \leq r \leq n$ olsun. $n$ elemanlı bir kümenin $r$ elemanlı her bir alt kümesine $n$'nin $r$'li **kombinasyonu** denir. Bu alt kümelerin toplam sayısı $\binom{n}{r}$ (veya $C(n,r)$) simgesiyle gösterilir ve "$n$'nin $r$'lisi" diye okunur.*

**Teorem 5.2.** *Her $0 \leq r \leq n$ için:*

$$\binom{n}{r} = \frac{P(n,r)}{r!} = \frac{n!}{r!(n-r)!}$$

*Kanıt.* $n$ elemanlı bir kümeden $r$ elemanlı sıralı bir dizi $(P(n,r))$ oluşturma sürecini iki aşamalı bir işlem olarak kurgulayalım:

1. **1. Aşama (Kümeyi seçmek):** Sırasına bakılmaksızın hangi $r$ elemanın seçileceğini belirleriz. Bu seçim tanım gereği $\binom{n}{r}$ farklı şekilde yapılabilir.
2. **2. Aşama (Seçilenleri dizmek):** Seçilen $r$ elemanı bir sıraya dizeriz. $r$ farklı nesne kendi arasında $r!$ farklı şekilde sıralanır.

Bölüm 3'teki çarpma kuralı uyarınca bu iki aşamanın çarpımı toplam sıralı seçimlerin sayısını $(P(n,r))$ verir:

$$\binom{n}{r} \times r! = P(n,r)$$

Eşitliğin her iki tarafını $r!$'e bölersek:

$$\binom{n}{r} = \frac{P(n,r)}{r!}$$

elde edilir. Bölüm 4'teki $P(n,r) = \frac{n!}{(n-r)!}$ ifadesi yerine konduğunda:

$$\binom{n}{r} = \frac{n!}{r!(n-r)!}$$

bulunur. □

Sınır durumları inceleyelim:

- $\binom{n}{0} = \frac{n!}{0!(n-0)!} = \frac{n!}{1 \cdot n!} = 1$ ($n$ elemandan 0 eleman seçmenin 1 yolu vardır: boş küme $\emptyset$).
- $\binom{n}{1} = \frac{n!}{1!(n-1)!} = \frac{n \cdot (n-1)!}{(n-1)!} = n$ ($n$ nesneden tek bir nesne seçmenin $n$ yolu vardır).
- $\binom{n}{n} = \frac{n!}{n!(n-n)!} = \frac{n!}{n! \cdot 0!} = 1$ ($n$ nesnenin tamamını seçmenin tek bir yolu vardır: kümenin kendisi).

### 5.1.1 Kümeler Kuramı ile Bağlantı

Bölüm 1'de $n$ elemanlı bir kümenin toplam $2^n$ adet alt kümesi olduğunu çarpma kuralıyla doğrulamıştık. Kombinasyon bu alt kümeleri eleman sayılarına göre ayrık sınıflara böler:

- 0 elemanlı alt küme sayısı: $\binom{n}{0}$
- 1 elemanlı alt küme sayısı: $\binom{n}{1}$
- 2 elemanlı alt küme sayısı: $\binom{n}{2}$
- …
- $n$ elemanlı alt küme sayısı: $\binom{n}{n}$

Bu alt küme sınıfları birbirini dışlayan ayrık durumlar olduğundan, Bölüm 3'teki toplama kuralı gereğince tüm alt kümelerin toplamı:

$$\sum_{r=0}^{n} \binom{n}{r} = \binom{n}{0} + \binom{n}{1} + \binom{n}{2} + \cdots + \binom{n}{n} = 2^n$$

eşitliğini sağlar. Bu temel özdeşliği Bölüm 6'da binom teoremi çerçevesinde yeniden ele alacağız.

### 5.1.2 Çözümlü Örnekler

**Örnek 5.3.** *Aşağıdaki kombinasyonların değerlerini hesaplayınız: a)* $\binom{7}{3}$ *b)* $\binom{10}{2}$ *c)* $\binom{8}{4}$

*Çözüm.* Kombinasyon hesaplarken büyük faktöriyelleri açmak yerine, payı $r$ adım geriye açıp paydaya $r!$ yazmak daha pratiktir:

- a) $\binom{7}{3} = \frac{7\times6\times5}{3\times2\times1} = \frac{210}{6} = 35.$
- b) $\binom{10}{2} = \frac{10\times9}{2\times1} = \frac{90}{2} = 45.$
- c) $\binom{8}{4} = \frac{8\times7\times6\times5}{4\times3\times2\times1} = \frac{8\times7\times6\times5}{24} = 7\times2\times5 = 70.$

□

**Örnek 5.4.** 12 *üyeli bir öğrenci topluluğundan* 4 *kişilik bir disiplin kurulu kaç farklı şekilde seçilebilir?*

*Çözüm.* Kurul üyeleri arasında unvan ya da görev farkı bulunmadığından seçim sırası önemsizdir:

$$\binom{12}{4} = \frac{12\times11\times10\times9}{4\times3\times2\times1}$$

Paydadaki $4\times3 = 12$ paydaki 12 ile sadeleşir. Kalan 2 ise 10'u sadeleştirip 5 yapar:

$$= 11\times5\times9 = 495$$

farklı şekilde kurul seçilebilir. □

**Örnek 5.5.** *Bir çember üzerinde birbirinden farklı* 8 *nokta işaretlenmiştir.*

1. *Köşeleri bu noktalardan seçilen kaç farklı doğru parçası (kiriş) çizilebilir?*
2. *Köşeleri bu noktalardan seçilen kaç farklı üçgen oluşturulabilir?*

*Çözüm.* Çember üzerindeki noktaların hiçbir üçü doğrusal değildir.

1. Bir doğru parçası belirlemek için bu 8 noktadan herhangi 2 tanesini seçmek yeterlidir. Noktaların seçilme sırası doğru parçasını değiştirmez ($AB$ ile $BA$ aynı kiriştir):
$$\binom{8}{2} = \frac{8\times7}{2\times1} = 28 \text{ kiriş çizilebilir.}$$
2. Bir üçgen belirlemek için bu 8 noktadan herhangi 3 tanesini seçmek gerekir:
$$\binom{8}{3} = \frac{8\times7\times6}{3\times2\times1} = 56 \text{ farklı üçgen çizilebilir.}$$

□

**Örnek 5.6.** *Dışbükey (konveks) bir n kenarlı çokgenin toplam kaç köşegeni vardır? Formülü kombinasyon yardımıyla elde ediniz.*

*Çözüm.* $n$ kenarlı bir çokgenin $n$ adet köşesi bulunur. Bu $n$ köşe arasından seçilecek herhangi iki köşe bir doğru parçası tanımlar:

$$\text{Tüm Doğru Parçaları} = \binom{n}{2}$$

Bu doğru parçaları iki gruptan oluşur:

- Çokgenin kenarları (komşu iki köşeyi birleştiren $n$ adet doğru parçası).
- Köşegenler (komşu olmayan iki köşeyi birleştiren doğru parçaları).

Dolayısıyla köşegen sayısı, tüm doğru parçalarının sayısından kenar sayısı çıkarılarak bulunur:

$$\text{Köşegen Sayısı} = \binom{n}{2} - n = \frac{n(n-1)}{2} - n = \frac{n^2 - n - 2n}{2} = \frac{n(n-3)}{2}$$

Örneğin bir altıgen için ($n = 6$): $\frac{6\times 3}{2} = 9$ köşegen elde edilir. □

## 5.2 Simetri ve Pascal Bağıntısı

Kombinasyon katsayıları belirgin cebirsel ve simetrik özellikler taşır. Bu özelliklerin ilkini basit bir soruyla görebiliriz:

10 kişilik bir sınıftan kampa gidecek 8 öğrenciyi mi seçmek daha kolaydır, yoksa geride kalacak 2 öğrenciyi mi?

Kampa gidecek 8 kişiyi belirlemek, geride kalacak 2 kişiyi belirlemekle aynı seçim işlemidir. Gidecekler belirlendiğinde, kalanlar da kendiliğinden belirlenir. Formülle inceleyelim:

$$\binom{10}{8} = \frac{10!}{8! \times 2!} = 45 \quad \text{ve} \quad \binom{10}{2} = \frac{10!}{2! \times 8!} = 45$$

İki değer birbirine eşittir.

**Teorem 5.7** (Simetri Özelliği)**.** *Her* $0 \leq r \leq n$ *için:*

$$\binom{n}{r} = \binom{n}{n-r}$$

*Kanıt.* Formülü doğrudan uygulayalım:

$$\binom{n}{n-r} = \frac{n!}{(n-r)! \times (n-(n-r))!} = \frac{n!}{(n-r)! \times r!}$$

Çarpmanın değişme özelliği gereği paydadaki $(n-r)! \times r! = r! \times (n-r)!$'dir. Buradan:

$$\binom{n}{n-r} = \frac{n!}{r!(n-r)!} = \binom{n}{r}$$

sonucuna varılır. □

Bu özellik hesaplamaları önemli ölçüde sadeleştirir. Örneğin $\binom{100}{98}$ ifadesi doğrudan simetri kuralıyla açılır:

$$\binom{100}{98} = \binom{100}{100-98} = \binom{100}{2} = \frac{100 \times 99}{2} = 4950$$

### 5.2.1 Pascal Bağıntısı: Bir Elemanın Durumu

Şimdi kombinatorik ispatların klasik yaklaşımlarından birini inceleyelim. $n$ kişilik bir gruptan $r$ kişilik bir komite kurmak istiyoruz ($\binom{n}{r}$).

Bu gruptaki belirli bir kişiye odaklanalım; bu kişi Ahmet olsun. Olası tüm komiteleri Ahmet'in durumuna göre iki ayrık duruma ayırabiliriz:

1. **1. Durum (Ahmet'in yer aldığı komiteler):** Ahmet komitededir. Komitenin kalan $r-1$ üyesi, gruptaki diğer $n-1$ kişi arasından seçilir:

$$\binom{n-1}{r-1} \text{ seçenek.}$$

2. **2. Durum (Ahmet'in yer almadığı komiteler):** Ahmet komitede yer almaz. Dolayısıyla $r$ kişilik komitenin tamamı diğer $n-1$ kişi arasından seçilmelidir:

$$\binom{n-1}{r} \text{ seçenek.}$$

Ahmet bir komitede ya vardır ya da yoktur; bu iki durum ayrıktır. Toplama kuralı gereğince iki durumun toplamı tüm komitelerin sayısını verir.

**Teorem 5.8** (Pascal Bağıntısı)**.** *Her* $1 \leq r \leq n$ *için:*

$$\binom{n}{r} = \binom{n-1}{r-1} + \binom{n-1}{r}$$

*Kanıt.* Yukarıdaki kombinatorik akıl yürütme doğrudan bir ispattır. Cebirsel olarak doğrulamak için sağ taraftaki terimleri ortak paydaya getirelim:

$$\binom{n-1}{r-1} + \binom{n-1}{r} = \frac{(n-1)!}{(r-1)!(n-r)!} + \frac{(n-1)!}{r!(n-1-r)!}$$

Birinci kesri $r$ ile, ikinci kesri $(n-r)$ ile genişletelim:

$$\begin{aligned} &= \frac{r \cdot (n-1)!}{r!(n-r)!} + \frac{(n-r) \cdot (n-1)!}{r!(n-r)!} \\ &= \frac{[r + (n-r)] \cdot (n-1)!}{r!(n-r)!} \\ &= \frac{n \cdot (n-1)!}{r!(n-r)!} = \frac{n!}{r!(n-r)!} = \binom{n}{r} \end{aligned}$$

böylece eşitlik gösterilmiş olur. □

### 5.2.2 Çözümlü Örnekler

**Örnek 5.9.** $\binom{10}{3} + \binom{10}{4} + \binom{11}{5}$ *toplamını tek bir binom katsayısı olarak yazınız.*

*Çözüm.* Pascal bağıntısını adım adım uygulayalım. İlk iki terimi ele alalım: $\binom{10}{3} + \binom{10}{4}$. Pascal kuralı gereğince $\binom{n-1}{r-1} + \binom{n-1}{r} = \binom{n}{r}$ olduğundan:

$$\binom{10}{3} + \binom{10}{4} = \binom{11}{4}$$

olur. Bu sonucu üçüncü terimle birleştirelim:

$$\binom{11}{4} + \binom{11}{5}$$

Tekrar Pascal kuralı uygulanırsa $(n = 12, r = 5)$:

$$\binom{11}{4} + \binom{11}{5} = \binom{12}{5}$$

elde edilir. □

**Örnek 5.10** (Kombinatorik İspat: Başkanlı Komite Modeli)**.** *Her* $1 \le r \le n$ *için aşağıdaki eşitliğin doğruluğunu gösteriniz:*

$$n\binom{n-1}{r-1} = r\binom{n}{r}$$

*Çözüm.* Bu eşitlik cebirsel yoldan doğrulanabileceği gibi, Bölüm 10'da ayrıntılandıracağımız iki yoldan sayma (çifte sayım) yöntemiyle de kurulabilir.

$n$ kişi arasından $r$ kişilik bir komite ve bu komiteye bir **başkan** seçeceğimizi varsayalım. Bu seçim iki farklı sırayla yapılabilir:

- **1. Yol (Önce başkan, sonra üyeler):** $n$ aday arasından başkan seçilir ($n$ seçenek). Kalan $n-1$ kişi arasından komitenin diğer $r-1$ üyesi belirlenir ($\binom{n-1}{r-1}$ seçenek). Çarpma kuralıyla:

$$n \times \binom{n-1}{r-1}$$

- **2. Yol (Önce komite, sonra başkan):** $n$ kişi arasından $r$ kişilik komite oluşturulur ($\binom{n}{r}$ seçenek). Seçilen $r$ kişi arasından bir başkan belirlenir ($r$ seçenek). Çarpma kuralıyla:

$$\binom{n}{r} \times r$$

Her iki sayım da aynı kümenin eleman sayısını verdiğinden:

$$n\binom{n-1}{r-1} = r\binom{n}{r}$$

eşitliği sağlanır. □

**Örnek 5.11** (C Dilinde Pascal Bağıntısıyla Rekürsif Kombinasyon)**.** *Pascal bağıntısını kullanarak kombinasyon hesaplayan bir C fonksiyonu yazınız ve algoritmayı açıklayınız.*

*Çözüm.* Pascal bağıntısı $\binom{n}{r} = \binom{n-1}{r-1} + \binom{n-1}{r}$ doğal bir rekürsiyon (rekürsiyon) yapısı sunar. Taban durumları ise $r = 0$ ve $r = n$ olduğunda değerin 1 olmasıdır:

```c
// Pascal bagintisi ile kombinasyon hesabi
long long kombinasyon(int n, int r) {
    // Taban durumlari (base cases)
    if (r == 0 || r == n) {
        return 1;
    }
    // Ozyinelemeli adim: Pascal kurali
    return kombinasyon(n - 1, r - 1) + kombinasyon(n - 1, r);
}
```

Bu fonksiyon matematiksel tanımı doğrudan uygular; ancak büyük $n$ değerlerinde aynı alt problemleri defalarca hesaplar. Bu verimsizlik Bölüm 26'da dinamik programlama yaklaşımıyla giderilecektir. □

## 5.3 Yıldızlar ve Bölücüler (Stars and Bars)

Şimdiye kadar ele aldığımız kombinasyon problemlerinde üzerinde çalışılan nesneler birbirinden ayırt edilebilirdi. Dağıtılacak nesnelerin **birbiriyle tamamen özdeş** olduğu durumlarda sayım nasıl yapılır?

Somut bir problem ele alalım: Birbirinin tıpatıp aynısı olan 7 elmayı 3 çocuğa $(A, B, C)$ kaç farklı şekilde paylaştırabiliriz?

Elmalar özdeş olduğundan hangi elmanın kime verildiği değil, **her çocuğun kaç elma aldığı** önemlidir. $A$'nın aldığı elma sayısına $x_1$, $B$'ninkine $x_2$, $C$'ninkine $x_3$ dersek, problem şu denkleme denktir:

$$x_1 + x_2 + x_3 = 7 \quad (x_1, x_2, x_3 \in \mathbb{N},\ \ x_i \geq 0)$$

Bu denklemin çözümlerini tek tek listelemek yerine, kombinatorikte sıkça başvurulan **Yıldızlar ve Bölücüler** (*Stars and Bars*) modelini kullanırız.

### 5.3.1 Yöntemin Mantığı

7 adet özdeş elmayı yan yana dizilmiş 7 adet yıldız ($\star$) ile gösterelim:

$$\star \quad \star \quad \star \quad \star \quad \star \quad \star \quad \star$$

Bu 7 yıldızı 3 gruba ayırmak için araya $3 - 1 = 2$ **adet ayraç (bölücü çizgi, $|$)** yerleştirmek yeterlidir.

Örnek dizilimler:

- $\star\star\,|\,\star\,\star\,\star\,|\,\star\star \implies x_1 = 2,\ \ x_2 = 3,\ \ x_3 = 2$
- $|\,\star\,\star\,\star\star\,|\,\star\,\star\,\star \implies x_1 = 0,\ \ x_2 = 4,\ \ x_3 = 3$ (ilk çocuk hiç elma almadı)
- $\star\,\star\,\star\,\star\,\star\,\star\,\star\,|| \implies x_1 = 7,\ \ x_2 = 0,\ \ x_3 = 0$ (tüm elmaları ilk çocuk aldı)

Her paylaşım, 7 yıldız ($\star$) ve 2 bölücüden ($|$) oluşan toplam $7+2 = 9$ karakterlik bir dizilimle birebir eşleşir.

Soru şuna indirgenir: Yan yana duran 9 boş pozisyondan hangi 2 tanesine bölücü ($|$) yerleştirilecektir? Geriye kalan yerlere yıldızlar gelecektir. Sıra önemsiz olduğundan bu doğrudan bir kombinasyondur:

$$\binom{9}{2} = \frac{9 \times 8}{2 \times 1} = 36$$

Demek ki 7 özdeş elma 3 çocuğa 36 farklı şekilde dağıtılabilir.

**Teorem 5.12** (Yıldızlar ve Bölücüler Teoremi — Negatif Olmayan Çözümler)**.** *$n$ adet özdeş nesnenin $k$ farklı kutuya dağıtılma biçimlerinin (veya $x_1+x_2+\cdots+x_k = n$ denkleminin $x_i \geq 0$ tam sayı çözümlerinin) toplam sayısı:*

$$\binom{n+k-1}{k-1}$$

*formülüyle hesaplanır.*

*Kanıt.* $n$ adet yıldız ve kutuları birbirinden ayırmak için $k-1$ adet bölücü kullanılır. Toplam pozisyon sayısı $n + (k-1) = n + k - 1$'dir. Bu pozisyonlardan bölücülerin yerleşeceği $k-1$ konumu belirlemenin yolu $\binom{n+k-1}{k-1}$ tanedir. $\square$

### 5.3.2 Her Kutuya En Az Bir Nesne Düşmesi Durumu ($x_i \geq 1$)

Her çocuğun **en az bir elma** alması koşulu konursa, yani $x_1 + x_2 + \cdots + x_k = n$ denkleminin **pozitif tam sayı** ($x_i \geq 1$) çözümleri aranırsa iki yoldan ilerlenebilir:

**1. Yol (Önden Dağıtma):** $n$ elmadan önce her çocuğa birer tane verilir. Toplam $k$ elma dağıtılmış olur ve geriye $n - k$ elma kalır. Kalan $n - k$ elma çocuklara serbestçe (negatif olmayan biçimde) paylaştırılabilir:

$$\binom{(n-k)+k-1}{k-1} = \binom{n-1}{k-1}$$

**2. Yol (Aralıklara Bölücü Koyma):** $n$ adet yıldız yan yana dizildiğinde aralarında tam $n - 1$ adet boşluk bulunur:

$$\star \underbrace{\quad}_{\text{boşluk}} \star \underbrace{\quad}_{\text{boşluk}} \star \ \ldots \ \star$$

Hiçbir kutunun boş kalmaması, iki bölücünün aynı boşluğa konmaması ve uçlara bölücü yerleştirilmemesi demektir. Dolayısıyla $k - 1$ adet bölücü, bu $n - 1$ boşluk arasından seçilen yerlere konur:

$$\binom{n-1}{k-1}$$

**Teorem 5.13** (Pozitif Tam Sayı Çözümleri)**.** *$x_1 + x_2 + \cdots + x_k = n$ denkleminin pozitif tam sayılardaki ($x_i \geq 1$) çözüm sayısı:*

$$\binom{n-1}{k-1}$$

*formülüyle verilir.*

### 5.3.3 Çözümlü Örnekler

**Örnek 5.14.** *Bir pastanede 4 çeşit kurabiye (çikolatalı, fındıklı, tarçınlı, vanilyalı) satılmaktadır ve her çeşitten yeterince vardır. Bir müşteri toplam 10 adet kurabiye seçerek bir paket yaptırmak istediğinde kaç farklı paket oluşturabilir?*

*Çözüm.* Kurabiyeler aynı çeşit içinde özdeştir. Seçilecek miktarlara $x_1, x_2, x_3, x_4$ dersek:

$$x_1 + x_2 + x_3 + x_4 = 10 \quad (x_i \geq 0)$$

Burada $n = 10$ nesne ve $k = 4$ farklı kategori söz konusudur. Yıldızlar ve bölücüler teoremini uygulayalım:

$$\binom{n+k-1}{k-1} = \binom{10+4-1}{4-1} = \binom{13}{3}$$

Değeri hesaplayalım:

$$\binom{13}{3} = \frac{13 \times 12 \times 11}{3 \times 2 \times 1} = \frac{13 \times 12 \times 11}{6} = 13 \times 2 \times 11 = 286$$

Farklı paket seçeneklerinin sayısı 286'dır. □

**Örnek 5.15** (Değişken Değiştirme Tekniği)**.** $a + b + c = 18$ *denkleminin* $a \geq 2$, $b \geq 3$ *ve* $c \geq 0$ *koşullarını sağlayan kaç farklı tam sayı çözümü vardır?*

*Çözüm.* Kısıtları standart $\geq 0$ biçimine dönüştürmek için değişken değiştirme uygularız:

- $a \geq 2 \implies a' = a - 2 \geq 0 \implies a = a' + 2$
- $b \geq 3 \implies b' = b - 3 \geq 0 \implies b = b' + 3$
- $c \geq 0 \implies c' = c \geq 0 \implies c = c'$

Bu tanımları denklemde yerine koyalım:

$$(a' + 2) + (b' + 3) + c' = 18 \implies a' + b' + c' + 5 = 18 \implies a' + b' + c' = 13$$

Böylece $n = 13$ ve $k = 3$ parametrelerine sahip standart biçime ulaşırız:

$$\binom{13 + 3 - 1}{3 - 1} = \binom{15}{2} = \frac{15 \times 14}{2} = 105$$

farklı çözüm vardır. □

**Örnek 5.16** (Üstten Kısıtlı Dağıtım)**.** $x_1 + x_2 + x_3 = 9$ *denkleminin* $0 \leq x_i \leq 4$ *kısıtını sağlayan tam sayı çözüm sayısını bulunuz.*

*Çözüm.* Değişkenlerin alabileceği en büyük değer 4 olarak sınırlandırılmıştır. Çözümü tümleyen küme üzerinden kuralım:

$$\text{İstenen} = \text{Tüm Çözümler} - \text{En az bir değişkenin } \geq 5 \text{ olduğu durumlar}$$

1. **Tüm Çözümler ($x_i \geq 0$):**

$$\binom{9 + 3 - 1}{3 - 1} = \binom{11}{2} = \frac{11 \times 10}{2} = 55$$

2. **Bir değişkenin $\geq 5$ olması:** Değişkenlerden biri (örneğin $x_1$) en az 5 olsun: $x_1 \geq 5$. $x_1' = x_1 - 5 \geq 0$ dönüşümüyle:

$$x_1' + x_2 + x_3 = 9 - 5 = 4$$

Bu denklemin çözüm sayısı:

$$\binom{4 + 3 - 1}{3 - 1} = \binom{6}{2} = 15$$

En az 5 olacak değişkeni seçmenin $\binom{3}{1} = 3$ yolu vardır ($x_1, x_2$ veya $x_3$).

3. **İki değişken birden $\geq 5$ olabilir mi?** İki değişken aynı anda $\geq 5$ olsaydı toplam en az $5+5=10$ olurdu; oysa denklem toplamı 9'dur. Dolayısıyla bu durum gerçekleşemez.

Koşulu bozan durumların sayısı $3 \times 15 = 45$'tir. O hâlde aranan çözüm sayısı:

$$55 - 45 = 10$$

olur. □

## 5.4 Karışık Seçme Problemleri

Kombinatorik problemlerinde çoğu zaman kombinasyon, permütasyon ve durum analizi bir arada kullanılır. Temel yaklaşım kuralı şudur: **Önce seçim yapılır (kombinasyon), gerek duyuluyorsa ardından sıralama uygulanır (permütasyon).**

### 5.4.1 Çözümlü Örnekler

**Örnek 5.17.** 6 *erkek ve* 5 *kadın arasından* 4 *kişilik bir temsilci heyeti seçilecektir.*

1. *Heyette* ***en az*** 2 ***kadın*** *bulunması şartıyla kaç farklı seçim yapılabilir?*
2. *Belirli iki kişinin (Ahmet ve Zeynep)* ***birlikte bulunmaması*** *şartıyla kaç farklı heyet kurulabilir?*

*Çözüm.* 1. Heyetteki kadın sayısı $2, 3$ veya 4 olabilir. Bu durumlar ayrık olduğundan ayrı ayrı hesaplanıp toplanır:

- **2 Kadın, 2 Erkek:** $\binom{5}{2} \times \binom{6}{2} = 10 \times 15 = 150$
- **3 Kadın, 1 Erkek:** $\binom{5}{3} \times \binom{6}{1} = 10 \times 6 = 60$
- **4 Kadın, 0 Erkek:** $\binom{5}{4} \times \binom{6}{0} = 5 \times 1 = 5$

Toplama kuralı gereğince:

$$150 + 60 + 5 = 215 \text{ farklı heyet kurulabilir.}$$

2. Tümleyen yaklaşımını kullanalım:

$$\text{İstenen} = \text{Tüm Heyetler} - \text{Ahmet ile Zeynep'in Bulunduğu Heyetler}$$

- Toplam $6 + 5 = 11$ kişi vardır. Kısıt olmaksızın 4 kişi:

$$\binom{11}{4} = \frac{11 \times 10 \times 9 \times 8}{4 \times 3 \times 2 \times 1} = 330$$

- Ahmet ve Zeynep'in ikisinin de seçildiği durumlar: Bu iki kişi ayrıldıktan sonra kalan $11-2=9$ kişiden komiteyi tamamlayacak $4-2=2$ kişi seçilir:

$$\binom{9}{2} = \frac{9\times 8}{2} = 36$$

Buna göre Ahmet ve Zeynep'in birlikte yer almadığı heyet sayısı:

$$330 - 36 = 294$$

olur.

□

**Örnek 5.18** (Izgarada Dikdörtgen Sayma)**.** *Yatayda* 5 *paralel doğru ile düşeyde* 7 *paralel doğru birbirini dik kesmektedir. Kenarları bu doğrular üzerinde bulunan kaç farklı dikdörtgen oluşur?*

*Çözüm.* Bir dikdörtgen tanımlamak için:

- Alt ve üst kenarı belirleyecek 2 **adet yatay doğru**,
- Sol ve sağ kenarı belirleyecek 2 **adet düşey doğru**

seçmek gerekir. Seçilen her yatay doğru çifti ile düşey doğru çifti tek bir dikdörtgen sınırlar:

- 5 yatay doğrudan 2 tanesini seçmenin yolu: $\binom{5}{2} = 10$.
- 7 düşey doğrudan 2 tanesini seçmenin yolu: $\binom{7}{2} = 21$.

Çarpma kuralı uyarınca toplam dikdörtgen sayısı:

$$\binom{5}{2} \times \binom{7}{2} = 10 \times 21 = 210$$

olur. Genel olarak $m$ yatay ve $n$ düşey doğru arasında $\binom{m}{2}\binom{n}{2}$ adet dikdörtgen bulunur. □

**Örnek 5.19** (Standart İskambil Eli Sayımı)**.** 52 *kartlık standart bir iskambil destesinde her biri* 13*'er karttan oluşan* 4 *simge (Kupa, Karo, Maça, Sinek) ve her simgede* 13 *farklı değer* $(A, 2, 3, \ldots, 10, J, Q, K)$ *bulunur. Bu desteden seçilen* 5 *kartlık bir elde* ***tam olarak bir per*** *(aynı değerden* 2 *kart ve diğer* 3 *kartın değerleri hem birbirinden hem de perden farklı) bulunma sayısını hesaplayınız.*

*Çözüm.* Sayım adımlarını sırayla oluşturalım:

1. **1. Adım (Perin değerini seçmek):** Perin hangi değerden oluşacağı belirlenir. 13 farklı değerden biri seçilir: $\binom{13}{1} = 13$ seçenek.

2. **2. Adım (Perin kartlarını seçmek):** Seçilen değerin 4 simgesinden 2 tanesi belirlenir: $\binom{4}{2} = 6$ seçenek.

3. **3. Adım (Kalan 3 kartın değerlerini seçmek):** Elin başka bir per içermemesi için diğer 3 kartın değerleri birbirinden ve ilk değerden farklı olmalıdır. Kalan 12 değer arasından 3 farklı değer seçilir: $\binom{12}{3} = \frac{12\times11\times10}{6} = 220$ seçenek.

4. **4. Adım (Kalan 3 kartın simgelerini seçmek):** Belirlenen 3 değerin her biri için 4 simgeden biri seçilir: $\binom{4}{1} \times \binom{4}{1} \times \binom{4}{1} = 4^3 = 64$ seçenek.

Çarpma kuralı gereğince tam olarak bir per içeren ellerin sayısı:

$$\binom{13}{1} \times \binom{4}{2} \times \binom{12}{3} \times 4^3 = 13 \times 6 \times 220 \times 64 = 1\,098\,240$$

olarak hesaplanır. □

# Bölüm 6

# Binom Açılımı ve Özdeşlikler

Bir parantezin karesini açmak çoğumuza tanıdık gelir:

$$(a+b)^2 = a^2 + 2ab + b^2.$$

Fakat bu satırdaki 2 sayısının nereden geldiği üzerinde durmak, yalnızca formülü ezberlemekten çok daha öğreticidir. $ab$ terimi iki ayrı seçimden doğar: ilk parantezden $a$, ikinciden $b$ seçilebilir; ya da ilkinden $b$, ikincisinden $a$ seçilebilir. Aynı çarpıma ulaştıran seçimleri saydığımız anda, cebir ile kombinatorik arasında doğrudan bir köprü kurmuş oluruz.

Buradaki temel fikir şudur: $(a+b)^n$ ifadesini açarken her parantezden ya $a$'yı ya da $b$'yi seçeriz. Benzer terimleri kaç farklı seçimin ürettiğini belirlemek ise doğrudan Bölüm 5'te ele aldığımız kombinasyon hesabıdır. Bu bölümde önce terimlerin küçük kuvvetlerde nasıl ortaya çıktığını adım adım inceleyecek, ardından binom teoremini kuracağız. Devamında binom katsayılarının oluşturduğu Pascal üçgenine, bu üçgenden doğan temel özdeşliklere ve Bölüm 10'da daha kapsamlı ele alacağımız çifte sayım fikrinin ilk örnekleri olan kombinatorik ispatlara yer vereceğiz.

## 6.1 Elle Açmadan Teoreme

Yüksek kuvvetli bir parantezi açarken yapılan en yaygın hata, işlemi yalnızca uzun bir çarpma alıştırması olarak görmektir. Oysa parantez sayısı arttıkça asıl gereken şey daha hızlı çarpmak değil, ortaya çıkan terimlerin hangi seçimlerden türediğini kavramaktır.

İlk olarak $(a+b)^3$ ifadesini adım adım ele alalım:

$$(a+b)^3 = (a+b)(a+b)(a+b).$$

Çarpımın açılımında bir terim oluşturmak için üç parantezin her birinden tam bir terim seçilir. Örneğin $a^2b$ terimini elde etmek istiyorsak iki parantezden $a$, bir

parantezden $b$ seçmeliyiz. $b$'nin hangi parantezden geldiğine göre üç farklı seçim mümkündür:

$$a \cdot a \cdot b, \qquad a \cdot b \cdot a, \qquad b \cdot a \cdot a.$$

Çarpma işlemi değişmeli olduğundan bu üç seçim de aynı $a^2b$ çarpımını verir. Buna karşılık $a^3$ yalnızca bütün parantezlerden $a$ seçilerek, $b^3$ de yalnızca bütün parantezlerden $b$ seçilerek oluşur. Demek ki

$$(a+b)^3 = a^3 + 3a^2b + 3ab^2 + b^3.$$

Burada katsayıların $1, 3, 3, 1$ çıkması tesadüf değildir. $a^2b$ teriminin katsayısı, üç parantezden hangisinden $b$ seçileceğinin sayısıdır; yani $\binom{3}{1} = 3$'tür. $ab^2$ için de $b$ alınacak iki parantez seçilir; bunun sayısı $\binom{3}{2} = 3$'tür.

**Tanım 6.1** (Binom ve binom açılımı)**.** *İki terimin toplamı ya da farkı biçimindeki $a+b$ ifadesine* ***binom*** *denir. $n \in \mathbb{N}$ için $(a+b)^n$ ifadesinin çarpma işlemi yapılarak toplam biçiminde yazılmasına* ***binom açılımı*** *denir.*

Şimdi genel duruma geçelim. $(a+b)^n$ çarpımında $n$ tane parantez yer alır. Açılımdaki $a^{n-k}b^k$ terimini elde etmek için bunların tam $k$ tanesinden $b$, kalan $n-k$ tanesinden $a$ seçmek gerekir. $b$ seçilecek $k$ parantezi belirlemek, $n$ nesneden $k$ tanesini sırasız seçmek demektir. Bu yüzden seçim sayısı doğrudan $\binom{n}{k}$ olur.

**Teorem 6.2** (Binom teoremi)**.** *$a$ ve $b$ gerçel sayılar, $n \in \mathbb{N}$ olsun. O zaman*

$$(a+b)^n = \sum_{k=0}^{n} \binom{n}{k} a^{n-k}b^k.$$

*Başka bir deyişle,*

$$(a+b)^n = \binom{n}{0}a^n + \binom{n}{1}a^{n-1}b + \binom{n}{2}a^{n-2}b^2 + \cdots + \binom{n}{n}b^n.$$

*Kanıt.* $n$ parantezin her birinden $a$ veya $b$ seçilir. $a^{n-k}b^k$ teriminin oluşması için tam $k$ parantezden $b$ seçilmelidir. Bu $k$ parantezin seçimi $\binom{n}{k}$ farklı yolla yapılır ve her seçim aynı çarpımı verir. Dolayısıyla söz konusu terimin katsayısı $\binom{n}{k}$'dır.

$k$ indisi, 0'dan $n$'ye kadar bütün değerleri alabilir. Bu değerlerin her biri farklı bir $b$ üssü ürettiğinden benzer başka terim kalmaz; bütün terimler toplandığında teoremdeki açılım elde edilir. □

Formülü uygularken iki düzeni birlikte takip etmek işi kolaylaştırır. Soldan sağa doğru ilerledikçe $a$'nın üssü birer birer azalırken $b$'nin üssü birer birer artar. Eşzamanlı olarak binom katsayıları $\binom{n}{0}, \binom{n}{1}, \ldots, \binom{n}{n}$ sırasında ilerler. Fark içeren ifadelerde ise ikinci terimin işaretini $b$'nin içinde taşımak güvenli bir yoldur: $(a-b)^n$ açılımında formüldeki $b$ yerine $-b$ yazılır.

### 6.1.1 Çözümlü örnekler

**Örnek 6.3.** $(2x-3)^4$ *ifadesini binom teoremiyle açınız.*

*Çözüm.* Burada $a = 2x$, $b = -3$ ve $n = 4$'tür. Dördüncü kuvvete karşılık gelen binom katsayıları $1, 4, 6, 4, 1$ olduğundan temel açılımı yazalım:

$$(2x-3)^4 = (2x)^4 + 4(2x)^3(-3) + 6(2x)^2(-3)^2 + 4(2x)(-3)^3 + (-3)^4.$$

Terimleri sadeleştirdiğimizde

$$\boxed{(2x-3)^4 = 16x^4 - 96x^3 + 216x^2 - 216x + 81}$$

elde edilir. İşaretlerin sırayla değişmesi, $(-3)$'ün tek kuvvetlerinin negatif, çift kuvvetlerinin pozitif olmasından kaynaklanır. □

**Örnek 6.4.** $(x^2+3x)^6$ *açılımında* $x^9$ *teriminin katsayısını bulunuz.*

*Çözüm.* İfadenin tamamını açmaya gerek yoktur. Genel terim, ikinci terimin $k$ kez seçildiği durumda

$$\binom{6}{k}(x^2)^{6-k}(3x)^k = \binom{6}{k}3^k x^{12-2k+k} = \binom{6}{k}3^k x^{12-k}$$

olur. İstenen kuvvet 9 olduğuna göre $12-k=9$, yani $k=3$ olmalıdır. Buradan katsayı

$$\binom{6}{3}3^3 = 20 \cdot 27 = 540$$

bulunur. Dolayısıyla aranan terim $540x^9$'dur. □

**Örnek 6.5.**

$$\left(x^2 + \frac{2}{x}\right)^6$$

*açılımındaki sabit terimi bulunuz.*

*Çözüm.* Sabit terimde $x$'in üssü sıfır olmalıdır. Genel terimi yazalım:

$$\binom{6}{k}(x^2)^{6-k}\left(\frac{2}{x}\right)^k = \binom{6}{k}x^{12-2k}\cdot 2^k x^{-k} = \binom{6}{k}2^k x^{12-3k}.$$

Sabit terim için $12 - 3k = 0$ denklemi çözülürse $k = 4$ elde edilir. O hâlde sabit terim

$$\binom{6}{4} 2^4 = 15 \cdot 16 = 240$$

olur. □

**Örnek 6.6.** $(1+t)^8$ *açılımındaki bütün katsayıların toplamını, açılımı yapmadan bulunuz.*

*Çözüm.* Katsayılar toplamını bulmak için değişkene $t = 1$ değerini vermek yeterlidir; böylece her $t^k$ terimi 1 çarpanına dönüşür:

$$(1 + 1)^8 = 2^8 = 256.$$

Bu sonuç, $\binom{8}{0} + \binom{8}{1} + \cdots + \binom{8}{8}$ toplamının 256 olduğunu doğrudan gösterir. □

**Sık yapılan hata:** Genel terimde $k$ indisi, ikinci terimin kaç kez seçildiğini belirtir. Bu kitapta binom teoremini daima $\binom{n}{k} a^{n-k} b^k$ biçiminde ele alacağız.

## 6.2 Pascal Üçgeni

Binom teoremi açılım katsayılarını kombinasyon formülüyle verir. Aynı katsayıları $n = 0, 1, 2, \ldots$ için satır satır alt alta yazdığımızda son derece düzenli bir tablo ortaya çıkar. Örneğin $(a+b)^4$ açılımındaki katsayılar sırasıyla $\binom{4}{0}, \binom{4}{1}, \ldots, \binom{4}{4}$, yani $1, 4, 6, 4, 1$'dir. Bir sonraki satırı elde etmek için faktöriyelleri baştan hesaplamak gerekmez; her sayı, hemen üstündeki iki sayının toplamından elde edilebilir.

**Tanım 6.7** (Pascal üçgeni)**.** *$n$. satırında soldan sağa $\binom{n}{0}, \binom{n}{1}, \ldots, \binom{n}{n}$ binom katsayılarının yer aldığı üçgensel sayı dizilimine* ***Pascal üçgeni*** *denir. Satır numaralandırması 0'dan başlar.*

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0. satır: | 1 | | | | | | |
| 1. satır: | 1 | 1 | | | | | |
| 2. satır: | 1 | 2 | 1 | | | | |
| 3. satır: | 1 | 3 | 3 | 1 | | | |
| 4. satır: | 1 | 4 | 6 | 4 | 1 | | |
| 5. satır: | 1 | 5 | 10 | 10 | 5 | 1 | |
| 6. satır: | 1 | 6 | 15 | 20 | 15 | 6 | 1 |

Üçgenin kenarlarındaki sayıların daima 1 olması, $\binom{n}{0} = \binom{n}{n} = 1$ eşitliklerinin sonucudur. Her satırın ortasına göre simetrik durması ise Bölüm 5'te gördüğümüz $\binom{n}{k} = \binom{n}{n-k}$ özelliğinin görsel yansımasıdır.

Pascal bağıntısının komite seçimi üzerinden kombinatorik ve cebirsel ispatını Bölüm 5'te kurmuştuk. Bu bağıntıyı üçgen üzerinde şöyle okuruz:

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k} \qquad (1 \le k \le n-1).$$

Demek ki Pascal üçgeninde kenarlar dışındaki her sayı, sol üst ve sağ üst çaprazındaki iki sayının toplamıdır. Böylece Bölüm 5'teki "belirli bir kişi grupta yer alıyor mu?" ayrımı, üçgende komşu iki sayının neden toplandığını da açıklar.

Pascal üçgeni, binom açılımlarını hızlıca yazmak için oldukça pratiktir. Örneğin 7. satır $1, 7, 21, 35, 35, 21, 7, 1$ olduğundan

$$\begin{aligned}(u+v)^7 = {} & u^7 + 7u^6v + 21u^5v^2 + 35u^4v^3 \\ & + 35u^3v^4 + 21u^2v^5 + 7uv^6 + v^7.\end{aligned}$$

Her terimde üslerin toplamının 7 olduğunu kontrol etmek, olası yazım hatalarını yakalamak için iyi bir yöntemdir.

### 6.2.1 Çözümlü örnekler

**Örnek 6.8.** *6. satırı kullanarak Pascal üçgeninin 7. satırını oluşturunuz. Ardından $(p+q)^7$ açılımının katsayılarını yazınız.*

*Çözüm.* 6. satır $1, 6, 15, 20, 15, 6, 1$ değerlerinden oluşur. Yeni satırın kenarlarına 1 yazılır; aradaki sayılar ardışık üst komşuların toplamıdır:

$$1,\ 1+6,\ 6+15,\ 15+20,\ 20+15,\ 15+6,\ 6+1,\ 1.$$

Böylece 7. satır $1, 7, 21, 35, 35, 21, 7, 1$ olarak bulunur. $(p+q)^7$ açılımının katsayıları da doğrudan bu dizidir. □

**Örnek 6.9.** *Pascal üçgenini kullanarak $(x-1)^5$ ifadesini açınız.*

*Çözüm.* 5. satırdaki katsayılar $1, 5, 10, 10, 5, 1$'dir. İkinci terim $-1$ olduğundan işaretler ardışık olarak değişir:

$$\boxed{(x-1)^5 = x^5 - 5x^4 + 10x^3 - 10x^2 + 5x - 1.}$$

Katsayıların büyüklüğü Pascal üçgeninden, işaretler ise $(-1)^k$ çarpanından gelir. □

**Örnek 6.10.** *Pascal üçgeninin 8. satırındaki ortanca terimi yalnızca bir önceki satırdan yararlanarak bulunuz.*

*Çözüm.* 7. satırdaki ortadaki iki sayı 35 ve 35'tir. 8. satırın ortasındaki sayı bu iki değerin toplamıdır:

$$\binom{8}{4} = 35 + 35 = 70.$$

Bu değer, $\frac{8\cdot 7\cdot 6\cdot 5}{4\cdot 3\cdot 2\cdot 1} = 70$ formülüyle de doğrulanır. □

**Örnek 6.11.** $1, 4, 6, 4, 1$ *sayılarının* $(a+b)^4$ *açılımının katsayıları olmasını parantez seçimi üzerinden açıklayınız.*

*Çözüm.* Dört parantezin hiçbirinden $b$ seçilmezse yalnızca $a^4$ terimi oluşur; katsayı $\binom{4}{0} = 1$'dir. Bir parantezden $b$ seçmek için 4 farklı seçenek vardır ve $a^3b$ katsayısı $\binom{4}{1} = 4$ olur. İki parantezden $b$ seçmenin yolu $\binom{4}{2} = 6$ tanedir ve bu da $a^2b^2$ terimini verir. Simetriden ötürü sonraki katsayılar 4 ve 1 olarak gelir. Sonuçta

$$(a+b)^4 = a^4 + 4a^3b + 6a^2b^2 + 4ab^3 + b^4$$

ifadesi elde edilir. □

## 6.3 Özdeşlikler

Bir özdeşlik, değişkenlerin tanım kümesindeki her değer için sağlanan bir eşitliktir. Binom teoremi yalnızca parantez açmaya yaramaz; $a$ ve $b$ yerine uygun sayılar yerleştirildiğinde kombinasyon katsayılarına dair pek çok toplam bağıntısı kendiliğinden belirir.

En temel durumla başlayalım. $(1+1)^n$ ifadesinde her terim 1 ile çarpılmış olur. Binom teoremi bu durumda satırdaki tüm katsayıların toplamını verir:

$$\binom{n}{0} + \binom{n}{1} + \cdots + \binom{n}{n} = 2^n.$$

Bu eşitlik, Bölüm 1'de ele aldığımız "$n$ elemanlı bir kümenin $2^n$ alt kümesi vardır" gerçeğiyle tam bir uyum içindedir: $\binom{n}{k}$ değeri tam $k$ elemanlı alt kümelerin sayısıdır ve bütün $k$ değerleri üzerinden toplamak tüm alt kümeleri sayar.

**Teorem 6.12** (Satır toplamı ve alternatif toplam)**.** $n \in \mathbb{N}$ *için*

$$\sum_{k=0}^{n} \binom{n}{k} = 2^n.$$

*Ayrıca* $n \geq 1$ *ise*

$$\sum_{k=0}^{n} (-1)^k \binom{n}{k} = 0.$$

*Kanıt.* Binom teoreminde $a = 1$, $b = 1$ alındığında $(1+1)^n = \sum_{k=0}^{n} \binom{n}{k}$ olur ve ilk eşitlik çıkar.

İkinci eşitlik için $a = 1$, $b = -1$ seçilir. $n \geq 1$ olduğundan $(1-1)^n = 0$'dır. Açılımın sağ tarafı ise $\binom{n}{0} - \binom{n}{1} + \binom{n}{2} - \cdots + (-1)^n \binom{n}{n}$ biçimindedir. Böylece işaret değiştiren toplamın sıfır olduğu görülür. □

Alternatif toplam, çift ve tek indisli katsayıların dengeli bir dağılım sergilediğini ortaya koyar. Çift indisli katsayıların toplamına $C$, tek indisli katsayıların toplamına $T$ dersek; satır toplamı $C + T = 2^n$, alternatif toplam ise $C - T = 0$ verir. Bu iki eşitlik birlikte çözüldüğünde

$$C = T = 2^{n-1}$$

bulunur. Yani boş kümeden farklı $n$ elemanlı bir kümenin çift sayıda eleman içeren alt kümeleri ile tek sayıda eleman içeren alt kümeleri eşit sayıdadır.

**Teorem 6.13** (Hokey sopası özdeşliği)**.** $n \geq r \geq 0$ *için*

$$\binom{r}{r} + \binom{r+1}{r} + \binom{r+2}{r} + \cdots + \binom{n}{r} = \binom{n+1}{r+1}.$$

*Kanıt.* Pascal bağıntısını ardışık olarak uygulayalım. İlk iki terim için $\binom{r}{r} = \binom{r+1}{r+1}$ yazarak

$$\binom{r}{r} + \binom{r+1}{r} = \binom{r+1}{r+1} + \binom{r+1}{r} = \binom{r+2}{r+1}$$

elde ederiz. Bu sonuca sıradaki $\binom{r+2}{r}$ terimi eklendiğinde Pascal bağıntısı yeniden çalışır ve $\binom{r+3}{r+1}$ bulunur. Bu zincirleme sadeleşme son terime kadar devam eder ve geriye yalnızca

$$\binom{n}{r+1} + \binom{n}{r} = \binom{n+1}{r+1}$$

kalır. □

Bu özdeşliğe, Pascal üçgeninde sabit bir köşegen boyunca ilerleyip son adımda yön değiştirildiğinde ortaya çıkan şekilden ötürü hokey sopası adı verilmiştir. Bölüm 5'te incelediğimiz $\binom{10}{3} + \binom{10}{4} + \binom{11}{5}$ toplamı bu zincirin kısa bir örneğiydi; burada aynı mekanizmayı genel hâliyle kurmuş olduk. Buradaki asıl kazanım, uzun bir toplamın tek bir kombinasyon terimine indirgenebilmesidir.

### 6.3.1 Çözümlü örnekler

**Örnek 6.14.** 10 *elemanlı bir kümenin çift sayıda elemana sahip kaç alt kümesi vardır?*

*Çözüm.* Küme boş kümeden farklıdır; dolayısıyla çift ve tek sayıda eleman içeren alt kümelerin sayıları eşittir. Toplam alt küme sayısı $2^{10}$ olduğuna göre aranan sayı bunun yarısıdır:

$$2^{10-1} = 2^9 = 512.$$

Sıfır çift bir sayı olduğundan, tek elemanı bulunmayan boş küme de bu sayıma dahildir. □

**Örnek 6.15.** *Aşağıdaki toplamı hesaplayınız:*

$$\binom{3}{3} + \binom{4}{3} + \binom{5}{3} + \binom{6}{3} + \binom{7}{3}.$$

*Çözüm.* İfade, $r = 3$ ve $n = 7$ parametreleriyle hokey sopası özdeşliğine tam olarak uyar:

$$\sum_{i=3}^{7} \binom{i}{3} = \binom{8}{4} = 70.$$

Terimleri tek tek toplamak ($1 + 4 + 10 + 20 + 35 = 70$) yerine özdeşlik sonucu doğrudan verir. □

**Örnek 6.16.**

$$\binom{9}{0} - \binom{9}{1} + \binom{9}{2} - \cdots - \binom{9}{9}$$

*toplamını bulunuz.*

*Çözüm.* İşaretler almaşık ilerlediği için bu ifade $n = 9$ durumundaki alternatif toplamdır. $9 \geq 1$ olduğundan doğrudan

$$(1-1)^9 = 0$$

elde edilir. Son terimin eksi işaretli oluşu, 9 tek sayı olduğundan $(-1)^9 = -1$ gelmesindendir. □

**Örnek 6.17.** *İlk $n$ pozitif tam sayının toplamını hokey sopası özdeşliğinden elde ediniz.*

*Çözüm.* Özdeşlikte $r = 1$ alalım:

$$\binom{1}{1} + \binom{2}{1} + \cdots + \binom{n}{1} = \binom{n+1}{2}.$$

$\binom{i}{1} = i$ olduğundan sol taraf doğrudan $1 + 2 + \cdots + n$ toplamıdır. Sağ taraf açıldığında $\binom{n+1}{2} = \frac{n(n+1)}{2}$ elde edilir. Böylece Bölüm 2'de tümevarımla kanıtladığımız toplama Pascal üçgeni üzerinden de ulaşmış oluruz. □

**Uyarı:** $n = 0$ için alternatif toplam 1'dir; çünkü açılımda yalnızca $\binom{0}{0} = 1$ terimi bulunur. Bu nedenle toplamın sıfıra eşit olması için $n \geq 1$ koşulu şarttır.

## 6.4 Kombinatorik İspat

Bir eşitliği cebirsel işlemlerle doğrulamak mümkündür; ancak kombinasyon katsayıları içeren durumlarda çoğu kez daha aydınlatıcı bir yol vardır: Eşitliğin iki yanını da aynı nesnelerin sayısı olarak yorumlamak. Bu yaklaşıma **kombinatorik ispat** ya da Bölüm 10'da ayrıntısıyla ele alacağımız adıyla **çifte sayım** denir.

Yöntemin dayanağı basittir. Belirli bir nesne kümesi belirlenir. Bu kümedeki elemanlar birinci bir bakış açısıyla sayılıp $A$ sonucu bulunur; ardından aynı nesneler farklı bir gruplama veya sıra gözetilerek sayılıp $B$ sonucu elde edilir. Aynı kümenin eleman sayısı iki farklı değer alamayacağından $A = B$ olmak zorundadır. Burada dikkat edilmesi gereken tek husus, iki yöntemin de harfiyen *aynı* nesneleri saydığından emin olmaktır.

Bölüm 5'te başkanlı komite kurgusuyla ispatladığımız

$$k\binom{n}{k} = n\binom{n-1}{k-1}$$

özdeşliği bu yöntemin ilk örneğiydi. Oradaki $k$ çarpanı, seçilen grubun içinden bir başkan belirleme hamlesine karşılık geliyordu. Şimdi bu fikri, grup büyüklüğünün serbest bırakıldığı daha kapsamlı bir toplama taşıyalım.

**Teorem 6.18.** *Her $n \geq 1$ için*

$$\sum_{k=1}^{n} k\binom{n}{k} = n2^{n-1}.$$

*Kanıt.* $n$ kişilik bir topluluktan boş olmayan bir çalışma grubu ve bu grubun içinden bir lider seçeceğimizi varsayalım.

Grubun büyüklüğü $k$ olarak belirlenmişse grubu seçmenin $\binom{n}{k}$, grup içinden lideri seçmenin ise $k$ yolu vardır. $k = 1, 2, \ldots, n$ durumları birbirini dışladığından toplam liderli grup sayısı $\sum_{k=1}^{n} k\binom{n}{k}$ olur.

Aynı seçimi ters sırada yapalım: Önce tüm topluluktan lideri seçelim; bunun $n$ farklı yolu vardır. Lider belirlendikten sonra kalan $n-1$ kişinin her biri gruba dahil olabilir ya da olmayabilir. Bu kişilerin oluşturabileceği alt grupların sayısı $2^{n-1}$'dir. Dolayısıyla toplam $n2^{n-1}$ farklı liderli grup kurulabilir. Her iki sayım da aynı nesneleri listelediğinden eşitlik kanıtlanmış olur. □

**Teorem 6.19** (Vandermonde özdeşliği)**.** *$m, n, r \in \mathbb{N}$ ve $r \leq m + n$ için*

$$\sum_{k=0}^{r} \binom{m}{k}\binom{n}{r-k} = \binom{m+n}{r}.$$

*Geçersiz seçimlere karşılık gelen binom katsayıları 0 kabul edilir.*

*Kanıt.* Bir toplulukta $m$ kişilik birinci grup ve $n$ kişilik ikinci grup bulunsun. Toplam $m+n$ kişi arasından $r$ kişilik bir kurul seçeceğiz.

Tüm topluluktan doğrudan seçim yapıldığında kurul sayısı $\binom{m+n}{r}$ olur. İkinci bir yolla, kurulda birinci gruptan yer alacak kişi sayısına $k$ diyelim. Birinci gruptan $k$ kişi $\binom{m}{k}$ yolla, ikinci gruptan kalan $r-k$ kişi ise $\binom{n}{r-k}$ yolla seçilir. Bu çarpım, içinde birinci gruptan tam $k$ üye barındıran kurulların sayısını verir. $k$'nın olası tüm değerleri ayrık durumlar oluşturduğundan, toplama kuralı gereğince sol taraf elde edilir. Her iki yol da aynı $r$ kişilik kurulları saydığı için eşitlik geçerlidir. □

### 6.4.1 Çözümlü örnekler

**Örnek 6.20.**

$$\sum_{k=0}^{n} \binom{n}{k}^2 = \binom{2n}{n}$$

*özdeşliğini Vandermonde özdeşliğinden çıkarınız.*

*Çözüm.* Vandermonde özdeşliğinde $m=n$ ve $r=n$ değerlerini yerine koyalım:

$$\sum_{k=0}^{n} \binom{n}{k}\binom{n}{n-k} = \binom{2n}{n}.$$

Bölüm 5'teki simetri kuralından $\binom{n}{n-k} = \binom{n}{k}$ olduğunu biliyoruz. Böylece sol taraftaki her çarpım terimi bir kareye dönüşür ve istenen eşitlik doğrudan çıkar. Örneğin $n=3$ için sol taraf $1^2+3^2+3^2+1^2 = 20$, sağ taraf ise $\binom{6}{3} = 20$ değerini verir. □

**Örnek 6.21.** *Bir sınıfta* 5 *kız ve* 7 *erkek öğrenci vardır. Dört kişilik bir ekibi kız öğrenci sayısına göre sınıflandırarak toplam ekip sayısını bulunuz.*

*Çözüm.* Ekipteki kız sayısı $0, 1, 2, 3$ veya 4 olabilir. Her durum için kız ve erkek öğrencileri bağımsız olarak seçeriz:

$$\begin{aligned}\binom{5}{0}\binom{7}{4} + \binom{5}{1}\binom{7}{3} + \binom{5}{2}\binom{7}{2} \\ + \binom{5}{3}\binom{7}{1} + \binom{5}{4}\binom{7}{0}.\end{aligned}$$

Vandermonde özdeşliği gereği bu toplam

$$\binom{5+7}{4} = \binom{12}{4} = 495$$

değerine eşittir. Aynı sonuca 12 kişi arasından doğrudan 4 kişi seçerek de varılır. □

**Örnek 6.22.**

$$\binom{8}{1} + 2\binom{8}{2} + 3\binom{8}{3} + \cdots + 8\binom{8}{8}$$

*toplamını bulunuz.*

*Çözüm.* Verilen ifade $\sum_{k=1}^{8} k\binom{8}{k}$ yapısındadır. Liderli grup özdeşliğinde $n = 8$ alınırsa

$$\sum_{k=1}^{8} k\binom{8}{k} = 8 \cdot 2^7 = 1024$$

elde edilir. Uzun toplamı terim terim hesaplamak yerine, ifadenin liderli grupları saydığını görmek çözümü doğrudan verir. □

**Örnek 6.23.**

$$\sum_{k=0}^{n} \binom{n}{k} 2^k = 3^n$$

*eşitliğini hem binom teoremiyle hem de sayma yoluyla açıklayınız.*

*Çözüm.* Binom teoreminde $a = 1$ ve $b = 2$ seçildiğinde $(1+2)^n = \sum_{k=0}^{n} \binom{n}{k} 2^k$ olur; bu da toplamın $3^n$ olduğunu gösterir.

Sayma yorumu için $n$ nesnenin her birini üç renkten biriyle boyadığımızı varsayalım. Her nesne için 3 bağımsız seçenek bulunduğundan toplam boyama sayısı $3^n$'dir. Alternatif olarak, ilk rengin dışındaki iki renkten birini alan nesnelerin sayısına $k$ diyelim. Önce bu $k$ nesne $\binom{n}{k}$ yolla belirlenir; ardından her biri kalan iki renkten biriyle $2^k$ farklı biçimde boyanır. $k$ üzerinden toplandığında sol taraf elde edilir. İki yaklaşım da aynı boyamaları saydığından eşitlik sağlanır. □

**Son kontrol noktası:** Bir kombinatorik ispatı incelerken daima "iki yöntem de tam olarak aynı nesne kümesini mi sayıyor?" sorusu sorulmalıdır. İlk yolun komiteleri, ikinci yolun ise başkanlı komiteleri sayması gibi ufak bir odak kayması ispatı geçersiz kılar. Sayılan nesneyi ispatın en başında açıkça tarif etmek bu tür yanılgıları engeller.

Bu bölümde binom katsayılarının yalnızca cebirsel birer çarpan olmadığını; parantezlerden yapılan seçimleri, Pascal üçgenindeki geometrik konumları ve farklı sayma yollarını temsil ettiğini gördük. Bir sonraki bölümde (Bölüm 7), kombinasyon formüllerine başvurmadan, nesneleri kutulara dağıtmanın kaçınılmaz sonuçlarını ele alan güvercin yuvası ilkesini inceleyeceğiz.

# Bölüm 7

# Güvercin Yuvası İlkesi

Bazı problemlerde nesnelerin tam olarak nereye yerleştiğini bilmeyiz; hatta yerleşim tamamen rastgele görünebilir. Buna rağmen yalnızca sayıların büyüklüğünden hareketle, kaçınılmaz bir çakışma olduğunu kanıtlayabiliriz. Güvercin yuvası ilkesi bu fikrin en yalın hâlidir: Güvercin sayısı yuva sayısından fazlaysa, en az bir yuvada birden çok güvercin bulunmak zorundadır.

İlkenin gücü, gerçek güvercinlerle sınırlı olmamasıdır. Öğrencileri doğdukları aylara, sayıları belirli çiftlere, noktaları küçük bölgelere, nesneleri renklerine veya son basamaklarına göre sınıflandırdığımızda; öğrenciler, sayılar ve noktalar "güvercin", sınıflar ise "yuva" rolünü üstlenir. Asıl ustalık, problemi doğru sınıflara ayırmaktır. Burada önce temel çelişkiyi kuracak, ardından ilkenin niceliksel genellemesini ve klasik uygulamalarda uygun yuvaları belirleme yollarını inceleyeceğiz.

## 7.1 Temel İlke

Bir çekmecede kırmızı ve mavi olmak üzere iki renkte çorap bulunduğunu düşünelim. Oda tamamen karanlıksa ilk iki çorabın farklı renk olması mümkündür. Fakat üçüncü çorap çekildiğinde, iki renkten birine ait iki çorabın mutlaka elimizde olacağını biliriz. Bu sonuca çorapların hangi sırayla geldiğini inceleyerek değil, yalnızca iki renk sınıfına üç çorap dağıtıldığını fark ederek ulaştık.

**Tanım 7.1** (Güvercin ve yuva)**.** *Bir güvercin yuvası probleminde sınıflandırılan nesnelere* ***güvercin****, nesnelerin yerleştirildiği sınıflara ise* ***yuva*** *denir. Bu adlar yalnızca benzetmedir; güvercinler sayılar, kişiler veya noktalar; yuvalar da renkler, aylar, aralıklar ya da başka uygun sınıflar olabilir.*

**Teorem 7.2** (Güvercin yuvası ilkesi)**.** $n+1$ *güvercin* $n$ *yuvaya yerleştirilirse, en az bir yuvada en az iki güvercin bulunur.*

*Kanıt.* İddianın tersini varsayalım: Her yuvada en fazla bir güvercin bulunsun. $n$

yuva olduğuna göre yerleştirilebilecek güvercin sayısı en fazla

$$1 + 1 + \cdots + 1 = n$$

olabilir. Oysa elimizde $n + 1$ güvercin vardır. Bu, varsayımımızla çelişir. Demek ki en az bir yuvada iki ya da daha fazla güvercin bulunmak zorundadır. □

İspatın dikkat çekici yönü, hangi yuvanın dolacağını söylememesidir. İlke yalnızca varlık bildirir: "Böyle bir yuva vardır." Olimpiyat problemlerinde çoğu zaman aranan da tam olarak budur. Bir örnekte iki öğrencinin hangi ayda doğduğunu bulmamız gerekmez; en az iki öğrencinin aynı ayda doğduğunu kanıtlamamız yeterlidir.

### 7.1.1 Çözümlü örnekler

**Örnek 7.3.** *Bir odada* 13 *kişi varsa, en az iki kişinin aynı ayda doğduğunu gösteriniz.*

*Çözüm.* Güvercin olarak 13 kişiyi, yuva olarak yılın 12 ayını seçelim. Her kişi doğduğu aya tam olarak bir kez yerleştirilir. 13 kişi, 12 aya dağıtıldığı için güvercin yuvası ilkesi en az bir ayda en az iki kişinin doğmuş olmasını zorunlu kılar.

Burada kişilerin doğum günlerini veya yıllarını bilmemiz gerekmez. Yalnızca ay sayısının 12 ve kişi sayısının 13 olması yeterlidir. □

**Örnek 7.4.** *Bir sınıfta* 31 *öğrenci bulunmaktadır. En az üç öğrencinin aynı doğum ayında doğduğunu bu ilkeden yararlanarak kanıtlayınız.*

*Çözüm.* İki öğrencinin aynı ayda doğması için 13 öğrenci yeterliydi. Ancak burada "en az üç" isteniyor; temel ilkenin doğrudan biçimi bunu tek başına söylemez. Tersini düşünelim: Her ayda en fazla iki öğrenci doğmuş olsun. O zaman 12 ayda bulunabilecek öğrenci sayısı en fazla

$$12 \cdot 2 = 24$$

olurdu. Sınıfta 31 öğrenci bulunduğundan bu imkânsızdır. Dolayısıyla en az bir ayda en az üç öğrenci doğmuştur.

Bu düşünce, bir sonraki kısımda formülleştireceğimiz genelleştirilmiş ilkenin zeminini hazırlar. □

**Örnek 7.5.** 1 *ile* 20 *arasından seçilen* 11 *tam sayı arasında aynı tekliğe–çiftliğe sahip iki sayı bulunduğunu gösteriniz.*

*Çözüm.* Bu kez yuvalar yalnızca iki tanedir: tek sayılar ve çift sayılar. Seçilen her sayı tam olarak bu iki sınıftan birine girer. 11 sayı iki yuvaya dağıtıldığı için, özellikle en az bir yuvada iki sayı bulunur. Böylece seçilen sayılardan en az ikisi ya ikisi de tek ya da ikisi de çifttir.

Bu sonuç zayıf görünebilir; fakat doğru sınıflandırma seçildiğinde çok daha keskin sonuçlara dönüşebilir. Bölüm 17'de tekliğin–çiftliğin imkânsızlık ispatlarında nasıl etkili bir araç olduğunu ayrıntılı biçimde ele alacağız. □

**Örnek 7.6.** *Bir sepette yalnızca sarı, yeşil ve kırmızı bilyeler vardır. Renk dağılımı bilinmiyor olsa bile, sepetten çekilen* 4 *bilye arasında aynı renk iki bilye bulunduğunu ispatlayınız.*

*Çözüm.* Üç renk, üç yuvadır; çekilen dört bilye ise dört güvercindir. Dört güvercini üç yuvaya yerleştirdiğimizde en az bir yuvada iki güvercin bulunur. Bu da çekilen bilyelerden en az ikisinin aynı renkte olması demektir. □

**Sık yapılan hata:** İlkede "nesne sayısı yuva sayısına eşitse" zorunlu olarak çakışma olmaz. Örneğin 12 kişi, yılın 12 farklı ayında doğmuş olabilir. Kesinlik için güvercin sayısının yuva sayısından *fazla* olması gerekir.

## 7.2 Genelleştirilmiş İlke

Temel ilke, bir yuvada en az iki nesne bulunacağını söyler. Fakat nesne sayısı yuva sayısından çok daha büyükse daha kuvvetli bir sonuç bekleriz. Örneğin 100 kitabı 7 rafa yerleştirirsek, yalnızca bir rafta iki kitap bulunduğunu söylemek yetersizdir; aslında bazı raflarda en az 15 kitap bulunmak zorundadır.

Bu zorunlu sayıyı bulmak için toplam nesne sayısını yuva sayısına böleriz. Bölüm sonucu tam sayı değilse, kesri yukarı yuvarlamamız gerekir. Çünkü "en az" ifadesi bir tam sayı değerini belirtir.

**Teorem 7.7** (Genelleştirilmiş güvercin yuvası ilkesi)**.** *$N$ güvercin $k$ yuvaya yerleştirilirse, en az bir yuvada en az*

$$\left\lceil \frac{N}{k} \right\rceil$$

*güvercin bulunur.*

*Kanıt.*

$$t = \left\lceil \frac{N}{k} \right\rceil$$

olsun. Tersini varsayalım; yani her yuvada en fazla $t-1$ güvercin bulunsun. O zaman bütün yuvalardaki güvercinlerin sayısı en fazla $k(t-1)$ olur.

$t$, $N/k$ sayısından büyük veya ona eşit olan en küçük tam sayı olduğundan $t-1 < N/k$'dır. Her iki tarafı pozitif $k$ sayısıyla çarparsak

$$k(t-1) < N$$

elde ederiz. Bu, yuvalarda en fazla $k(t-1)$ güvercin varken toplamda $N$ güvercin bulunduğu varsayımıyla çelişir. Demek ki en az bir yuvada en az $t = \lceil N/k \rceil$ güvercin vardır. □

Bu teorem, en az kaç nesnenin bir arada bulunacağını verir. Bazen problem ters yönde sorulur: Her yuvada en fazla $r$ nesne olmasını istiyorsak, $k$ yuvaya en fazla $kr$ nesne yerleştirebiliriz. Bir nesne daha eklemek, yani $kr+1$ nesneye çıkmak, bir yuvada en az $r+1$ nesneyi zorunlu kılar.

### 7.2.1 Çözümlü örnekler

**Örnek 7.8.** 100 *kitabın* 7 *rafa yerleştirildiği bir kütüphanede, en az bir rafta en az kaç kitap bulunur?*

*Çözüm.* Kitaplar güvercin, raflar yuva olsun. Genelleştirilmiş ilke doğrudan

$$\left\lceil \frac{100}{7} \right\rceil = \left\lceil 14 + \frac{2}{7} \right\rceil = 15$$

sonucunu verir. Raflardan en az birinde en az 15 kitap bulunur.

14'ün yeterli olmadığını görmek önemlidir: Yedi rafa 14'er kitap koyarsak yalnızca 98 kitap yerleştiririz; kalan iki kitabın raflardan birine veya ikisine eklenmesi gerekir. □

**Örnek 7.9.** 73 *öğrenci altı çalışma grubuna dağıtılacaktır. En az bir grupta en az* 13 *öğrenci bulunacağını gösteriniz.*

*Çözüm.* Ortalama öğrenci sayısı $73/6 = 12 + 1/6$'dır. Öğrenci sayısı kesirli olamayacağı için ortalamanın üstündeki ilk tam sayı olan 13 zorunludur:

$$\left\lceil \frac{73}{6} \right\rceil = 13.$$

Tersinden de kontrol edebiliriz: Her grupta en çok 12 öğrenci olsaydı toplam öğrenci sayısı en fazla $6 \cdot 12 = 72$ olurdu. Bu, 73 öğrenciyi yerleştirmeye yetmez. □

**Örnek 7.10.** *Bir laboratuvarda* 51 *ölçüm, sıcaklık değerinin tam kısmına göre* 10 *sınıfa ayrılmıştır:* 0*'dan* 9*'a kadar. En az bir sınıfta en az kaç ölçüm bulunduğu kesindir?*

*Çözüm.* On sıcaklık sınıfı yuva, 51 ölçüm güvercindir. Bu nedenle en dolu sınıfta en az

$$\left\lceil \frac{51}{10} \right\rceil = 6$$

ölçüm vardır. Beşer ölçüm on sınıfta ancak 50 ölçüm eder; elli birinci ölçüm sınıflardan birini altıya çıkarır. □

**Örnek 7.11.** *Bir kutuda* 8 *farklı renk etiketi vardır. En az* 25 *nesne etiketlenirse, aynı etiketi taşıyan en az dört nesne bulunacağını kanıtlayınız.*

*Çözüm.* Nesneleri etiketlerinin rengine göre sekiz yuvaya ayıralım. Genelleştirilmiş ilke

$$\left\lceil \frac{25}{8} \right\rceil = 4$$

verir. Sezgisel bir kontrol de şudur: Her renkten en fazla üç nesne olsaydı toplam nesne sayısı en fazla $8 \cdot 3 = 24$ olurdu. Yirmi beşinci nesne, renklerden birinin dördüncü örneği olmak zorundadır. □

**Uyarı:** $\lceil N/k \rceil$ sonucu, "her yuvada bu kadar nesne vardır" anlamına gelmez. Sadece *en az bir* yuvanın bu sayıya ulaştığını söyler. 100 kitap örneğinde bazı raflar boş bile kalabilir.

### 7.2.2 Garanti sayısını tersine bulmak

Genelleştirilmiş ilke, "$N$ nesne dağıtıldıysa ne kadar kalabalık bir yuva vardır?" sorusunu yanıtlar. Aynı fikri tersine çevirmek, daha sık karşılaşılan bir soruyu çözer: Bir yuvada en az $r$ nesne bulunmasını *garanti etmek* için kaç nesne gerekir?

Her yuvada en fazla $r-1$ nesne kalacak en kötü senaryoyu ele alalım. $k$ yuvanın her birine $r-1$ nesne koyabiliriz; böylece toplam

$$k(r-1)$$

nesne yerleştirilir ve hiçbir yuvada $r$ nesne bulunmaz. Bir nesne daha eklediğimiz anda, yani $k(r-1)+1$ nesneye ulaştığımızda, en az bir yuva $r$ nesneye çıkmak zorundadır.

**Teorem 7.12** (Kesin garanti eşiği)**.** *$k$ yuvadan birinde en az $r$ güvercin bulunmasını garanti etmek için $k(r-1)+1$ güvercin yeterlidir; daha azı yeterli değildir.*

*Kanıt.* $k(r-1)+1$ güvercini $k$ yuvaya dağıtalım. Her yuvada en fazla $r-1$ güvercin bulunduğunu varsayarsak toplam en fazla $k(r-1)$ olurdu. Bu, elimizdeki güvercin sayısından bir eksiktir; dolayısıyla bir yuvada en az $r$ güvercin vardır.

Öte yandan $k(r-1)$ güvercin için her yuvaya tam $r-1$ güvercin koymak mümkündür. Bu düzende hiçbir yuva $r$ güvercine ulaşmaz. Demek ki eşikteki "+1" gereklidir. □

**Örnek 7.13.** *Yılın ayları bakımından en az dört kişinin aynı ayda doğmasını garanti etmek için bir odada en az kaç kişi bulunmalıdır?*

*Çözüm.* Yuvalar 12 ay, hedef ise aynı yuvadaki $r = 4$ kişidir. Kesin garanti eşiği

$$12(4-1)+1 = 12 \cdot 3 + 1 = 37$$

kişidir.

36 kişinin neden yetmediği de açıktır: Her ayda tam üç kişi doğmuş olabilir. Bu düzen 36 kişiyi içerir, fakat hiçbir ayda dört kişi bulunmaz. Otuz yedinci kişi ise mutlaka aylardan birinin dördüncü kişisi olur. □

**Örnek 7.14.** *Beş farklı renkte kalemin bulunduğu bir kutudan, aynı renkten en az altı kalem çekmeyi garanti etmek için en az kaç kalem çekilmelidir?*

*Çözüm.* Beş renk yuva, aynı renkten altı kalem hedef olduğundan $k = 5$, $r = 6$ alırız:

$$5(6-1)+1 = 26.$$

İlk 25 kalemde en kötü ihtimalle her renkten beşer tane çekilmiş olabilir. Yirmi altıncı kalem, renklerden birinin altıncısı olmak zorundadır. □

## 7.3 Uygulamalar

Güvercin yuvası ilkesi hazır bir formülü yerine koymaktan ibaret değildir. Çözümde ilk hamle şu olmalıdır: Nesneleri hangi ortak özelliğe göre sınıflandırırsak aradığımız iki nesnenin aynı sınıfa düşmesi anlamlı bir sonuç üretir? Bazen yuvalar renkler kadar belirgindir; bazen ise toplamı sabit yapan ikililer, bir sayının içindeki tek çarpan veya düzlemi bölen küçük kareler olarak gizlenir.

### 7.3.1 Sayıları doğru yuvalara ayırmak

**Örnek 7.15.** $1, 2, \ldots, 2n$ *sayıları arasından seçilen herhangi* $n+1$ *sayı içinde, toplamı* $2n+1$ *olan iki sayı bulunduğunu gösteriniz.*

*Çözüm.* Sayıları toplamı $2n+1$ olan çiftler hâlinde gruplandıralım:

$$\{1, 2n\},\ \{2, 2n-1\},\ \ldots,\ \{n, n+1\}.$$

Tam $n$ yuva vardır ve her yuvada iki sayı bulunur. Seçilen $n+1$ sayı güvercin olsun; her sayı ait olduğu çifte yerleştirilir. $n+1$ sayı $n$ yuvaya dağıtıldığı için seçilen iki sayı aynı yuvaya düşer. Aynı yuvadaki iki sayının toplamı tanım gereği $2n+1$'dir.

Buradaki püf noktası, yuvaları tek ve çift sayılar olarak ayırmamaktır. Böyle bir seçim yalnızca aynı tekliğe sahip iki sayı verirdi; istenen toplamı garanti etmezdi. □

**Örnek 7.16.** $1$ *ile* $2n$ *arasından seçilen herhangi* $n+1$ *sayı arasında, biri diğerini tam bölen iki sayı bulunduğunu gösteriniz.*

*Çözüm.* Her pozitif tam sayı, 2'nin kuvveti ile bir tek sayının çarpımı biçiminde tek bir şekilde yazılabilir. Örneğin

$$12 = 2^2 \cdot 3, \qquad 40 = 2^3 \cdot 5, \qquad 7 = 2^0 \cdot 7.$$

Her sayının bu yazılışındaki tek çarpanını yuva olarak belirleyelim. 1 ile $2n$ arasındaki olası tek çarpanlar

$$1, 3, 5, \ldots, 2n-1$$

olduğundan tam $n$ tanedir.

Seçilen $n+1$ sayı bu $n$ yuvaya dağıtılınca ikisinin tek çarpanı aynı olur. Bu iki sayıyı $2^p m$ ve $2^q m$ biçiminde yazalım; burada $m$ aynı tek sayıdır. Gerekirse adlandırmayı düzenleyip $p \leq q$ diyelim. O zaman

$$2^q m = 2^{q-p}(2^p m)$$

olduğundan $2^p m$ sayısı $2^q m$ sayısını tam böler. Böylece seçilen iki sayıdan biri diğerini böler. □

Bu örnekte tek çarpan yaklaşımı ilk bakışta yapay görünebilir. Ancak bir sayının içindeki bütün 2 çarpanlarını ayırdığımızda, aynı tek çarpanı taşıyan sayılar $m, 2m, 4m, \ldots$ biçiminde bir zincir oluşturur. Aynı zincirden seçilen iki sayının bölme ilişkisi taşıması kaçınılmazdır. Bölünebilmenin temel araçları Bölüm 11'de incelenecektir.

**Örnek 7.17.** *On bir tam sayı veriliyor. Bu sayıların ikisinin,* 10*'a bölündüğünde aynı birler basamağına sahip olduğunu gösteriniz.*

*Çözüm.* Birler basamağı için yalnızca on olasılık vardır: $0, 1, 2, \ldots, 9$. Bu on olasılığı yuva, verilen on bir sayıyı güvercin kabul edelim. Her tam sayı birler basamağına göre tam bir yuvaya düşer. On bir sayı on yuvaya dağıtıldığı için iki sayının birler basamağı aynıdır. □

### 7.3.2 Geometrik yuvalar

Geometrik problemlerde güvercinler çoğunlukla noktalardır. Yuvalar ise şeklin küçük parçalara bölünmesiyle elde edilir. İki noktanın aynı küçük parçaya düşmesi, aralarındaki uzaklığın o parçanın çapı kadar veya daha küçük olmasını sağlar.

**Örnek 7.18.** *Birim kenarlı bir karenin içine yerleştirilen* 5 *nokta arasında, uzaklığı en fazla* $\sqrt{2}/2$ *olan iki nokta bulunduğunu gösteriniz.*

*Çözüm.* Büyük kareyi kenar uzunluğu $1/2$ olan dört eş kareye ayıralım. Bu dört küçük kare yuvalar, beş nokta ise güvercinlerdir. Güvercin yuvası ilkesi gereğince iki nokta aynı küçük kareye düşer.

Aynı küçük karede bulunabilecek iki noktanın birbirinden en uzak olduğu durum, karşılıklı iki köşede yer aldıkları andır. Küçük karenin köşegeni Pisagor bağıntısıyla

$$\sqrt{\left(\frac{1}{2}\right)^2 + \left(\frac{1}{2}\right)^2} = \frac{\sqrt{2}}{2}$$

olur. Dolayısıyla aynı küçük karedeki iki noktanın uzaklığı en fazla $\sqrt{2}/2$'dir. □

**Örnek 7.19.** *Kenar uzunluğu* 2 *olan bir karenin içinde bulunan* 10 *noktadan ikisinin uzaklığının en fazla* $\sqrt{2}$ *olduğunu gösteriniz.*

*Çözüm.* Kareyi kenarı 1 olan dört eş kareye bölelim. On nokta, dört küçük kareye dağılır. Genelleştirilmiş ilke en az bir küçük karede en az $\lceil 10/4 \rceil = 3$ nokta olduğunu söyler; ancak çözüm için iki nokta yeterlidir.

Aynı birim kareye düşen iki nokta arasındaki uzaklık, bu küçük karenin köşegeninden fazla olamaz. Köşegen uzunluğu $\sqrt{1^2 + 1^2} = \sqrt{2}$ olduğundan istenen çift bulunur. □

**Örnek 7.20.** *Bir kenarı* 3 *olan karenin içine* 10 *nokta yerleştirilmiştir. Bu noktalardan ikisinin uzaklığının en fazla* $\sqrt{2}$ *olduğunu kanıtlayınız.*

*Çözüm.* Büyük kareyi dokuz adet birim kareye bölmek uygundur. Dokuz küçük kare yuva, on nokta güvercindir. Temel ilkeye göre iki nokta aynı birim karede yer alır. Birim karenin köşegeni $\sqrt{2}$ olduğundan bu iki noktanın uzaklığı en fazla $\sqrt{2}$ olur.

Burada kilit nokta, birim uzunluğun hangi bölmede işe yarayacağını önceden belirlemektir. Yuva sayısını nokta sayısından az tutarken, her yuvanın çapını soruda istenen uzaklığa denk getirdik. □

**Yuvaları seçme stratejisi:** Sayı sorularında istenen ilişkiyi sağlayan denklik sınıflarını kurgulayın; geometri sorularında ise şekli, küçük parçaların çapı hedeflenen uzaklığa uyacak biçimde bölün. İlkenin kendisi doğrudan uygulanır; asıl zihinsel emek doğru yuvaları belirleme aşamasındadır.

Bu bölümde güvercin yuvası ilkesinin, olası tüm durumları tek tek listelemeden nasıl zorunlu bir çakışma verdiğini ele aldık. Bölüm 8'de ise kümelerin kesişimlerindeki mükerrer elemanları systematically hesaba katan içerme-dışarma ilkesini inceleyeceğiz.

# Bölüm 8

# İçerme-Dışarma

Bölüm 3'te ele aldığımız temel sayma kurallarından ilki toplama kuralıydı: Eğer iki küme birbirinden bütünüyle ayrıksa, bu iki kümenin birleşiminin eleman sayısı, kümelerin eleman sayıları toplamına eşittir. Ancak uygulamada ve algoritmik problemlerde karşımıza çıkan kümeler nadiren bu denli birbirinden yalıtılmıştır. Kümeler sıklıkla kesişir, ortak elemanlar barındırır ve bu ortak elemanlar üst üste biner.

Birbirini kesen kümelerin birleşimini sayarken doğrudan toplama yapmak bizi hataya sürükler; çünkü ortak elemanları birden fazla kez saymış oluruz. Bu bölümde, üst üste binen kümelerin eleman sayısını hatasız bir şekilde belirlememizi sağlayan temel yöntemlerden birini, **içerme-dışarma** (inclusion–exclusion) ilkesini inşa edeceğiz.

## 8.1 İki ve Üç Küme

### 8.1.1 İki Küme ve Çift Sayımı Düzeltme

İlkenin mantığını somut bir soru üzerinden görelim.

**Örnek 8.1.** *1'den 100'e kadar olan tam sayılar arasından 2 veya 3 ile bölünebilen kaç sayı vardır?*

*Çözüm.* 2'ye bölünen tam sayıların kümesine $A$, 3'e bölünen tam sayıların kümesine de $B$ diyelim. Amacımız bu iki kümenin birleşimi olan $A \cup B$ kümesinin eleman sayısını, yani $|A \cup B|$ değerini bulmaktır.

1 ile 100 arasında 2'ye bölünen tam sayılar $2, 4, 6, \ldots, 100$ biçimindedir. Bu sayıların adedi:

$$|A| = \left\lfloor \frac{100}{2} \right\rfloor = 50$$

Benzer biçimde, 3'e bölünen tam sayılar $3, 6, 9, \ldots, 99$ biçimindedir ve adedi:

$$|B| = \left\lfloor \frac{100}{3} \right\rfloor = 33$$

Bu iki değeri doğrudan toplarsak $50 + 33 = 83$ buluruz; ancak bu toplam doğru sonucu vermez.

Küçük bir sayıyı, örneğin 6'yı ele alalım. 6 sayısı çift olduğu için $A$ kümesinin içindedir; dolayısıyla 50'nin içinde bir kez sayılmıştır. Öte yandan 6 sayısı 3'ün de katı olduğundan $B$ kümesinin içindedir ve 33'ün içinde de bir kez sayılmıştır. Demek ki 6 sayısı bu 83 toplamına bir değil, iki kez dahil edilmiştir. Yalnızca 6 değil; $12, 18, 24, \ldots$ gibi hem 2'ye hem de 3'e bölünen tüm sayılar ikişer kez sayılmıştır.

2 ve 3 aralarında asal olduğundan (Bölüm 11 ve 12'deki bölünebilme ilkeleri uyarınca), bir sayının hem 2'ye hem de 3'e bölünebilmesi o sayının 6'ya tam bölünmesi demektir. Bu ortak sayıların kümesi tam olarak $A \cap B$ kesişimidir. 1 ile 100 arasında 6'ya bölünen tam sayıların adedi:

$$|A \cap B| = \left\lfloor \frac{100}{6} \right\rfloor = 16$$

Her bir ortak elemanı fazladan birer kez saydığımıza göre, bu fazlalığı gidermenin yolu kesişimdeki eleman sayısını toplamdan bir kez çıkarmaktır. O hâlde aranan yanıt:

$$|A \cup B| = |A| + |B| - |A \cap B| = 50 + 33 - 16 = 67$$

olarak elde edilir. □

Bu basit örnek genel kuralın özünü gösterir: Bir elemanı iki kez saydıysak, net sayımını 1'e indirmek için onu bir kez dışarı çıkarmalıyız (dışarma).

**Teorem 8.2** (İki Küme İçin İçerme-Dışarma İlkesi)**.** *$A$ ve $B$ sonlu iki küme olmak üzere,*

$$|A \cup B| = |A| + |B| - |A \cap B|$$

*eşitliği geçerlidir.*

*Kanıt.* $A \cup B$ kümesindeki herhangi bir $x$ elemanını ele alalım. Bu eleman için tam olarak üç durum söz konusudur:

1. $x \in A$ fakat $x \notin B$ (yalnızca $A$'da olanlar): Bu durumda $x$ elemanı $|A|$ içinde 1 kez, $|B|$ içinde 0 kez, $|A \cap B|$ içinde 0 kez sayılır. Eşitliğin sağ tarafındaki net katkısı $1 + 0 - 0 = 1$'dir.

2. $x \in B$ fakat $x \notin A$ (yalnızca $B$'de olanlar): Bu durumda $x$ elemanı $|A|$ içinde 0 kez, $|B|$ içinde 1 kez, $|A \cap B|$ içinde 0 kez sayılır. Eşitliğin sağ tarafındaki net katkısı $0 + 1 - 0 = 1$'dir.

3. $x \in A$ ve $x \in B$ (ortak olanlar): Bu durumda $x$ elemanı $|A|$ içinde 1 kez, $|B|$ içinde 1 kez ve $|A \cap B|$ içinde 1 kez sayılır. Eşitliğin sağ tarafındaki net katkısı $1 + 1 - 1 = 1$'dir.

Her durumda birleşime ait olan herhangi bir eleman sağ taraftaki formülde net olarak tam 1 kez sayılmaktadır. Dolayısıyla eşitlik her zaman doğrudur. □

### 8.1.2 Üç Kümeye Geçiş: Dengeyi Kurmak

Şimdi üçüncü bir koşul ekleyerek adımları genişletelim.

**Örnek 8.3.** *1'den 100'e kadar olan tam sayılar arasından 2, 3 veya 5 ile bölünebilen kaç sayı vardır?*

*Çözüm.* 2'ye bölünenlerin kümesi $A$, 3'e bölünenlerin kümesi $B$ ve 5'e bölünenlerin kümesi $C$ olsun. Amacımız $|A \cup B \cup C|$ değerini bulmaktır.

Önce her kümenin büyüklüğünü tek tek hesaplayalım:

$$|A| = \left\lfloor \frac{100}{2} \right\rfloor = 50$$
$$|B| = \left\lfloor \frac{100}{3} \right\rfloor = 33$$
$$|C| = \left\lfloor \frac{100}{5} \right\rfloor = 20$$

Tüm bu elemanları doğrudan toplarsak:

$$|A| + |B| + |C| = 50 + 33 + 20 = 103$$

elde ederiz. 1'den 100'e kadar zaten 100 sayı varken sonucun 103 çıkması, kimi elemanların birden fazla kez sayıldığını gösterir.

İkişerli ortak elemanları ele alalım:

- $2 \times 3 = 6$ ile bölünenler hem $A$'da hem $B$'dedir: $|A \cap B| = \lfloor 100/6 \rfloor = 16$.
- $2 \times 5 = 10$ ile bölünenler hem $A$'da hem $C$'dedir: $|A \cap C| = \lfloor 100/10 \rfloor = 10$.
- $3 \times 5 = 15$ ile bölünenler hem $B$'de hem $C$'dedir: $|B \cap C| = \lfloor 100/15 \rfloor = 6$.

İki küme probleminde yaptığımız gibi, ikişerli kesişimleri dışarı atalım:

$$103 - (16 + 10 + 6) = 103 - 32 = 71$$

Peki işlem bu aşamada tamamlandı mı?

30 sayısına bakalım. 30, $2 \times 3 \times 5 = 30$ olduğundan hem 2'ye, hem 3'e, hem de 5'e tam bölünür; yani $30 \in A \cap B \cap C$ kümesindedir. 30'un yukarıdaki işlemlerde kaç kez hesaba katıldığını kontrol edelim:

- $|A|$, $|B|$ ve $|C|$ toplamında 3 kez eklendi (+3).
- $|A \cap B|$, $|A \cap C|$ ve $|B \cap C|$ çıkarılırken 3 kez çıkarıldı (-3).

Net durum: $+3 - 3 = 0$. Yani 30 sayısı (ve onun gibi 2, 3, 5'in ortak katları olan 60 ve 90), elde ettiğimiz 71 sonucunun içinde **hiç sayılmamıştır**. Birleşime ait bir elemanı bütünüyle dışarıda bırakmış olduk.

Bu eksikliği gidermek ve bu elemanları doğru biçimde saymak için, üç kümenin kesişimini tekrar **içeri almalı**, yani eklemeliyiz:

$$|A \cap B \cap C| = \left\lfloor \frac{100}{30} \right\rfloor = 3$$

Böylece doğru sayım:

$$|A \cup B \cup C| = 71 + 3 = 74$$

olarak bulunur. □

Bu döngü içerme-dışarmanın temel mekanizmasıdır: Teklileri ekle (içerme), çiftlileri çıkar (dışarma), üçlüleri geri ekle (içerme).

**Teorem 8.4** (Üç Küme İçin İçerme-Dışarma İlkesi)**.** *$A$, $B$ ve $C$ sonlu üç küme olmak üzere,*

$$|A \cup B \cup C| = |A| + |B| + |C| - (|A \cap B| + |A \cap C| + |B \cap C|) + |A \cap B \cap C|$$

*eşitliği geçerlidir.*

*Kanıt.* $A \cup B \cup C$ kümesindeki herhangi bir $x$ elemanının formülün sağ tarafına yaptığı net katkıyı, $x$'in tam olarak kaç kümenin elemanı olduğuna göre inceleyelim:

1. $x$ bu üç kümeden **tam birine** ait olsun (örneğin sadece $A$'da): Tekli toplamlardan $|A|$'dan 1 katkı gelir. İkili veya üçlü kesişimlerde yer almadığından diğer terimlerden 0 gelir. Net katkı: 1.

2. $x$ bu üç kümeden **tam ikisine** ait olsun (örneğin $A$ ve $B$'de, ama $C$'de değil): Teklilerden $|A|$ ve $|B|$ olmak üzere 2 katkı gelir. İkililerden yalnızca $|A \cap B|$ içinde yer aldığından -1 katkı gelir. Üçlü kesişimde yer almaz (0). Net katkı: $2 - 1 = 1$.

3. $x$ bu üç kümenin **üçüne de** ait olsun ($A \cap B \cap C$): Tekli terimlerin üçünde de yer alır (+3). İkili kesişimlerin üçünde de yer alır (-3). Üçlü kesişimde yer alır (+1). Net katkı: $3 - 3 + 1 = 1$.

Hangi durumda olursa olsun, birleşime ait her eleman formül tarafından tam 1 kez sayılmaktadır. Dolayısıyla eşitlik doğrudur. □

### 8.1.3 Çözümlü Örnekler

**Örnek 8.5.** *40 kişilik bir sınıfta 22 öğrenci matematik, 18 öğrenci fizik, 15 öğrenci kimya dersini almaktadır. 9 öğrenci hem matematik hem fizik, 7 öğrenci hem matematik hem kimya, 5 öğrenci ise hem fizik hem kimya dersini almaktadır. Her üç dersi de alan 3 öğrenci olduğuna göre, bu üç dersten* ***en az birini*** *alan kaç öğrenci vardır? Hiçbir dersi almayan kaç öğrenci vardır?*

*Çözüm.* Matematik alanların kümesi $M$, fizik alanların kümesi $F$, kimya alanların kümesi $K$ olsun. Verilen büyüklükler şunlardır:

$$\begin{aligned} |M| = 22, \quad |F| = 18, \quad |K| = 15 \\ |M \cap F| = 9, \quad |M \cap K| = 7, \quad |F \cap K| = 5 \\ |M \cap F \cap K| = 3 \end{aligned}$$

En az bir dersi alan öğrenciler $M \cup F \cup K$ kümesini oluşturur. Üç küme içermedışarma formülünü uygularız:

$$\begin{aligned} |M \cup F \cup K| &= (|M| + |F| + |K|) \\ &\quad - (|M \cap F| + |M \cap K| + |F \cap K|) + |M \cap F \cap K| \\ &= (22 + 18 + 15) - (9 + 7 + 5) + 3 \\ &= 55 - 21 + 3 = 37 \end{aligned}$$

Demek ki öğrencilerin 37'si en az bir ders almaktadır.

Sınıf mevcudu 40 olduğuna göre, hiçbir dersi almayan öğrenci sayısı evrensel kümeden birleşimin çıkarılmasıyla bulunur:

$$40 - |M \cup F \cup K| = 40 - 37 = 3$$

olarak elde edilir. □

**Örnek 8.6.** *1'den 600'e kadar olan tam sayılar arasından 4 veya 6 ile bölünen fakat 5 ile* ***bölünmeyen*** *kaç sayı vardır?*

*Çözüm.* İstenen koşul iki parçadan oluşur: Sayı 4 veya 6'ya bölünmeli, fakat 5'e bölünmemelidir. Bu tip sorularda 4 veya 6'ya bölünenleri bulup içlerinden 5'e bölünenleri çıkarmak en güvenli yoldur.

4 ile bölünenler $A$, 6 ile bölünenler $B$, 5 ile bölünenler $C$ olsun. Bizden istenen küme $(A \cup B) \setminus C$ kümesidir. Kümeler teorisinden:

$$|(A \cup B) \setminus C| = |A \cup B| - |(A \cup B) \cap C|$$

yazabiliriz. Dağılma özelliğinden $(A \cup B) \cap C = (A \cap C) \cup (B \cap C)$ olur.

Adım adım hesaplayalım:

1. Önce $A \cup B$ kümesini bulalım. Hem 4'e hem 6'ya bölünen en küçük pozitif tam sayı (Bölüm 13'te ele aldığımız EKOK tanımıyla) $\text{EKOK}(4,6) = 12$'dir.

$$|A| = \left\lfloor \frac{600}{4} \right\rfloor = 150, \quad |B| = \left\lfloor \frac{600}{6} \right\rfloor = 100, \quad |A \cap B| = \left\lfloor \frac{600}{12} \right\rfloor = 50$$

   Buradan:

$$|A \cup B| = 150 + 100 - 50 = 200$$

2. Şimdi bu sayıların arasından 5'e de bölünenleri, yani $(A \cap C) \cup (B \cap C)$ kümesini bulalım:

   - $A \cap C$: Hem 4'e hem 5'e bölünenler ($\text{EKOK}(4,5) = 20$'ye bölünenler):

$$|A \cap C| = \left\lfloor \frac{600}{20} \right\rfloor = 30$$

   - $B \cap C$: Hem 6'ya hem 5'e bölünenler ($\text{EKOK}(6,5) = 30$'a bölünenler):

$$|B \cap C| = \left\lfloor \frac{600}{30} \right\rfloor = 20$$

   - $(A \cap C) \cap (B \cap C) = A \cap B \cap C$: Hem 4, hem 6, hem 5'e bölünenler. $\text{EKOK}(4,6,5) = 60$'a bölünür:

$$|A \cap B \cap C| = \left\lfloor \frac{600}{60} \right\rfloor = 10$$

   İki küme formülünü $(A \cap C)$ ve $(B \cap C)$ için uygularsak:

$$|(A \cap C) \cup (B \cap C)| = 30 + 20 - 10 = 40$$

3. Son olarak istenen farkı alırız:

$$|(A \cup B) \setminus C| = 200 - 40 = 160$$

Demek ki 1'den 600'e kadar aradığımız özellikte 160 sayı vardır. □

**Örnek 8.7.** *6 basamaklı, rakamları toplamı 4 olan kaç farklı doğal sayı yazılabilir?*

*Çözüm.* Bir doğal sayının 6 basamaklı olması için ilk basamağının sıfırdan farklı olması gerekir. Sayımız $a_1a_2a_3a_4a_5a_6$ olsun. Buradaki kısıtlar şunlardır:

$$a_1 + a_2 + a_3 + a_4 + a_5 + a_6 = 4$$

ve $a_1 \geq 1$, diğer $a_i \geq 0$ olmalıdır.

İlk basamağın sıfır olmaması kuralını uygulamak için $a_1' = a_1 - 1$ tanımlayalım. Bu durumda $a_1' \geq 0$ olur ve denklemimiz:

$$a_1' + a_2 + a_3 + a_4 + a_5 + a_6 = 4 - 1 = 3$$

hâline gelir. Burada her bir değişken negatif olmayan birer tam sayıdır ($a_i \geq 0$). Normalde her bir basamak en fazla 9 değerini alabilir; ancak toplamları zaten 3 olduğu için hiçbir değişkenin 9'u aşma tehlikesi yoktur. Dolayısıyla basamakların üst sınır kısıtı kendiliğinden sağlanır.

Bölüm 5'te gördüğümüz ayraç (stars and bars) yöntemine göre, $n = 3$ özdeş nesneyi $k = 6$ farklı kutuya dağıtma sayısı:

$$\binom{n+k-1}{k-1} = \binom{3+6-1}{6-1} = \binom{8}{5} = \binom{8}{3} = \frac{8 \times 7 \times 6}{3 \times 2 \times 1} = 56$$

olarak bulunur.

Üst sınır kısıtlarının toplamı aşmadığı durumlarda içerme-dışarmaya gerek kalmadan doğrudan çözüme ulaşılır; ancak toplamın 9'u aştığı durumlarda içerme-dışarma zorunlu hâle gelir. □

## 8.2 Genel Formül

### 8.2.1 Daha Fazla Küme ve Değişen İşaretler Deseni

İki kümede tek bir çıkarma yaptık: $+-$. Üç kümede bir çıkarma, bir toplama yaptık: $+-+$. Dört, beş veya genel olarak $n$ küme olduğunda da benzer bir yol izleriz.

Her adımda birleşim alanını dengelemek için bir sonraki seviyedeki tüm kesişimleri hesaba katmamız gerekir. Bu dalgalı işaret deseni (önce ekle, sonra çıkar, tekrar ekle...) kombinatorik saymanın temel yapılarından biridir.

**Teorem 8.8** (Genel İçerme-Dışarma İlkesi)**.** *$A_1, A_2, \ldots, A_n$ sonlu kümeler olsun. Bu kümelerin birleşiminin eleman sayısı:*

$$\begin{aligned}|A_1 \cup A_2 \cup \cdots \cup A_n| = &\sum_{1 \leq i \leq n} |A_i| - \sum_{1 \leq i < j \leq n} |A_i \cap A_j| \\ &+ \sum_{1 \leq i < j < k \leq n} |A_i \cap A_j \cap A_k| - \cdots + (-1)^{n+1} |A_1 \cap A_2 \cap \cdots \cap A_n|\end{aligned}$$

*toplamı ile verilir. Toplam sembolüyle ifade edecek olursak:*

$$\left| \bigcup_{i=1}^{n} A_i \right| = \sum_{k=1}^{n} (-1)^{k+1} \sum_{1 \leq i_1 < i_2 < \cdots < i_k \leq n} |A_{i_1} \cap A_{i_2} \cap \cdots \cap A_{i_k}|$$

*olur.*

### 8.2.2 Bu Formül Neden Her Zaman Doğrudur?

Formülün geçerliliğini görmek için birleşime ait **tek bir elemanın** adımlardaki durumunu izlemek yeterlidir.

Birleşim kümesinden herhangi bir $x \in \bigcup_{i=1}^{n} A_i$ elemanı alalım. Bu eleman, verilen $n$ kümeden **tam** $m$ **tanesinin** içinde bulunsun ($1 \leq m \leq n$).

Formülün sağ tarafındaki terimlerin $x$ elemanını kaç kez saydığını inceleyelim:

1. **Tekli kümeler ($\sum |A_i|$):** $x$ tam $m$ kümenin içinde olduğuna göre, bu toplama tam $m = \binom{m}{1}$ kez katkı sağlar ($+\binom{m}{1}$).

2. **İkili kesişimler ($\sum |A_i \cap A_j|$):** $x$'in bir $A_i \cap A_j$ kesişiminde yer alması için, hem $A_i$'nin hem de $A_j$'nin $x$'i içeren o $m$ küme arasından seçilmiş olması gerekir. $m$ küme arasından 2 küme $\binom{m}{2}$ farklı şekilde seçilebilir. Dolayısıyla bu terimden $-\binom{m}{2}$ katkı gelir.

3. **Üçlü kesişimler ($\sum |A_i \cap A_j \cap A_k|$):** Benzer şekilde, $x$ elemanı bu $m$ küme arasından seçilen herhangi üçlünün kesişimindedir. Bu terimden $+\binom{m}{3}$ katkı gelir.

4. Genel olarak, $k$'lı kesişimlerin toplamından $x$'e gelen katkı $(-1)^{k+1}\binom{m}{k}$ olacaktır. Eğer $k > m$ ise, $x$ en fazla $m$ kümede yer aldığından daha büyük kesişimlerde bulunamaz; bu terimlerden gelen katkı 0'dır.

O hâlde formülün sağ tarafının $x$ elemanına verdiği **net sayım değeri**:

$$\text{Net Sayım} = \binom{m}{1} - \binom{m}{2} + \binom{m}{3} - \binom{m}{4} + \cdots + (-1)^{m+1}\binom{m}{m}$$

olur. Bu toplamın değerini bulmak için Bölüm 6'da gördüğümüz binom açılımını hatırlayalım:

$$(1-1)^m = \binom{m}{0} - \binom{m}{1} + \binom{m}{2} - \binom{m}{3} + \cdots + (-1)^m\binom{m}{m}$$

Sol taraf $(0)^m = 0$ olduğundan:

$$0 = 1 - \left[\binom{m}{1} - \binom{m}{2} + \binom{m}{3} - \cdots + (-1)^{m+1}\binom{m}{m}\right]$$

Köşeli parantezin içindeki ifade tam olarak aradığımız net sayım değeridir. Buradan:

$$\text{Net Sayım} = 1$$

elde edilir. Demek ki bir eleman kaç kümenin kesişiminde olursa olsun ($m = 1, 2, \ldots, n$), formüldeki ekleme ve çıkarmalar birbirini götürür ve o eleman sonuca **tam 1 kez** eklenmiş olur.

Eleman hiçbir kümede yer almıyorsa ($m = 0$), zaten hiçbir terimde sayılmaz, katkısı 0'dır. İspat tamamlanmıştır.

### 8.2.3 Tümleyen Sayımı (Complementary Counting)

Çoğu problemde "en az bir koşulu sağlayanlar" yerine, "verilen koşulların **hiçbirini** sağlamayanlar" sorulur.

Evrensel kümemiz $U$ olsun. $A_1, A_2, \ldots, A_n$ kümeleri de istenmeyen durumları temsil etsin. Bu durumda hiçbir istenmeyen duruma düşmeyen elemanların kümesi:

$$|\overline{A_1} \cap \overline{A_2} \cap \cdots \cap \overline{A_n}| = |U| - |A_1 \cup A_2 \cup \cdots \cup A_n|$$

olarak ifade edilir. İçerme-dışarma formülünü buraya uygularsak:

$$|U \setminus (A_1 \cup \cdots \cup A_n)| = |U| - \sum |A_i| + \sum |A_i \cap A_j| - \sum |A_i \cap A_j \cap A_k| + \cdots$$

eşitliğini elde ederiz. Bu yaklaşım, özellikle doğrudan sayımın zor olduğu durumlarda işlemi belirgin biçimde kolaylaştırır.

**Tanım 8.9** (Yaygın Hata Uyarısı)**.** *İçerme-dışarma uygularken en sık yapılan hata terimlerin işaretlerini şaşırmaktır:*

- ***Birleşim arıyorsak:*** *Tekliler +, ikililer -, üçlüler +, dörtlüler - şeklinde ilerler.*
- ***Tümleyen (hiçbirini sağlamayanlar) arıyorsak:*** *Evrensel küme $|U|$ ile başlarız (+), teklileri çıkarırız (-), ikilileri ekleriz (+), üçlüleri çıkarırız (-).*

*Son terimin işaretini ezberlemek yerine, tekli adımlarda işaretin pozitif mi negatif mi olduğunu takip ederek ilerleyiniz.*

### 8.2.4 Çözümlü Örnekler

**Örnek 8.10.** *1'den 1000'e kadar olan tam sayılar arasından 2, 3, 5 veya 7 sayılarından **hiçbirine bölünmeyen** kaç sayı vardır?*

*Çözüm.* Evrensel kümemiz $U = \{1, 2, \ldots, 1000\}$ ve $|U| = 1000$'dir. İstenmeyen durumlar için kümeler tanımlayalım:

- $A_1$: 2 ile bölünenler
- $A_2$: 3 ile bölünenler
- $A_3$: 5 ile bölünenler
- $A_4$: 7 ile bölünenler

İstenen değer $|\overline{A_1} \cap \overline{A_2} \cap \overline{A_3} \cap \overline{A_4}|$'tür. Tümleyen formülüne göre:

$$\text{Sonuç} = 1000 - S_1 + S_2 - S_3 + S_4$$

Burada $S_k$, $k$'lı kesişimlerin eleman sayıları toplamıdır.

**1. Adım: Tekli Kümeler ($S_1$)**

$$|A_1| = \lfloor 1000/2 \rfloor = 500$$
$$|A_2| = \lfloor 1000/3 \rfloor = 333$$
$$|A_3| = \lfloor 1000/5 \rfloor = 200$$
$$|A_4| = \lfloor 1000/7 \rfloor = 142$$
$$S_1 = 500 + 333 + 200 + 142 = 1175$$

**2. Adım: İkili Kesişimler ($S_2$)** İkişerli çarpımlara bölünenleri sayarız ($\binom{4}{2} = 6$ terim):

- $|A_1 \cap A_2| = \lfloor 1000/6 \rfloor = 166$
- $|A_1 \cap A_3| = \lfloor 1000/10 \rfloor = 100$
- $|A_1 \cap A_4| = \lfloor 1000/14 \rfloor = 71$
- $|A_2 \cap A_3| = \lfloor 1000/15 \rfloor = 66$
- $|A_2 \cap A_4| = \lfloor 1000/21 \rfloor = 47$
- $|A_3 \cap A_4| = \lfloor 1000/35 \rfloor = 28$

$$S_2 = 166 + 100 + 71 + 66 + 47 + 28 = 478$$

**3. Adım: Üçlü Kesişimler ($S_3$)** Üçerli çarpımlara bölünenleri sayarız ($\binom{4}{3} = 4$ terim):

- $|A_1 \cap A_2 \cap A_3| = \lfloor 1000/(2 \times 3 \times 5) \rfloor = \lfloor 1000/30 \rfloor = 33$
- $|A_1 \cap A_2 \cap A_4| = \lfloor 1000/(2 \times 3 \times 7) \rfloor = \lfloor 1000/42 \rfloor = 23$
- $|A_1 \cap A_3 \cap A_4| = \lfloor 1000/(2 \times 5 \times 7) \rfloor = \lfloor 1000/70 \rfloor = 14$
- $|A_2 \cap A_3 \cap A_4| = \lfloor 1000/(3 \times 5 \times 7) \rfloor = \lfloor 1000/105 \rfloor = 9$

$$S_3 = 33 + 23 + 14 + 9 = 79$$

**4. Adım: Dörtlü Kesişim ($S_4$)** Dördünün çarpımına ($2 \times 3 \times 5 \times 7 = 210$) bölünenler:

$$S_4 = |A_1 \cap A_2 \cap A_3 \cap A_4| = \lfloor 1000/210 \rfloor = 4$$

**5. Adım: Birleştirme**

$$\text{Sonuç} = 1000 - 1175 + 478 - 79 + 4 = 228$$

Koşulları sağlayan 228 sayı bulunmaktadır. □

**Örnek 8.11.** $x_1+x_2+x_3+x_4 = 15$ *denkleminin, her* $i \in \{1,2,3,4\}$ *için* $0 \le x_i \le 6$ *koşulunu sağlayan kaç farklı tam sayı çözümü vardır?*

*Çözüm.* Üst sınır kısıtı ($x_i \le 6$) olmasaydı, problem Bölüm 5'te ele aldığımız klasik negatif olmayan tam sayılarda ayraç yöntemiyle çözülebilirdi. Koşulu bozan durumlar, herhangi bir değişkenin $x_i \ge 7$ olmasıdır. Bu istenmeyen durumları içerme-dışarma ile ayıklayabiliriz.

Evrensel küme $U$, hiçbir üst sınır olmaksızın $x_1+x_2+x_3+x_4 = 15$ denkleminin $x_i \ge 0$ çözümlerinin sayısıdır:

$$|U| = \binom{15+4-1}{4-1} = \binom{18}{3} = \frac{18 \times 17 \times 16}{6} = 816$$

İstenmeyen durum kümeleri olarak $A_i$, $x_i \ge 7$ olan çözümler kümesini göstersin ($i = 1,2,3,4$). İstenen çözüm sayısı:

$$\text{Çözüm Sayısı} = |U| - S_1 + S_2 - S_3 + S_4$$

**1. Bir değişkenin kısıtı aşması ($S_1$):** Diyelim ki $x_1 \ge 7$. Değişken değişimi yapalım: $x_1' = x_1 - 7 \ge 0$. Denklem şuna dönüşür:

$$x_1' + x_2 + x_3 + x_4 = 15 - 7 = 8$$

Bu denklemin negatif olmayan tam sayılarda çözüm sayısı:

$$\binom{8+4-1}{4-1} = \binom{11}{3} = \frac{11 \times 10 \times 9}{6} = 165$$

Kısıtı aşabilecek $\binom{4}{1} = 4$ farklı değişken olduğundan:

$$S_1 = 4 \times 165 = 660$$

**2. İki değişkenin kısıtı aşması ($S_2$):** Hem $x_1 \ge 7$ hem de $x_2 \ge 7$ olsun. Değişken değişimiyle $x_1' = x_1 - 7$ ve $x_2' = x_2 - 7$ dersek:

$$x_1' + x_2' + x_3 + x_4 = 15 - 7 - 7 = 1$$

Çözüm sayısı:

$$\binom{1+4-1}{4-1} = \binom{4}{3} = 4$$

İki değişken $\binom{4}{2} = 6$ farklı şekilde seçilebildiğinden:

$$S_2 = 6 \times 4 = 24$$

**3. Üç veya daha fazla değişkenin kısıtı aşması ($S_3, S_4$):** Üç değişken birden en az 7 olursa, toplamları en az $7 \times 3 = 21$ olur. Ancak tüm değişkenlerin toplamı 15 verildiğinden bu durum olanaksızdır. Dolayısıyla $S_3 = 0$ ve $S_4 = 0$'dır.

**Sonuç:**

$$\text{Çözüm Sayısı} = |U| - S_1 + S_2 = 816 - 660 + 24 = 180$$

olarak hesaplanır. □

## 8.3 Düzensiz Dizilimler (Derangements) ve Şapka Problemi

### 8.3.1 Şapka Problemi: Sabit Noktasız Permütasyonlar

Bölüm 4'te incelediğimiz permütasyon kavramının klasik uygulamalarından biri vestiyer ya da şapka problemidir:

> $n$ kişi bir restorana gider ve şapkalarını vestiyere teslim eder. Çıkışta vestiyer görevlisi şapkaları rastgele bir biçimde bu $n$ kişiye dağıtır. **Hiç kimsenin kendi şapkasını almama** olasılığı veya bunu sağlayan durum sayısı kaçtır?

Matematiksel olarak bu soru, $\{1, 2, \ldots, n\}$ kümesinin hiçbir $i$ elemanı için $\pi(i) = i$ koşulunu sağlamayan $\pi$ permütasyonlarını bulmaya karşılık gelir. Bir permütasyonda $\pi(i) = i$ eşitliğini sağlayan elemanlara **sabit nokta** (fixed point) denir. Biz sabit noktası bulunmayan permütasyonların sayısını aramaktayız.

**Tanım 8.12** (Düzensiz Dizilim / Derangement)**.** *$\{1, 2, \ldots, n\}$ kümesinin hiçbir elemanını kendi orijinal yerinde bırakmayan permütasyonlarına **düzensiz dizilim** (derangement) denir. $n$ elemanlı bir kümenin düzensiz dizilimlerinin sayısı $!n$ veya $D_n$ simgesiyle gösterilir.*

### 8.3.2 Küçük Değerleri Elle İnceleyelim

Genel formüle geçmeden önce küçük $n$ değerleri için olası permütasyonları inceleyelim:

- $\mathbf{n = 1}$: Tek bir eleman vardır: (1). Kendi yerindedir. Sabit noktası olmayan dizilim yoktur:
$$!1 = 0$$
- $\mathbf{n = 2}$: Olası dizilimler: $(1, 2)$ ve $(2, 1)$.
  - $(1, 2) \to$ her ikisi de kendi yerinde.
  - $(2, 1) \to$ 1, 2'nin yerinde; 2, 1'in yerinde. İkisi de yerinde değil.

$$!2 = 1$$

- $\mathbf{n = 3}$: Toplam $3! = 6$ permütasyon vardır:

$$\begin{aligned}
&(1, 2, 3) \to 3 \text{ sabit nokta} \\
&(1, 3, 2) \to 1 \text{ sabit nokta } (1) \\
&(3, 2, 1) \to 1 \text{ sabit nokta } (2) \\
&(2, 1, 3) \to 1 \text{ sabit nokta } (3) \\
&(2, 3, 1) \to \text{Sabit nokta yok} \quad \checkmark \\
&(3, 1, 2) \to \text{Sabit nokta yok} \quad \checkmark
\end{aligned}$$

6 permütasyondan yalnızca 2 tanesinde hiçbir eleman kendi yerinde değildir:

$$!3 = 2$$

- $\mathbf{n = 4}$: Toplam $4! = 24$ permütasyon bulunur. Bunları elediğimizde 9 tanesinin sabit noktası olmadığını görürüz:

$$!4 = 9$$

Dizi $0, 1, 2, 9, \ldots$ biçiminde ilerler.

### 8.3.3 İçerme-Dışarma ile Genel Formülün Türetilmesi

Genel $!n$ formülü, içerme-dışarma ilkesinin doğrudan bir sonucudur.

**Teorem 8.13** (Düzensiz Dizilim Formülü)**.** *$n \geq 1$ tam sayısı için düzensiz dizilimlerin sayısı:*

$$!n = n!\left(1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + \frac{1}{4!} - \cdots + \frac{(-1)^n}{n!}\right) = n!\sum_{k=0}^{n}\frac{(-1)^k}{k!}$$

*formülü ile hesaplanır.*

*Kanıt.* Tüm permütasyonların kümesi evrensel kümemiz olsun: $|U| = n!$.

Bir permütasyonun koşulu ihlal etmesi, en az bir elemanın kendi yerinde kalması demektir. $i \in \{1, 2, \ldots, n\}$ için:

$$A_i = \{\pi \mid \pi(i) = i\}$$

kümesini tanımlayalım. $A_i$, $i$. elemanın kendi yerinde kaldığı permütasyonların kümesidir.

Düzensiz dizilimler, bu durumların hiçbirinin gerçekleşmediği permütasyonlardır:

$$!n = |U \setminus (A_1 \cup A_2 \cup \cdots \cup A_n)|$$

Tümleyen içerme-dışarma ilkesini yazalım:

$$!n = |U| - \sum_i |A_i| + \sum_{i<j} |A_i \cap A_j| - \sum_{i<j<k} |A_i \cap A_j \cap A_k| + \cdots + (-1)^n |A_1 \cap \cdots \cap A_n|$$

Kesişimlerin büyüklüklerini adım adım hesaplayalım:

1. $|A_i|$: $i$. eleman kendi yerine sabitlenmiştir. Kalan $n-1$ eleman kendi arasında $(n-1)!$ farklı şekilde sıralanabilir:

   $$|A_i| = (n-1)!$$

   Bu türden $\binom{n}{1}$ küme olduğundan:

   $$\sum_i |A_i| = \binom{n}{1}(n-1)! = \frac{n!}{1!(n-1)!}(n-1)! = \frac{n!}{1!}$$

2. $|A_i \cap A_j|$: Hem $i$ hem de $j$ kendi yerindedir (2 eleman sabit). Kalan $n-2$ eleman $(n-2)!$ biçimde dizilir:

$$|A_i \cap A_j| = (n-2)!$$

Bu türden $\binom{n}{2}$ çift olduğundan:

$$\sum_{i<j} |A_i \cap A_j| = \binom{n}{2}(n-2)! = \frac{n!}{2!(n-2)!}(n-2)! = \frac{n!}{2!}$$

3. Benzer biçimde, herhangi $k$ elemanın kendi yerinde sabitlendiği durumların sayısı:

$$\sum_{i_1<\cdots<i_k} |A_{i_1} \cap \cdots \cap A_{i_k}| = \binom{n}{k}(n-k)! = \frac{n!}{k!(n-k)!}(n-k)! = \frac{n!}{k!}$$

Bu terimleri formülde yerine yazarsak:

$$\begin{aligned} !n &= n! - \frac{n!}{1!} + \frac{n!}{2!} - \frac{n!}{3!} + \cdots + (-1)^n \frac{n!}{n!} \\ &= n!\left(1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + \cdots + \frac{(-1)^n}{n!}\right) \end{aligned}$$

elde edilir. □

Formülü küçük değerlerde kontrol edelim:

$$\begin{aligned} !1 &= 1!(1-1) = 0 \\ !2 &= 2!\left(1 - 1 + \frac{1}{2}\right) = 2 \times \frac{1}{2} = 1 \\ !3 &= 6\left(1 - 1 + \frac{1}{2} - \frac{1}{6}\right) = 6 \times \frac{2}{6} = 2 \\ !4 &= 24\left(1 - 1 + \frac{1}{2} - \frac{1}{6} + \frac{1}{24}\right) = 24 \times \frac{12-4+1}{24} = 9 \\ !5 &= 120\left(\frac{9}{24} - \frac{1}{120}\right) = 5 \times 9 - 1 = 44 \end{aligned}$$

Sonuçlar elle bulduğumuz değerlerle bütünüyle örtüşmektedir.

### 8.3.4 Doğal Bir Sabit: 1/e ile Bağlantı

Parantez içindeki ifadeye dikkatle bakalım:

$$1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + \frac{1}{4!} - \cdots + \frac{(-1)^n}{n!}$$

$e^x$ fonksiyonunun Taylor serisini hatırlayalım:

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

Burada $x = -1$ aldığımızda:

$$\frac{1}{e} = e^{-1} = 1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + \frac{1}{4!} - \cdots \approx 0.367879\ldots$$

elde edilir. Bu bağıntı çarpıcı bir gerçeği ortaya koyar: $n$ büyüdükçe rastgele dağıtılan şapkaların **hiçbirinin** doğru sahibine gitmeme olasılığı:

$$\frac{!n}{n!} \approx \frac{1}{e} \approx \%36.8$$

değerine hızla yaklaşır. Kişi sayısı ne kadar büyük olursa olsun, kimsenin kendi şapkasını alamama olasılığı yaklaşık üçte birdir ($\approx \%36.8$).

Pratik hesaplamalarda $!n$, $\frac{n!}{e}$ değerine en yakın tam sayıdır:

$$!n = \left\lfloor \frac{n!}{e} + \frac{1}{2} \right\rfloor \quad (n \geq 1)$$

### 8.3.5 Çözümlü Örnekler

**Örnek 8.14.** *5 mektup ve bu mektupların gönderileceği 5 farklı adresli zarf vardır. Mektuplar zarflara rastgele konulduğunda* ***tam olarak 2*** *mektubun doğru zarfa konulmuş olma sayısı kaçtır?*

*Çözüm.* Bu durum iki aşamada incelenebilir:

1. Doğru zarfa gidecek 2 mektup seçilir.
2. Kalan 3 mektup, hiçbirinin doğru zarfa gitmeyeceği şekilde dağıtılır. (Eğer kalanlardan biri bile doğru zarfa giderse, doğru zarf sayısı 2'yi aşacaktır.)

Doğru zarfa gidecek 2 mektup $\binom{5}{2}$ farklı şekilde seçilebilir:

$$\binom{5}{2} = \frac{5 \times 4}{2} = 10$$

Kalan 3 mektubun kalan 3 zarfa tamamen yanlış dağılması ise düzensiz dizilim sayısı olan $!3$ kadardır:

$$!3 = 2$$

Bölüm 3'teki çarpma kuralı gereği:

$$\binom{5}{2} \times !3 = 10 \times 2 = 20$$

farklı durum bulunur. □

**Örnek 8.15.** *Bir yuvarlak masa etrafında 4 evli çift (toplam 8 kişi) oturacaktır. Kadınlar ve erkekler dönüşümlü (bir kadın, bir erkek) oturacaktır. Kadınlar masaya yerleştikten sonra, hiçbir erkeğin kendi eşinin* ***hemen sağında*** *oturmamasını istiyoruz. Erkekler bu masaya kaç farklı biçimde oturabilir?*

*Çözüm.* Kadınların yerleri belirlendikten sonra aralarında erkekler için 4 boş sandalye kalır. Bu sandalyeleri 1, 2, 3, 4 olarak numaralandıralım; öyle ki $i$. sandalye tam olarak $i$. kadının sağındaki sandalye olsun.

$i$. erkeğin kendi eşinin sağına oturması, $i$. sandalyeye oturması demektir. Hiçbir erkeğin kendi eşinin sağında oturmaması koşulu ise hiçbir $i$. erkeğin $i$. sandalyeye oturmaması anlamına gelir.

Bu durum, 4 erkeğin 4 sandalyeye yerleşiminde sabit noktasız bir permütasyon, yani bir düzensiz dizilim problemidir.

O hâlde aranan oturma düzeni sayısı doğrudan !4 ile verilir:

$$!4 = 9$$

Erkekler, kadınların konumu sabitken bu koşula uygun 9 farklı biçimde oturabilirler. □

**Örnek 8.16.** $\{1, 2, 3, 4, 5, 6\}$ *kümesinin permütasyonları arasından 1'in ilk sırada olmadığı* ($\pi(1) \neq 1$) *ve 6'nın son sırada olmadığı* ($\pi(6) \neq 6$) *kaç farklı permütasyon vardır?*

*Çözüm.* Bu problem tüm elemanların değil, yalnızca iki elemanın konumunu kısıtlamaktadır. Bu nedenle tam bir düzensiz dizilim formülü yerine iki küme içerme-dışarma ilkesini uygulamak yeterlidir.

Tüm permütasyonların sayısı $|U| = 6! = 720$'dir.

İstenmeyen durumlar:

- $A$: 1'in birinci sırada olması ($\pi(1) = 1$). 1 sabitlendiğinden kalan 5 eleman $5! = 120$ şekilde dizilir: $|A| = 120$.
- $B$: 6'nın altıncı sırada olması ($\pi(6) = 6$). 6 sabitlendiğinden kalan 5 eleman $5! = 120$ şekilde dizilir: $|B| = 120$.
- $A \cap B$: Hem $\pi(1) = 1$ hem de $\pi(6) = 6$ olması. Bu iki eleman sabitlendiğinden kalan 4 eleman $4! = 24$ şekilde dizilir: $|A \cap B| = 24$.

İçerme-dışarma kuralına göre istenmeyen durumlardan en az birine düşen permütasyon sayısı:

$$|A \cup B| = |A| + |B| - |A \cap B| = 120 + 120 - 24 = 216$$

Her iki ihlalden de kaçınmak istediğimize göre, evrensel kümeden birleşimi çıkarırız:

$$|\overline{A} \cap \overline{B}| = |U| - |A \cup B| = 720 - 216 = 504$$

Koşulları sağlayan 504 farklı permütasyon vardır. □

**Örnek 8.17.** *İçerme-dışarma mantığını algoritmik olarak nasıl ifade ederiz? N tane kümenin kesişim boyutları verildiğinde genel birleşim boyutunu hesaplayan mantığı gösteriniz.*

*Çözüm.* Programlama yarışmalarında $N$ genellikle 15–20 gibi küçük değerler aldığında, tüm alt kümeleri gezmek için **bitmask** (bitmask) tekniği kullanılır. Bir maskede 1 olan bitler seçilen kümeleri, seçilen küme adedinin tek veya çift olması ise formüldeki + ya da - işaretini belirler:

```
function inclusionExclusion(n, intersectionSize)
    totalUnion = 0

    for mask = 1 to (2^n - 1) do
        elementCount = countSetBits(mask)
        intersection = intersectionSize[mask]

        if elementCount mod 2 == 1 then
            totalUnion = totalUnion + intersection
        else
            totalUnion = totalUnion - intersection
        end if
    end for

    return totalUnion
end function
```

Bu algoritma $2^n - 1$ alt kümenin her birini içerme-dışarma ilkesinin belirlediği katsayıyla $((-1)^{k+1})$ toplama katar. □

# Bölüm 9

# Izgara Yolları ve Yineleme Bağıntıları

Kombinatorik sayma problemlerinde nesneleri tek tek listelemek çoğu zaman imkânsızdır. Ancak problem iki temel yaklaşımdan birine izin verdiğinde karmaşıklık ortadan kalkar: Ya nesneleri geometrik veya sembolik olarak iyi bilinen başka bir yapıya (örneğin iki boyutlu bir ızgara üzerindeki yollara) eşleriz ya da problemi kendisinin daha küçük kopyalarına parçalayarak bir *yineleme bağıntısı* (rekürsiyon) kurarız.

Kafes yolları, ardışık durumlar arasındaki geçişleri modelleyen yineleme ilişkileri, Fibonacci dizisi ve dizilerin kapalı formlarına ulaşmamızı sağlayan karakteristik denklem yöntemi olimpiyat matematiği ile bilgisayar biliminin kesiştiği temel konulardır.

## 9.1 Izgara Yolları

Bir robotun sokakları kare şeklinde parsellenmiş bir şehirde sadece doğuya (sağa) ve kuzeye (yukarı) hareket edebildiğini düşünelim. Robot başlangıç kavşağından hedef kavşağa kaç farklı rota izleyerek gidebilir? Bu soru, ilk bakışta basit bir yol bulma problemi gibi görünse de kombinatorik ve dinamik programlama arasındaki temel köprülerden biridir.

### 9.1.1 Kafes Yolu Modeli ve Temel Sayma

Somut bir örnekle başlayalım. Robotumuz Kartezyen düzlemde $(0, 0)$ noktasında bulunsun ve her adımda yalnızca bir birim sağa ($S$) ya da bir birim yukarı ($Y$) gidebilsin. Robotun $(2, 1)$ noktasına ulaşmasını istiyoruz.

Hedefe varmak için robotun toplamda tam olarak 2 birim sağa ve 1 birim yukarı gitmesi gerektiği açıktır. Hareketlerin sırası değişebilir, ancak atılan toplam adım

sayısı daima $2 + 1 = 3$ olmak zorundadır. Olası tüm rotaları yazalım:

$$SSY, \quad SYS, \quad YSS$$

Toplam 3 farklı yol vardır. Bu listeyi dikkatle incelediğimizde, her yolun 3 harfli bir kelime olduğunu ve bu kelimelerin her birinde tam olarak 2 tane $S$ (ve 1 tane $Y$) bulunduğunu fark ederiz. Aslında Bölüm 5'te gördüğümüz kombinasyon mantığıyla, 3 boşlukluk bir dizilimde sağa gideceğimiz 2 konumu seçmekteyiz:

$$\binom{3}{2} = \binom{3}{1} = 3.$$

**Tanım 9.1** (Kafes Yolu)**.** *Tamsayı koordinatlı noktaların oluşturduğu ızgaraya kafes (lattice) denir.* $(x_1, y_1)$ *noktasından* $(x_2, y_2)$ *noktasına giden, her adımda koordinatları yalnızca* $(x + 1, y)$ *ya da* $(x, y + 1)$ *yapan yollara kuzeydoğu kafes yolu veya kısaca ızgara yolu denir.*

**Teorem 9.2** (Temel Izgara Yolu Teoremi)**.** $(0, 0)$ *noktasından* $(m, n)$ *noktasına, yalnızca 1 birim sağa (*$S$*) ve 1 birim yukarı (*$Y$*) adımlar atılarak gidilebilecek farklı yolların sayısı*

$$\binom{m+n}{m} = \binom{m+n}{n} = \frac{(m+n)!}{m!\,n!}$$

*ile verilir.*

*Kanıt.* $(0, 0)$ noktasından $(m, n)$ noktasına ulaşmak isteyen bir hareketli, $x$ koordinatını 0'dan $m$'ye çıkarmak için tam $m$ adet sağa ($S$) adım, $y$ koordinatını 0'dan $n$'ye çıkarmak için tam $n$ adet yukarı ($Y$) adım atmalıdır.

Toplam adım sayısı zorunlu olarak $m + n$ adettir. Bu adımların hangi sıralamayla atılacağı yolu tekil olarak belirler. Toplam $m + n$ adımdan oluşan bir dizilimde, hangi $m$ tanesinin sağa adım olacağını seçmek $\binom{m+n}{m}$ farklı biçimde yapılabilir. Geriye kalan $n$ pozisyonun tamamı zorunlu olarak yukarı adım ($Y$) olacaktır. Benzer şekilde, $n$ adet yukarı adımın yerini seçmek de $\binom{m+n}{n}$ yolla yapılabilir ve bu iki değer birbirine eşittir. □

Başlangıç noktası $(0, 0)$ yerine keyfi bir $(x_1, y_1)$ noktası ve hedef $(x_2, y_2)$ olduğunda (burada $x_2 \geq x_1$ ve $y_2 \geq y_1$), düzlemi öteleyerek aynı sonucu elde ederiz. Atılması gereken sağa adım sayısı $\Delta x = x_2 - x_1$, yukarı adım sayısı $\Delta y = y_2 - y_1$ olur. Toplam yol sayısı:

$$\binom{\Delta x + \Delta y}{\Delta x} = \binom{(x_2 - x_1) + (y_2 - y_1)}{x_2 - x_1}$$

olur.

### 9.1.2 Ara Noktalardan Geçme ve Engellerden Kaçınma

Yarışma problemlerinde doğrudan $(0,0)$'dan $(m,n)$'ye gitmek nadiren sorulur. Genellikle yolda zorunlu olarak uğranması gereken duraklar veya kesinlikle basılmaması gereken "engelli" noktalar bulunur.

**Örnek 9.3.** *Bir robot* $(0,0)$ *noktasından* $(6,5)$ *noktasına gidecektir.*

*a) Robotun* $(2,3)$ *noktasından geçmesi zorunluysa kaç farklı yol izleyebilir?*

*b) Robotun* $(2,3)$ *noktasındaki çukura düşmemesi (bu noktadan geçmemesi) gerekiyorsa kaç farklı yol izleyebilir?*

*Çözüm.* (a) şıkkında yolu iki bağımsız etaba bölebiliriz: Önce $(0,0)$'dan zorunlu ara nokta olan $(2,3)$'e ulaşmalı, ardından $(2,3)$'ten hedef $(6,5)$'e devam etmeliyiz. Çarpma kuralı bize toplam yolu verecektir.

(b) şıkkında ise "yasaklı noktadan geçmeme" koşulunu doğrudan saymak yerine, tüm serbest yollardan yasaklı noktadan geçen yolları çıkarmak (tümleyen sayma) çok daha pratiktir.

**a)**

1. Aşama: $(0,0)$ noktasından $(2,3)$ noktasına: $\Delta x_1 = 2$, $\Delta y_1 = 3$. Yol sayısı:
$$N_1 = \binom{2+3}{2} = \binom{5}{2} = \frac{5\times 4}{2} = 10.$$

2. Aşama: $(2,3)$ noktasından $(6,5)$ noktasına: $\Delta x_2 = 6-2 = 4$, $\Delta y_2 = 5-3 = 2$. Yol sayısı:
$$N_2 = \binom{4+2}{4} = \binom{6}{2} = \frac{6\times 5}{2} = 15.$$

Çarpma kuralı gereği, $(2,3)$ üzerinden geçen toplam yol sayısı:

$$N_{\text{zorunlu}} = N_1 \times N_2 = 10 \times 15 = 150.$$

**b)** Hiçbir kısıtlama olmasaydı $(0,0)$'dan $(6,5)$'e gidilebilecek tüm yolların sayısı:
$$N_{\text{tüm}} = \binom{6+5}{6} = \binom{11}{5} = \frac{11\times 10\times 9\times 8\times 7}{5\times 4\times 3\times 2\times 1} = 462.$$

$(2,3)$ noktasından geçmeyen yolların sayısı:

$$N_{\text{temiz}} = N_{\text{tüm}} - N_{\text{zorunlu}} = 462 - 150 = 312.$$

□

Birden fazla engelli nokta olduğunda Bölüm 8'de ele aldığımız İçerme-Dışarma İlkesi devreye girer.

**Örnek 9.4.** $(0,0)$ *noktasından* $(5,5)$ *noktasına giden bir hareketli* $A(2,1)$ *ve* $B(3,4)$ *noktalarından geçmek istememektedir. Bu koşulu sağlayan kaç geçerli rota vardır?*

*Çözüm.* Tüm yolların kümesine $U$, $A$'dan geçen yolların kümesine $\mathcal{A}$, $B$'den geçen yolların kümesine $\mathcal{B}$ diyelim. İstenen değer $|\mathcal{A}^c \cap \mathcal{B}^c| = |U| - |\mathcal{A} \cup \mathcal{B}|$'dir. İçerme-Dışarma İlkesi ile:

$$|\mathcal{A} \cup \mathcal{B}| = |\mathcal{A}| + |\mathcal{B}| - |\mathcal{A} \cap \mathcal{B}|.$$

Her bir bileşeni ayrı ayrı hesaplayalım:

$$|U| = \binom{5+5}{5} = \binom{10}{5} = 252,$$
$$|\mathcal{A}| = \binom{2+1}{2} \times \binom{(5-2)+(5-1)}{5-2} = \binom{3}{2} \times \binom{7}{3} = 3 \times 35 = 105,$$
$$|\mathcal{B}| = \binom{3+4}{3} \times \binom{(5-3)+(5-4)}{5-3} = \binom{7}{3} \times \binom{3}{2} = 35 \times 3 = 105.$$

Hem $A$'dan hem $B$'den geçen yollar için rota zorunlu olarak $(0,0) \to A(2,1) \to B(3,4) \to (5,5)$ sırasını izlemelidir:

$$|\mathcal{A} \cap \mathcal{B}| = \binom{2+1}{2} \times \binom{(3-2)+(4-1)}{3-2} \times \binom{(5-3)+(5-4)}{5-3}$$
$$= \binom{3}{2} \times \binom{4}{1} \times \binom{3}{2} = 3 \times 4 \times 3 = 36.$$

Dolayısıyla en az bir engelden geçen yollar:

$$|\mathcal{A} \cup \mathcal{B}| = 105 + 105 - 36 = 174.$$

Her iki engeli de atlatan yolların sayısı:

$$252 - 174 = 78.$$

□

**Örnek 9.5** (Köşegeçen Sınırı — Yansıma Fikrine Giriş)**.** $(0,0)$ *noktasından* $(n,n)$ *noktasına giden ve* $y > x$ *bölgesine asla geçmeyen (yani daima köşegen üzerinde veya altında kalan,* $y \leq x$ *şartını sağlayan) ızgara yollarının sayısını* $n = 3$ *için elle listeleyelim ve genel formülü tartışalım.*

*Çözüm.* Her adımda $x$ koordinatımız sağa adımların sayısını, $y$ koordinatımız ise yukarı adımların sayısını verir. $y \leq x$ koşulu, yolun herhangi bir anında atılan yukarı adım sayısının sağa adım sayısını aşamayacağı anlamına gelir. Bu, açılan ve kapanan parantezlerin dengeli olma şartıyla birebir aynıdır: Her $S$ adımını bir sol parantez "(' ve her $Y$ adımını bir sağ parantez ")' olarak görebiliriz.

$n = 3$ için toplam 3 adet $S$ ve 3 adet $Y$ vardır. Geçerli dizilimleri yazalım:

1. $SSSYYY \longleftrightarrow ((()))$
2. $SSYSYY \longleftrightarrow (()())$
3. $SSYYSY \longleftrightarrow (())()$
4. $SYSSYY \longleftrightarrow ()(())$
5. $SYSYSY \longleftrightarrow ()()()$

Toplam 5 yol bulunur.

Genel durumda $(0,0)$'dan $(n,n)$'ye serbest tüm yolların sayısı $\binom{2n}{n}$'dir. $y = x$ çizgisinin üstüne çıkan (yani $y = x+1$ doğrusuna değen) geçersiz yolların sayısı André'nin Yansıma İlkesi (Reflection Principle) ile hesaplandığında $\binom{2n}{n-1}$ çıkar. Dolayısıyla geçerli yol sayısı:

$$C_n = \binom{2n}{n} - \binom{2n}{n-1} = \frac{1}{n+1}\binom{2n}{n}$$

olur. Bu sayılara $n$. *Catalan sayısı* denir. $n = 3$ için $C_3 = \frac{1}{4}\binom{6}{3} = \frac{20}{4} = 5$ doğrulanır. □

### 9.1.3 Izgara Yollarının Kodlanması

Büyük ızgaralarda yolların sayısını bir modulo değerinde (örneğin $10^9 + 7$) hesaplarken iki temel yol vardır: Faktöriyeller üzerinden kombinasyon almak ($O(n)$) veya dinamik programlama ile kafesi doldurmak ($O(m \times n)$). Engel sayısı çok olduğunda dinamik programlama doğrudan uygulanır:

```
FUNCTION countGridPaths(m, n, obstacles)
    MOD = 10^9 + 7
    INITIALIZE 2D array dp of size (m + 1) x (n + 1) with 0

    IF (0, 0) NOT IN obstacles THEN
        dp[0][0] = 1
    ELSE
        dp[0][0] = 0
    END IF

    FOR i FROM 0 TO m DO
        FOR j FROM 0 TO n DO
            IF (i, j) IN obstacles OR (i == 0 AND j == 0) THEN
                CONTINUE
            END IF

            fromTop = (i > 0) ? dp[i - 1][j] : 0
            fromLeft = (j > 0) ? dp[i][j - 1] : 0

            dp[i][j] = (fromTop + fromLeft) MOD MOD
```

```
        END FOR
    END FOR

    RETURN dp[m][n]
END FUNCTION
```

```
#define MOD 1000000007

int dp[1005][1005];
int engel[1005][1005]; // 1 ise engelli, 0 ise serbest

int izgara_yollari(int m, int n) {
    for (int i = 0; i <= m; i++) {
        for (int j = 0; j <= n; j++) {
            if (engel[i][j]) {
                dp[i][j] = 0;
                continue;
            }
            if (i == 0 && j == 0) {
                dp[i][j] = 1;
            } else {
                long long sol = (i > 0) ? dp[i - 1][j] : 0;
                long long asagi = (j > 0) ? dp[i][j - 1] : 0;
                dp[i][j] = (sol + asagi) % MOD;
            }
        }
    }
    return dp[m][n];
}
```

Bu tablodaki $dp[i][j] = dp[i-1][j] + dp[i][j-1]$ geçişi, bir durumu önceki durumlar cinsinden ifade etmenin doğal bir örneğidir.

## 9.2 Yineleme Bağıntısı Kavramı

Doğrudan kombinatorik bir formül (örneğin tek bir binom katsayısı) bulmak her zaman kolay değildir. Karmaşık sayma problemlerinde temel strateji şudur: **"Boyutu $n$ olan bir problemin çözümünü, boyutu $n$'den daha küçük olan aynı problemin çözümleri cinsinden yazabilir miyiz?"**

Bu soruya verilen olumlu yanıt bir *yineleme bağıntısı* (recurrence relation) kurmamızı sağlar.

**Tanım 9.6** (Yineleme Bağıntısı ve Başlangıç Koşulları)**.** *Bir $\{a_n\}$ dizisinde, $n$. terimi kendisinden önceki terimler $(a_{n-1}, a_{n-2}, \ldots, a_{n-k})$ cinsinden ifade eden bir eşitliğe yineleme bağıntısı denir. Bağıntının çözülebilmesi ve her terimin tek bir sayısal değere karşılık gelebilmesi için verilen ilk terimlere başlangıç koşulları (taban durumlar) denir.*

### 9.2.1 "Son Hamleye Bakma" Tekniği

Yineleme bağıntısı kurarken izlenen temel kural, **yapılan en son işlemi veya dizilimin en son elemanını sınıflandırmaktır.**

Somut bir problemle bu yaklaşımı inceleyelim: Uzunluğu $n$ olan ve yalnızca "A', "B', "C' harflerinden oluşan kelimeler yazmak istiyoruz. Kısıtlamamız şudur: "A' harfinden hemen sonra yine "A' harfi gelemez (yan yana iki "A' bulunamaz). Bu şartı sağlayan $n$ uzunluğundaki kelime sayısına $a_n$ diyelim.

$n = 1$ için: "A", "B", "C" $\implies a_1 = 3$. $n = 2$ için: Toplam $3^2 = 9$ kelimeden sadece "AA" yasaktır $\implies a_2 = 9 - 1 = 8$.

Peki $a_n$'i $a_{n-1}$ cinsinden nasıl kurarız? Boyutu $n$ olan geçerli bir kelimenin **son harfine** bakalım:

- **Durum 1: Kelime "B' ile bitiyor.** Son harfi sildiğimizde geriye kalan $n-1$ uzunluğundaki kelime geçerli bir kelimedir. "B' harfi herhangi bir yasak oluşturmadığından, $n-1$ uzunluğundaki *her* geçerli kelimenin sonuna "B' ekleyebiliriz. Buradan $a_{n-1}$ tane durum gelir.
- **Durum 2: Kelime "C' ile bitiyor.** Tıpkı "B' gibi, $n-1$ uzunluğundaki her geçerli kelimenin sonuna "C' eklenebilir. Buradan da $a_{n-1}$ tane durum gelir.
- **Durum 3: Kelime "A' ile bitiyor.** Son harf "A' ise bir önceki harf "A' olamaz. O halde son iki harf ya "BA" ya da "CA" şeklinde bitmelidir. Son iki harfi sildiğimizde geriye $n-2$ uzunluğunda geçerli bir kelime kalır. Tersine, $n-2$ uzunluğundaki herhangi bir geçerli kelimenin sonuna "BA" veya "CA" eklersek, yan yana iki "A' içermeyen $n$ uzunluğunda geçerli bir kelime elde ederiz. Buradan $2a_{n-2}$ durum gelir.

Bu durumlar ayrık ve olası tüm dizilimleri kapsadığından:

$$a_n = 2a_{n-1} + 2a_{n-2}, \quad (n \geq 3)$$

bağıntısını buluruz. Başlangıç koşulları $a_1 = 3$ ve $a_2 = 8$'dir. Bağıntıyı test edelim:

$$a_3 = 2a_2 + 2a_1 = 2(8) + 2(3) = 16 + 6 = 22.$$

Tek tek sayarak da $3^3 - (\text{"AAA", "AAB", "AAC", "BAA", "CAA"}) = 27 - 5 = 22$ olduğunu doğrulayabiliriz.

### 9.2.2 Modelleme Örnekleri

**Örnek 9.7** (Hanoi Kuleleri)**.** *3 direk ve farklı boyutlarda n adet disk verilmiştir. Başlangıçta tüm diskler küçük disk daima büyük diskin üzerinde olacak şekilde A direğindedir. Her hamlede yalnızca bir disk başka bir direğe taşınabilir ve hiçbir disk kendisinden küçük bir diskin üzerine konulamaz. Tüm diskleri C direğine taşımak için gereken minimum hamle sayısı $H_n$ kaçtır?*

*Çözüm.* En büyük diski (en alttaki $n$. diski) A direğinden C direğine taşıyabilmek için üzerindeki $n-1$ diskin tamamı B direğinde toplanmış olmalıdır.

1. Adım: En üstteki $n-1$ diski C'yi ara durak olarak kullanarak B'ye taşı. Bu işlem $H_{n-1}$ hamle sürer.
2. Adım: A'da yalnız kalan en büyük diski doğrudan C'ye taşı. Bu işlem 1 hamle sürer.
3. Adım: B'de duran $n-1$ diski A'yı ara durak olarak kullanarak C'ye, en büyük diskin üzerine taşı. Bu işlem de $H_{n-1}$ hamle sürer.

Bu üç adım zorunludur ve daha az hamleyle bu transfer yapılamaz. O halde:

$$H_n = 2H_{n-1} + 1, \quad H_1 = 1.$$

İlk birkaç terimi hesaplayarak yapıyı görelim:

$$\begin{aligned} H_1 &= 1, \\ H_2 &= 2(1) + 1 = 3, \\ H_3 &= 2(3) + 1 = 7, \\ H_4 &= 2(7) + 1 = 15. \end{aligned}$$

Bu değerler genel formülün $H_n = 2^n - 1$ olduğunu düşündürür. Bölüm 2'deki tümevarım yöntemiyle kanıtlayalım: $H_1 = 2^1 - 1 = 1$ doğrudur. $H_k = 2^k - 1$ kabul edersek,

$$H_{k+1} = 2(2^k - 1) + 1 = 2^{k+1} - 2 + 1 = 2^{k+1} - 1$$

elde edilir ve ispat tamamlanır. □

**Örnek 9.8** (Düzlemi Bölen Doğrular)**.** *Düzlemde bulunan $n$ adet doğru, hiçbir ikisi birbirine paralel olmayacak ve hiçbir üçü tek bir noktada kesişmeyecek şekilde çizilmiştir (genel konumdalar). Bu $n$ doğrunun düzlemi kaç ayrık bölgeye ayırdığını ($R_n$) bulunuz.*

*Çözüm.* Düzlemde halihazırda $n-1$ doğru çizilmiş olsun. $n$. doğruyu çizdiğimizde yeni kaç bölge oluşur? Yeni çizilen doğru, mevcut $n-1$ doğrunun her birini farklı bir noktada keser (paralel değiller ve üçü aynı noktadan geçmiyor). Dolayısıyla $n$. doğru üzerinde $n-1$ adet kesişim noktası oluşur. Bu $n-1$ nokta, $n$. doğruyu tam $n$ adet parçaya (ışın ve doğru parçalarına) ayırır. Bu $n$ parçanın her biri, önceden var olan bir bölgeyi tam ikiye böler; yani düzleme $n$ adet yeni bölge eklenir.

Buradan yineleme bağıntısı:

$$R_n = R_{n-1} + n, \quad R_0 = 1 \text{ (sıfır doğru varken düzlem 1 tek parçadır).}$$

Teleskopik toplam yaparsak:

$$\begin{aligned} R_n - R_{n-1} &= n \\ R_{n-1} - R_{n-2} &= n-1 \\ &\vdots \\ R_1 - R_0 &= 1 \end{aligned}$$

Taraf tarafa topladığımızda:

$$R_n - R_0 = 1 + 2 + \cdots + n = \frac{n(n+1)}{2} \implies R_n = \frac{n(n+1)}{2} + 1 = \frac{n^2+n+2}{2}.$$

□

**Örnek 9.9** (Ardışık Sıfır İçermeyen İkili Diziler)**.** *Uzunluğu $n$ olan ve yan yana iki tane "0' içermeyen ikili $(0-1)$ dizilerin sayısı $a_n$ olsun. $a_n$ için bir yineleme bağıntısı bulunuz.*

*Çözüm.* Son bite odaklanalım:

- Dizi "1' ile bitiyorsa: Önündeki $n-1$ bit, yan yana iki 0 içermeyen herhangi bir geçerli dizi olabilir. Buradan $a_{n-1}$ durum gelir.
- Dizi "0' ile bitiyorsa: Yan yana iki sıfır olamayacağından, bir önceki bit zorunlu olarak "1' olmalıdır. Yani dizi "10" ile bitmelidir. Bu son iki biti kaldırdığımızda, önünde kalan $n-2$ bit geçerli bir dizi olmalıdır. Buradan $a_{n-2}$ durum gelir.

O halde bağıntımız:

$$a_n = a_{n-1} + a_{n-2}$$

olur. Taban durumlar: $n = 1$ için: "0", "1" $\implies a_1 = 2$. $n = 2$ için: "01", "10", "11" $\implies a_2 = 3$. Bu dizi bizi doğrudan Fibonacci sayılarına ulaştırır. □

**Tanım 9.10** (En Sık Yapılan Hata: Sayma Durumlarının Kesişmesi)**.** *Yineleme bağıntısı kurarken durumları ayrıştırırken durumların **ayrık** olduğundan emin olunmalıdır. Eğer durumlar kesişiyorsa aynı nesne birden fazla sayılır ve sonuç hatalı çıkar. Durumların ortak bir eleman içerip içermediği her zaman denetlenmelidir.*

## 9.3 Fibonacci ve Uygulamaları

Fibonacci dizisi, doğadaki biçimsel yapılardan algoritmaların çalışma zamanı analizine kadar matematikte ve bilgisayar biliminde en sık karşılaşılan dizilerden biridir.

**Tanım 9.11** (Fibonacci Dizisi)**.** *$F_0 = 0$, $F_1 = 1$ olmak üzere, $n \geq 2$ için*

$$F_n = F_{n-1} + F_{n-2}$$

*biçiminde tanımlanan diziye Fibonacci dizisi denir. İlk birkaç terimi:*

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, \ldots$$

Olimpiyatlarda Fibonacci bağıntısı pek çok farklı kılıkta karşımıza çıkar.

### 9.3.1 Merdiven Çıkma Problemi

Bir çocuk $n$ basamaklı bir merdiveni tırmanmaktadır. Çocuk her adımında ya 1 basamak ya da 2 basamak yukarı çıkabilmektedir. Çocuğun $n$ basamağı çıkması için kaç farklı adım dizilimi vardır?

Merdivenin tepesine ($n$. basamağa) ulaşacak son adımı ele alalım. Çocuk son hamlesinde ya $(n-1)$. basamaktan 1 basamak çıkmıştır ya da $(n-2)$. basamaktan 2 basamak çıkmıştır.

- Eğer son adım 1 basamaklık ise önceki $n-1$ basamağı çıkmanın yollarının sayısı $M_{n-1}$'dir.
- Eğer son adım 2 basamaklık ise önceki $n-2$ basamağı çıkmanın yollarının sayısı $M_{n-2}$'dir.

Bu iki durum birbirini dışlar ve tüm olası bitişleri kapsar:

$$M_n = M_{n-1} + M_{n-2}.$$

Taban durumlar: 1 basamak için: (1) $\implies$ $M_1 = 1$. 2 basamak için: (1+1), (2) $\implies$ $M_2 = 2$. 3 basamak için: (1+1+1), (1+2), (2+1) $\implies$ $M_3 = 3$. Dizimiz $1, 2, 3, 5, 8, \ldots$ şeklinde devam eder. Görüldüğü üzere $M_n = F_{n+1}$ eşitliği geçerlidir.

### 9.3.2 $2 \times n$ Boyutlu Tahtayı Domino Taşlarıyla Kaplama

Elimizde sınırsız sayıda $2 \times 1$ ve $1 \times 2$ boyutunda domino taşları bulunmaktadır. Bu taşlarla $2 \times n$ boyutundaki bir tahtayı hiç boşluk bırakmadan ve taşlar üst üste gelmeden kaplamak istiyoruz. Kaç farklı kaplama yapılabilir?

Tahtanın en sağ sütununa odaklanalım:

- En sağ sütunu dikey bir domino ($2 \times 1$) ile kapatabiliriz. Bu durumda geriye $2 \times (n-1)$ boyutunda bir tahta kalır. Bu tahtayı kaplamanın yolu $D_{n-1}$ tanedir.

- Eğer en sağ sütunun üst karesine yatay bir domino ($1 \times 2$) koyarsak, altındaki kareyi kapatmak için onun altına da mecburen ikinci bir yatay domino koymak zorundayız. Bu iki yatay domino birlikte $2 \times 2$'lik bir blok oluşturur. Geriye $2 \times (n-2)$ boyutunda bir tahta kalır. Bu tahtayı kaplamanın yolu $D_{n-2}$ tanedir.

Başka bir ihtimal yoktur. Dolayısıyla:

$$D_n = D_{n-1} + D_{n-2}.$$

Taban durumlar: $D_1 = 1$ (tek dikey domino), $D_2 = 2$ (iki dikey ya da iki yatay). Burada da $D_n = F_{n+1}$ bağıntısı karşımıza çıkar.

### 9.3.3 Fibonacci Özdeşlikleri ve İki Yoldan Sayma

Fibonacci sayıları arasında pek çok cebirsel özdeşlik mevcuttur. Bu özdeşlikleri cebirsel manipülasyonlarla veya tümevarımla ispatlayabileceğimiz gibi, Bölüm 10'da ayrıntılandıracağımız iki yoldan sayma (çifte sayım) yöntemiyle de doğrudan gösterebiliriz.

**Teorem 9.12** (Fibonacci Kareleri Toplamı)**.** *Tüm $n \geq 1$ için:*

$$F_1^2 + F_2^2 + F_3^2 + \cdots + F_n^2 = F_n F_{n+1}.$$

*Kanıt.* Kenar uzunlukları $F_1, F_2, F_3, \ldots, F_n$ olan kareleri sırasıyla yan yana ve üst üste ekleyerek geometrik bir dikdörtgen inşa edelim:

- Kenarı $F_1 = 1$ olan bir kare ile başlarız.
- Yanına kenarı $F_2 = 1$ olan bir kare koyarız: $1 \times 2$'lik dikdörtgen oluşur.
- Altına kenarı $F_3 = 2$ olan kareyi koyarız: $2 \times 3$'lük dikdörtgen oluşur ($F_3 \times (F_1 + F_2) = F_3 \times F_3$).
- Yanına kenarı $F_4 = 3$ olan kareyi koyarız: $3 \times 5$'lik dikdörtgen oluşur ($F_4 \times (F_2 + F_3) = F_4 \times F_4$).
- Bu şekilde $n$. kareyi eklediğimizde, oluşan büyük şekil bir dikdörtgendir ve kenar uzunlukları $F_n$ ile $F_{n+1}$ olur.

Büyük dikdörtgenin alanı iki farklı yoldan hesaplanabilir:

1. İçindeki karelerin alanlarının toplamı: $F_1^2 + F_2^2 + \cdots + F_n^2$.
2. Büyük dikdörtgenin eni ile boyunun çarpımı: $F_n \times F_{n+1}$.

Her iki ifade aynı alanı temsil ettiğinden eşitlik sağlanır. □

**Teorem 9.13** (Fibonacci Toplamı)**.** *Tüm $n \geq 1$ için:*

$$\sum_{i=1}^{n} F_i = F_1 + F_2 + \cdots + F_n = F_{n+2} - 1.$$

*Kanıt.* Fibonacci tanımından $F_k = F_{k+2} - F_{k+1}$ yazabiliriz. Bu eşitliği $k = 1$'den $n$'ye kadar alt alta yazalım:

$$\begin{aligned} F_1 &= F_3 - F_2 \\ F_2 &= F_4 - F_3 \\ F_3 &= F_5 - F_4 \\ &\vdots \\ F_n &= F_{n+2} - F_{n+1} \end{aligned}$$

Taraf tarafa topladığımızda sağ taraftaki terimler teleskopik olarak birbirini götürür:

$$\sum_{i=1}^{n} F_i = F_{n+2} - F_2.$$

$F_2 = 1$ olduğundan $\sum_{i=1}^{n} F_i = F_{n+2} - 1$ elde edilir. □

### 9.3.4 Fibonacci Sayılarının Hesaplanması

Bir bilgisayar yarışmasında $F_n \pmod{10^9 + 7}$ istendiğinde doğrudan rekürsiyon ($F(n) = F(n-1) + F(n-2)$ çağrısı) $O(2^n)$ karmaşıklığa yol açacağı için zaman sınırını aşar ($n = 45$ bile saniyeler sürer).

Doğrusal zamanda ($O(n)$) döngü ile çözüm:

```
FUNCTION fibonacci(n, mod = 10^9 + 7)
    IF n = 0 THEN
        RETURN 0
    END IF

    a = 0
    b = 1

    FOR i FROM 2 TO n DO
        next_val = (a + b) MOD mod
        a = b
        b = next_val
    END FOR

    RETURN b
END FUNCTION
```

```
#define MOD 1000000007

int fibonacci(int n) {
    if (n == 0) return 0;
    long long a = 0, b = 1;
    for (int i = 2; i <= n; i++) {
        long long c = (a + b) % MOD;
        a = b;
        b = c;
    }
    return (int)b;
}
```

## 9.4 Kapalı Form Sezgisi

Yineleme bağıntısı terimleri adım adım üretmemizi sağlar. Ancak bir problemde $n = 10^9$ verildiğinde, $O(n)$ karmaşıklığındaki bir döngü bile zaman aşımına uğrar. Bu durumda doğrudan $n$'i yerine koyup terimi hesaplayabileceğimiz bir bağıntı gerekir. Buna *kapalı form* (closed form) denir.

### 9.4.1 Karakteristik Denklem Fikrinin Doğuşu

Birinci dereceden bir yineleme bağıntısını ele alalım:

$$a_n = r \cdot a_{n-1}$$

Bu bir geometrik dizidir ve çözümü doğrudan $a_n = a_0 \cdot r^n$ şeklindedir.

İkinci dereceden homojen doğrusal bir bağıntı verildiğinde:

$$a_n - c_1 a_{n-1} - c_2 a_{n-2} = 0$$

Çözümün yine geometrik dizi formunda, yani $a_n = r^n$ biçiminde olabileceğini varsayabilir miyiz?

$a_n = r^n$ ifadesini bağıntıda yerine koyalım ($r \neq 0$ varsayımıyla):

$$r^n - c_1 r^{n-1} - c_2 r^{n-2} = 0.$$

Eşitliğin her iki tarafını $r^{n-2}$ ile bölelim:

$$r^2 - c_1 r - c_2 = 0.$$

Yineleme bağıntısı böylece $r$ değişkenine bağlı ikinci dereceden bir cebirsel denkleme dönüşür. Bu denkleme bağıntının *karakteristik denklemi* denir.

**Teorem 9.14** (Farklı Kökler Durumu)**.** $r^2 - c_1 r - c_2 = 0$ *karakteristik denkleminin iki farklı kökü* $r_1$ *ve* $r_2$ *olsun* ($r_1 \neq r_2$)*. O halde* $a_n = c_1 a_{n-1} + c_2 a_{n-2}$ *bağıntısının genel çözümü,* $A$ *ve* $B$ *başlangıç koşulları tarafından belirlenen sabitler olmak üzere şöyledir:*

$$a_n = A \cdot r_1^n + B \cdot r_2^n.$$

*İspat Sezgisi.* Eğer $r_1^n$ ve $r_2^n$ denklemi ayrı ayrı sağlıyorsa, bu dizilerin herhangi bir doğrusal kombinasyonu olan $Ar_1^n + Br_2^n$ de denklemi sağlar (çünkü sistem doğrusaldır). $n = 0$ ve $n = 1$ için:

$$\begin{aligned} a_0 &= A + B \\ a_1 &= Ar_1 + Br_2 \end{aligned}$$

$r_1 \neq r_2$ olduğundan bu iki bilinmeyenli lineer denklem sistemi $A$ ve $B$ için tekil bir çözüme sahiptir. □

### 9.4.2 Binet Formülü: Fibonacci'nin Kapalı Formu

Bu tekniği Fibonacci dizisine uygulayalım:

$$F_n = F_{n-1} + F_{n-2} \implies F_n - F_{n-1} - F_{n-2} = 0.$$

Karakteristik denklem:

$$r^2 - r - 1 = 0.$$

Kökleri bulalım:

$$r_{1,2} = \frac{-(-1) \pm \sqrt{(-1)^2 - 4(1)(-1)}}{2} = \frac{1 \pm \sqrt{5}}{2}.$$

Bu kökler altın oran $\phi = \frac{1+\sqrt{5}}{2} \approx 1.618$ ve onun eşleniği $\psi = \frac{1-\sqrt{5}}{2} \approx -0.618$'dir. Genel çözüm:

$$F_n = A\left(\frac{1+\sqrt{5}}{2}\right)^n + B\left(\frac{1-\sqrt{5}}{2}\right)^n.$$

Başlangıç koşullarını ($F_0 = 0, F_1 = 1$) kullanalım:

$$\begin{aligned} n = 0 &\implies A + B = 0 \implies B = -A \\ n = 1 &\implies A\left(\frac{1+\sqrt{5}}{2}\right) - A\left(\frac{1-\sqrt{5}}{2}\right) = 1 \\ &\qquad A\left(\frac{1+\sqrt{5}-1+\sqrt{5}}{2}\right) = 1 \implies A\sqrt{5} = 1 \implies A = \frac{1}{\sqrt{5}}. \end{aligned}$$

Böylece $B = -\frac{1}{\sqrt{5}}$ olur ve kapalı form elde edilir:

**Teorem 9.15** (Binet Formülü)**.** *n. Fibonacci sayısı için kapalı form:*

$$F_n = \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right].$$

İçinde irrasyonel $\sqrt{5}$ terimleri ve kesirler bulunan bu ifade, her $n$ pozitif tamsayısı için bir tamsayı üretir; çünkü Bölüm 6'da gördüğümüz binom açılımı uygulandığında $\sqrt{5}$'li tek dereceli terimler birbirini götürür.

Ayrıca $|\psi| = |\frac{1-\sqrt{5}}{2}| \approx 0.618 < 1$ olduğundan, $n$ büyüdükçe $\psi^n \to 0$ olur. Dolayısıyla değer:

$$F_n = \text{round}\left(\frac{\phi^n}{\sqrt{5}}\right)$$

şeklinde en yakın tamsayıya yuvarlanarak da hesaplanabilir.

### 9.4.3 Karakteristik Denklem Uygulamaları

**Örnek 9.16.** *$a_0 = 1$, $a_1 = 4$ ve her $n \geq 2$ için $a_n = 5a_{n-1} - 6a_{n-2}$ olarak verilen dizinin kapalı formunu bulunuz.*

*Çözüm.* Terimleri tek tarafa toplayarak karakteristik denklemi yazarız:

$$a_n - 5a_{n-1} + 6a_{n-2} = 0 \implies r^2 - 5r + 6 = 0.$$

Çarpanlarına ayıralım:

$$(r-2)(r-3) = 0 \implies r_1 = 2, \quad r_2 = 3.$$

Kökler birbirinden farklı olduğundan genel çözüm:

$$a_n = A \cdot 2^n + B \cdot 3^n.$$

$a_0 = 1$ ve $a_1 = 4$ başlangıç koşullarını uygulayalım:

$$n = 0 \implies A \cdot 2^0 + B \cdot 3^0 = 1 \implies A + B = 1$$
$$n = 1 \implies A \cdot 2^1 + B \cdot 3^1 = 4 \implies 2A + 3B = 4$$

Birinci denklemi 2 ile çarpıp ikinciden çıkarırsak:

$$(2A + 3B) - (2A + 2B) = 4 - 2 \implies B = 2.$$

$A + B = 1$ olduğundan $A = 1 - 2 = -1$ bulunur. Böylece kapalı form:

$$a_n = -1 \cdot 2^n + 2 \cdot 3^n = 2 \cdot 3^n - 2^n.$$

İlk birkaç terim kontrol edildiğinde: $a_0 = 2(1) - 1 = 1$, $a_1 = 2(3) - 2 = 4$, $a_2 = 5(4) - 6(1) = 14$ ve formülden $a_2 = 2 \cdot 3^2 - 2^2 = 18 - 4 = 14$ eşitliği doğrulanır. □

### 9.4.4 Çakışık Kök Durumu ($r_1 = r_2$)

Karakteristik denklemin deltası sıfır olduğunda (örneğin $r^2 - 4r + 4 = 0$ ise tek kök $r = 2$'dir) çözüm yalnızca $a_n = A \cdot 2^n$ olarak yazılamaz; çünkü tek katsayı iki başlangıç koşulunu ($a_0$ ve $a_1$) aynı anda sağlamaya yetmez.

Lineer diferansiyel denklemlerde olduğu gibi, kök katlı olduğunda ikinci bağımsız çözüm $n \cdot r^n$ teriminden gelir.

**Teorem 9.17** (Çakışık Kökler Durumu)**.** *$r^2 - c_1 r - c_2 = 0$ karakteristik denkleminin tek bir çift katlı kökü varsa ($r_1 = r_2 = r$), genel çözüm:*

$$a_n = (A + B \cdot n) r^n$$

*biçimindedir.*

**Örnek 9.18.** *$a_0 = 1$, $a_1 = 6$ ve her $n \geq 2$ için $a_n = 4a_{n-1} - 4a_{n-2}$ olan dizinin kapalı formunu bulunuz.*

*Çözüm.* Karakteristik denklem:

$$r^2 - 4r + 4 = 0 \implies (r-2)^2 = 0 \implies r = 2 \text{ (çift katlı kök).}$$

Genel form:

$$a_n = (A + Bn) \cdot 2^n.$$

Başlangıç koşullarını uygulayalım:

$$n = 0 \implies (A + 0) \cdot 2^0 = 1 \implies A = 1$$
$$n = 1 \implies (1 + B) \cdot 2^1 = 6 \implies 2(1 + B) = 6 \implies 1 + B = 3 \implies B = 2.$$

Böylece kapalı form:

$$a_n = (1 + 2n) \cdot 2^n$$

olarak bulunur. Doğrulayalım: Yinelemeden $a_2 = 4(6) - 4(1) = 20$, kapalı formdan $a_2 = (1+4) \cdot 2^2 = 5 \times 4 = 20$ elde edilir. □

### 9.4.5 Çözümlü Bir Problem

**Örnek 9.19.** *$n \geq 1$ için, $1 \times n$ boyutundaki bir şerit $1 \times 1$'lik kırmızı, mavi veya yeşil karelerle ya da $1 \times 2$'lik sarı dikdörtgenlerle döşenecektir. Yan yana iki kırmızı karenin bulunmadığı geçerli döşemelerin sayısını $a_n$ olarak tanımlayalım.*

*a) $a_n$ için bir yineleme bağıntısı kurunuz.*

*b) $a_1$ ve $a_2$ değerlerini hesaplayınız.*

*Çözüm.* Şeridin en son parçasına odaklanarak türüne ve rengine göre durumları ayrıştıralım:

- **Son parça $1 \times 2$ sarı dikdörtgen ise:** Geriye $1 \times (n-2)$ şerit kalır. Sarı taş kırmızı kısıtını etkilemediğinden buradan $a_{n-2}$ durum gelir.
- **Son parça $1 \times 1$ mavi veya yeşil kare ise:** 2 farklı renk seçeneği vardır. Bu renkler kırmızı olmadığından bir önceki kare ne olursa olsun kural bozulmaz. Kalan $1 \times (n-1)$ şerit için $2a_{n-1}$ durum gelir.
- **Son parça $1 \times 1$ kırmızı kare ise:** Bir önceki parça kırmızı $1 \times 1$ olamaz. Sondan bir önceki parçanın durumuna bakalım:
  - Sondan bir önceki parça $1 \times 2$ sarı ise: Son iki parça (sarı, kırmızı) toplam 3 birim kaplar. Kalan $n-3$ uzunluk serbesttir $\implies$ $a_{n-3}$ durum.
  - Sondan bir önceki parça $1 \times 1$ mavi veya yeşil ise: Son iki parça (mavi/-yeşil, kırmızı) 2 birim kaplar. Kalan $n-2$ uzunluk serbesttir $\implies$ $2a_{n-2}$ durum.
  - Sondan bir önceki parça kırmızı olamaz.

Tüm bu durumları toplayarak homojen bağıntıyı kurarız:

$$a_n = 2a_{n-1} + (a_{n-2} + 2a_{n-2}) + a_{n-3} = 2a_{n-1} + 3a_{n-2} + a_{n-3}.$$

Başlangıç terimlerini belirleyelim:

- $n = 1$: Yalnızca $1 \times 1$ kareler kullanılabilir. Kırmızı, Mavi, Yeşil $\implies$ $a_1 = 3$.
- $n = 2$:
  - 1 adet $1 \times 2$ sarı: 1 durum.
  - 2 adet $1 \times 1$ kare: Toplam $3^2 = 9$ durum vardır. Sadece (Kırmızı, Kırmızı) yasaktır, yani $9 - 1 = 8$ durum.

  Toplam: $a_2 = 1 + 8 = 9$.
- $n = 3$: Bağıntı için $a_0$'a ihtiyaç duyarız. Boş şeridi döşemenin 1 yolu vardır ($a_0 = 1$). Bağıntıdan:

$$a_3 = 2a_2 + 3a_1 + a_0 = 2(9) + 3(3) + 1 = 18 + 9 + 1 = 28.$$

□

**Tanım 9.20** (Tuzak ve Uyarı: İndeks Kayması)**.** *Yineleme bağıntılarını kodlarken veya kapalı form çözerken sık yapılan hata, başlangıç indekslerinin ($n = 0$ veya $n = 1$) karıştırılmasıdır. Örneğin merdiven probleminde $M_1 = 1, M_2 = 2$ iken standart Fibonacci tanımında $F_1 = 1, F_2 = 1$'dir; yani $M_n = F_{n+1}$ şeklinde bir indeks kayması mevcuttur. Taban durumlar daima küçük değerlerle ($n = 0, 1, 2$) elle kontrol edilmeli ve bağıntının ürettiği ilk terimler doğrulanmalıdır.*

Izgara yollarının kombinatorik yapısı ile yineleme bağıntılarının tümevarımsal mantığı birleştiğinde, karmaşık sayma problemleri sistematik biçimde çözülebilir. İlerleyen konularda bu yaklaşım dinamik programlama algoritmalarının da temelini oluşturacaktır.

# Bölüm 10

# Çifte Sayım ve Birebir Eşleme

Matematik ve bilgisayar biliminde bir problemi çözmenin en etkili yollarından biri, karmaşık cebirsel sadeleştirmeler veya uzun denklemler kurmak yerine saymak istediğimiz nesnelere farklı bir açıdan bakmaktır. Bir büyüklüğü iki farklı yöntemle sayıp sonuçları birbirine eşitlemek ya da zor bir kümenin elemanlarını yapısını çok iyi bildiğimiz başka bir kümenin elemanlarıyla birebir eşleştirmek, kombinatorik düşüncenin temel dayanaklarındandır.

TÜBİTAK Bilgisayar Olimpiyatı birinci aşama sınavında karşımıza çıkan pek çok soru, ilk bakışta karmaşık toplam formülleri veya iç içe döngüler gibi görünür. Oysa doğru eşlemeyi kurduğumuzda ya da bir matrisin satır ve sütun toplamlarını karşılaştırdığımızda, sayfalar sürecek cebirsel işlemler tek bir satırda sadeleşir. Bu çerçevede kombinatorik ispatın temel yöntemlerinden ikisine odaklanacağız: **çifte sayım** ve **birebir eşleme**.

## 10.1 Çifte Sayım (Double Counting)

Bir salonda $n$ kişi olduğunu ve bazılarının birbiriyle el sıkıştığını hayal edelim. Toplam el sıkışma sayısını bulmak istiyoruz. Salondaki her kişiye tek tek yaklaşıp kaç kişiyle el sıkıştığını sorsak ve aldığımız tüm yanıtları toplasak, bu toplam gerçek el sıkışma sayısına mı eşit olur?

Hemen küçük ve somut bir örnekle deneyelim. Üç kişi olsun: Ahmet, Berk ve Can. Ahmet; Berk ve Can ile el sıkışmış olsun. Berk ile Can ise birbirleriyle el sıkışmamış olsun.

- Ahmet'in el sıkışma sayısı: 2
- Berk'in el sıkışma sayısı: 1
- Can'ın el sıkışma sayısı: 1

Bu sayıları toplarsak $2 + 1 + 1 = 4$ elde ederiz. Fakat salonda gerçekleşen toplam tokalaşma sayısı sadece 2'dir (Ahmet-Berk ve Ahmet-Can). Aradaki fark nereden kaynaklanıyor?

Cevap açıktır: Her el sıkışma eylemi iki elin birbirine temas etmesini gerektirir; dolayısıyla her bir el sıkışma, o eyleme katılan iki kişi tarafından da birer kez sayılmış, yani tam olarak iki defa hesaba katılmıştır. O halde kişilerin el sıkışma sayılarının toplamı, gerçekleşen toplam el sıkışma sayısının tam 2 katıdır:

$$4 = 2 \times 2$$

Bu gözlem, bizi aynı büyüklüğü iki farklı perspektiften sayarak kurulan bir eşitliğe ulaştırır. Bu yönteme **çifte sayım** (iki yoldan sayma) denir.

### 10.1.1 El Sıkışma Lemması (Handshaking Lemma)

Bu fikri bir graf (graph) modeli üzerinde biçimselleştirelim. Salondaki her insanı bir düğüm (node/köşe), gerçekleşen her el sıkışmayı ise bu iki insanı birleştiren bir kenar (edge) olarak düşünelim. Bir $v$ düğümüne bağlı kenar sayısına o düğümün derecesi denir ve $\deg(v)$ ile gösterilir.

**Teorem 10.1** (El Sıkışma Lemması)**.** *Bir $G = (V, E)$ sonlu grafında, tüm düğümlerin derecelerinin toplamı, toplam kenar sayısının ($|E|$) iki katına eşittir:*

$$\sum_{v \in V} \deg(v) = 2|E|$$

*Kanıt.* Kenarlar ile düğümler arasındaki ilişkiyi saymak için şu ikililerin kümesini tanımlayalım:

$$S = \{(v, e) : v \in V, e \in E \text{ ve köşe } v \text{ kenar } e\text{'nin bir uç noktasıdır}\}$$

$|S|$ büyüklüğünü iki farklı yoldan hesaplayalım:

**1. Yol (Düğümler üzerinden sayma):** Her bir $v \in V$ düğümünü ele alalım. $v$ düğümünün uç noktası olduğu kenarların sayısı doğrudan o düğümün derecesidir ($\deg(v)$). Dolayısıyla tüm düğümler üzerinden toplarsak:

$$|S| = \sum_{v \in V} \deg(v)$$

**2. Yol (Kenarlar üzerinden sayma):** Her bir $e \in E$ kenarını ele alalım. Tanım gereği her kenarın tam olarak 2 uç noktası vardır. Dolayısıyla her $e$ kenarı, $S$ kümesinde tam olarak 2 adet $(v, e)$ ikilisine katkı sağlar. Tüm kenarlar üzerinden toplarsak:

$$|S| = \sum_{e \in E} 2 = 2|E|$$

Her iki yöntem de aynı $|S|$ kümesini saydığı için:

$$\sum_{v \in V} \deg(v) = 2|E|$$

eşitliği elde edilir. □

Bu eşitliğin doğrudan bir sonucu vardır:

**Teorem 10.2.** *Herhangi bir grafta, tek dereceli düğümlerin sayısı daima çifttir.*

*Kanıt.* Düğümleri derecesi tek olanlar ($V_{\text{tek}}$) ve çift olanlar ($V_{\text{çift}}$) olarak iki ayrık kümeye ayıralım:

$$\sum_{v \in V} \deg(v) = \sum_{v \in V_{\text{tek}}} \deg(v) + \sum_{u \in V_{\text{çift}}} \deg(u) = 2|E|$$

Eşitliğin sağ tarafı $2|E|$ çift bir sayıdır. Sol taraftaki ikinci terim, çift sayıların toplamı olduğu için çifttir. Dolayısıyla ilk terim olan $\sum_{v \in V_{\text{tek}}} \deg(v)$ de çift olmak zorundadır. Tek sayıların toplamının çift olabilmesi ancak ve ancak toplanan terim sayısının çift olmasıyla mümkündür. O halde $|V_{\text{tek}}|$ çifttir. □

### 10.1.2 Matrisler ve Görünüm Değiştirme

Çifte sayım yönteminin en yaygın biçimsel modeli, 0 ve 1'lerden oluşan bir matrisin elemanlarını saymaktır. Bir $M$ matrisindeki toplam 1'lerin sayısı iki farklı yolla bulunabilir:

$$\sum_{i} (i.\ \text{satırdaki 1'lerin sayısı}) = \sum_{j} (j.\ \text{sütundaki 1'lerin sayısı})$$

Bu teknik, "satır toplamı= sütun toplamı" prensibi olarak da adlandırılır ve kombinatorik tasarım problemlerinde büyük kolaylık sağlar.

**Örnek 10.3.** *Bir okulda* 15 *öğrenci ve* 15 *kulüp bulunmaktadır. Her kulüpte tam olarak* 4 *öğrenci yer almaktadır. Herhangi iki kulübün en fazla* 1 *ortak öğrencisi olduğu bilindiğine göre, en az kulübe üye olan öğrencinin üye olduğu kulüp sayısı en fazla kaç olabilir?*

*Çözüm.* Öğrenciler ile kulüpler arasındaki üyelik ilişkisini bir matris olarak modelleyelim. Satırlar kulüpleri ($K_1, \ldots, K_{15}$), sütunlar öğrencileri ($O_1, \ldots, O_{15}$) temsil etsin. Eğer $O_j$ öğrencisi $K_i$ kulübüne üye ise matrisin $(i, j)$ hücresine 1, aksi halde 0 yazalım.

Buradaki temel kısıt, herhangi iki kulübün en fazla 1 ortak öğrencisi olmasıdır. Bu kısıt doğrudan şu yapıyı saymaya işaret eder: Bir öğrenci ve onun üye olduğu iki farklı kulüpten oluşan üçlüler. Yani öyle $(O, \{K_a, K_b\})$ üçlülerini sayalım ki, $O$ öğrencisi hem $K_a$ hem de $K_b$ kulübüne üye olsun.

Bu üçlülerin sayısına $T$ diyelim ve iki yoldan hesaplayalım:

**1. Yol (Kulüp çiftleri üzerinden sayma):** Toplam $\binom{15}{2} = \frac{15\times 14}{2} = 105$ farklı kulüp çifti vardır. Soruda verilen kurala göre, herhangi iki kulübün en fazla 1 ortak öğrencisi olabilir. Dolayısıyla her kulüp çifti en fazla 1 öğrenci tarafından paylaşılır. Buradan:

$$T \leq 105 \times 1 = 105$$

**2. Yol (Öğrenciler üzerinden sayma):** $j$. öğrencinin üye olduğu kulüp sayısı $d_j$ olsun. Bu öğrenci, kendi üye olduğu kulüpler arasından $\binom{d_j}{2}$ farklı kulüp çiftinde ortak üyedir. O halde:

$$T = \sum_{j=1}^{15} \binom{d_j}{2}$$

Bu iki hesabı birleştirirsek:

$$\sum_{j=1}^{15} \binom{d_j}{2} \leq 105$$

Toplam üyelik sayısını da çifte sayımla kontrol edelim. Her kulüpte 4 öğrenci olduğuna göre, matristeki toplam 1 sayısı $15 \times 4 = 60$'tır. Dolayısıyla:

$$\sum_{j=1}^{15} d_j = 60$$

15 öğrencinin dereceleri toplamı 60'tır; yani öğrenci başına ortalama $60/15 = 4$ kulüp düşer. Eğer tüm öğrenciler eşit sayıda, yani $d_j = 4$ kulübe üye olsalardı:

$$\sum_{j=1}^{15} \binom{4}{2} = 15 \times 6 = 90 \leq 105$$

olurdu ve bu eşitsizlik sağlanırdı.

Peki en az kulübe üye olan öğrencinin üye olduğu kulüp sayısı 4 olabilir mi? Evet, herkes 4 kulübe üye olursa minimum değer 4 olur. Minimum değer 5 olabilir mi? Eğer herkes en az 5 kulübe üye olsaydı, toplam üyelik sayısı en az $15 \times 5 = 75$ olurdu; oysa toplam üyelik 60'tır. Bu bir çelişkidir.

Dolayısıyla en az kulübe üye olan öğrencinin kulüp sayısı en fazla 4 olabilir. □

**Örnek 10.4.** *$n$ elemanlı bir kümenin alt kümeleri arasından öyle bir $\mathcal{F}$ ailesi seçiliyor ki, bu ailedeki hiçbir küme bir diğerinin alt kümesi değildir (böyle bir aileye Sperner ailesi veya antizincir denir). $\mathcal{F}$ ailesinin eleman sayısı en fazla kaç olabilir?*

*Çözüm.* Bu klasik sonuç Lubell-Yamamoto-Meshalkin (LYM) eşitsizliği ile kanıtlanır. Doğrudan alt kümeleri saymak yerine, $\{1, 2, \ldots, n\}$ kümesinin tüm $n!$ permütasyonlarını ele alalım. Bir $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ permütasyonu verildiğinde, bunun başlangıç öbekleri (önekleri) şunlardır:

$$\emptyset, \{\pi_1\}, \{\pi_1, \pi_2\}, \ldots, \{\pi_1, \ldots, \pi_n\}$$

Dikkat edilirse, bu öneklerin her biri bir öncekinin alt kümesidir; yani tam bir zincir oluştururlar. Dolayısıyla, $\mathcal{F}$ ailesinden bir $A$ kümesi, verilen bir permütasyonun öneki olarak **en fazla bir kez** yer alabilir; çünkü iki farklı küme yer alsaydı biri diğerini kapsamak zorunda kalırdı, bu da antizincir tanımına aykırıdır.

Şimdi şu ikilileri iki yoldan sayalım:

$$S = \{(A, \pi) : A \in \mathcal{F}, \pi \text{ bir permütasyon ve } A \text{ kümesi } \pi\text{'nin bir öneki}\}$$

**1. Yol (Permütasyonlar üzerinden):** Her $\pi$ permütasyonu için, başlangıç önekleri arasında $\mathcal{F}$'ye ait en fazla 1 küme bulunabilir. Toplam $n!$ permütasyon olduğuna göre:

$$|S| \leq n!$$

**2. Yol ($\mathcal{F}$'deki kümeler üzerinden):** Sabit bir $A \in \mathcal{F}$ kümesi alalım. $|A| = k$ olsun. $A$'nın bir permütasyonun ilk $k$ elemanını oluşturması için:

- İlk $k$ pozisyona $A$'nın elemanları kendi içinde $k!$ farklı şekilde dizilmelidir.
- Kalan $n - k$ pozisyona $A$'da olmayan elemanlar $(n - k)!$ farklı şekilde dizilmelidir.

Dolayısıyla her $A$ kümesi tam olarak $k!(n - k)!$ adet permütasyonda önek olarak belirir. O halde:

$$|S| = \sum_{A \in \mathcal{F}} |A|!(n - |A|)!$$

İki hesabı birleştirelim:

$$\sum_{A \in \mathcal{F}} |A|!(n - |A|)! \leq n!$$

Her iki tarafı $n!$'e bölersek:

$$\sum_{A \in \mathcal{F}} \frac{|A|!(n - |A|)!}{n!} \leq 1 \implies \sum_{A \in \mathcal{F}} \frac{1}{\binom{n}{|A|}} \leq 1$$

Bu eşitsizlik **LYM Eşitsizliği** olarak bilinir.

Bölüm 5 ve 6'da incelediğimiz üzere $\binom{n}{k}$ katsayısı, $k = \lfloor n/2 \rfloor$ değerinde maksimuma ulaşır. Dolayısıyla her $A$ için:

$$\frac{1}{\binom{n}{|A|}} \geq \frac{1}{\binom{n}{\lfloor n/2 \rfloor}}$$

Buradan:

$$\sum_{A\in\mathcal{F}} \frac{1}{\binom{n}{\lfloor n/2\rfloor}} \le \sum_{A\in\mathcal{F}} \frac{1}{\binom{n}{|A|}} \le 1$$

$$\frac{|\mathcal{F}|}{\binom{n}{\lfloor n/2\rfloor}} \le 1 \implies |\mathcal{F}| \le \binom{n}{\lfloor n/2\rfloor}$$

elde edilir. Eşitlik durumu, tam olarak $\lfloor n/2\rfloor$ elemanlı tüm alt kümeler seçildiğinde sağlanır. □

### 10.1.3 Algoritmik Açıdan Çifte Sayım

Bilgisayar biliminde bir algoritmanın karmaşıklığını hesaplarken iç içe döngülerin adım sayısını bulmak genellikle bir çifte sayım problemidir.

Aşağıdaki C kod parçasına bakalım:

```
long long toplam = 0;
for (int i = 1; i <= n; i++) {
    for (int j = i; j <= n; j += i) {
        toplam++;
    }
}
```

Bu döngünün kaç adım çalıştığını iki yoldan sayabiliriz:

1. **Dış döngü ($i$) üzerinden:** Her $i$ sayısı için iç döngü $\lfloor n/i\rfloor$ kez döner. Toplam adım sayısı:

   $$\sum_{i=1}^{n} \lfloor \frac{n}{i} \rfloor$$

   Bu toplam harmonik seriden ötürü yaklaşık $n \ln n$, yani $O(n \log n)$ karmaşıklığındadır.

2. **İç döngünün vurduğu ($j$) değerleri üzerinden:** Bir $j$ değeri kaç defa ziyaret edilir? Bölüm 11'deki bölünebilme tanımını hatırlarsak, $j$'nin her pozitif böleni $i$ için bir kez ziyaret edilir. Dolayısıyla $d(j)$, $j$'nin pozitif bölen sayısı olmak üzere:

   $$\text{Toplam Adım} = \sum_{j=1}^{n} d(j)$$

İki yoldan sayarak şu temel özdeşliği doğrudan elde ederiz:

$$\sum_{i=1}^{n} \lfloor \frac{n}{i} \rfloor = \sum_{j=1}^{n} d(j)$$

Böylece 1'den $n$'e kadar olan sayıların bölen sayılarının ortalamasının $\ln n$ civarında olduğunu ileri analize başvurmadan çifte sayımla görmüş oluruz.

```
FUNCTION sumDivisorCounts(n)
    divisorCounts = ARRAY OF SIZE (n + 1) INITIALIZED WITH 0
    FOR i FROM 1 TO n DO
        FOR j FROM i TO n STEP i DO
            divisorCounts[j] = divisorCounts[j] + 1
        END FOR
    END FOR
    RETURN SUM(divisorCounts)
END FUNCTION

FUNCTION sumFloorDivisions(n)
    total = 0
    FOR i FROM 1 TO n DO
        total = total + FLOOR(n / i)
    END FOR
    RETURN total
END FUNCTION

n = 1000
ASSERT sumDivisorCounts(n) EQUALS sumFloorDivisions(n)
```

**En Sık Yapılan Hata (Çifte Sayım):** Bir nesneyi iki yoldan sayarken, sayılan ikililerin kümesini ($S$) kesin ve net bir biçimde tanımlamadan işe başlamaktır. Hangi elemanın hangi kurala göre sayıldığı netleştirilmezse, bazı elemanlar mükerrer sayılabilir veya tamamen gözden kaçabilir.

## 10.2 Birebir Eşleme (Bijection)

Bir sepet elma ile bir sepet armudun sayılarının eşit olup olmadığını anlamanın en ilkel yolu nedir? İki sepetteki meyveleri de teker teker sayıp sayıları karşılaştırabilirsiniz. Peki ya saymayı bilmiyorsanız?

Daha doğrudan bir yaklaşım vardır: Sepetten bir elma alıp diğer sepetten bir armutla eşleştirerek masaya koyarsınız. Bu işlemi meyveler bitene kadar tekrarlarsınız. Masada açıkta tek bir meyve kalmadığında, başlangıçta her iki sepette de aynı sayıda meyve olduğunu anlarsınız.

Kombinatorikte bu yaklaşıma **birebir eşleme** (bijection) denir. Eleman sayısını belirlemek istediğimiz zor bir küme yerine eleman sayısı bilinen denk bir küme seçer ve iki küme arasında birebir ve örten bir dönüşüm kurarız.

**Tanım 10.5** (Birebir ve Örten Fonksiyon)**.** *$A$ ve $B$ iki sonlu küme olsun. Bir $f : A \to B$ fonksiyonu:*

1. ***Birebir (injective)** ise: Her $x_1, x_2 \in A$ için $f(x_1) = f(x_2) \implies x_1 = x_2$ olmalıdır. (Farklı elemanlar farklı yerlere gider.)*

2. ***Örten (surjective)*** *ise: Her $y \in B$ için $f(x) = y$ olacak şekilde en az bir $x \in A$ bulunmalıdır. (Açıkta eleman kalmaz.)*

*Hem birebir hem de örten olan bir fonksiyona* ***birebir eşleme*** *(bijection) denir. Eğer $A$ ile $B$ arasında bir birebir eşleme varsa:*

$$|A| = |B|$$

Birebir eşleme kurarken izlenmesi gereken sistematik adımlar şunlardır:

1. Zor küme $A$'yı ve bilinen/kolay küme $B$'yi belirle.
2. Her $a \in A$ elemanını bir $b \in B$ elemanına götüren açık bir $f(a)$ kuralı yaz.
3. Ters kuralı göster: Her $b \in B$ elemanını geriye tek bir $a \in A$ elemanına götüren $f^{-1}(b)$ kuralını tanımla.
4. Ters fonksiyonun varlığı ($f^{-1}(f(a)) = a$ ve $f(f^{-1}(b)) = b$), fonksiyonun otomatikman hem birebir hem örten olduğunu kanıtlar.

### 10.2.1 Somut Örnek: Parite Eşlemesi

$n \geq 1$ elemanlı bir $X = \{x_1, x_2, \ldots, x_n\}$ kümesini düşünelim. Bu kümenin çift sayıda elemana sahip alt kümelerinin sayısı ile tek sayıda elemana sahip alt kümelerinin sayısı birbirine eşit midir?

Hemen $n = 3$ için $X = \{1, 2, 3\}$ ile deneyelim:

- Çift boyutlu alt kümeler ($A_{\text{çift}}$): $\emptyset$ (0 elemanlı), $\{1,2\}$, $\{1,3\}$, $\{2,3\}$ $\implies$ Toplam 4 tane.
- Tek boyutlu alt kümeler ($A_{\text{tek}}$): $\{1\}$, $\{2\}$, $\{3\}$, $\{1,2,3\}$ $\implies$ Toplam 4 tane.

Sayılar eşit çıktı ($4 = 4$). Genel durumda bunu nasıl ispatlarız?

**Teorem 10.6.** *$n \geq 1$ elemanlı bir kümenin çift sayıda elemana sahip alt kümelerinin sayısı, tek sayıda elemana sahip alt kümelerinin sayısına eşittir:*

$$\sum_{k \text{ çift}} \binom{n}{k} = \sum_{k \text{ tek}} \binom{n}{k} = 2^{n-1}$$

*Kanıt.* Çift boyutlu bir alt kümeyi tek boyutlu bir alt kümeye dönüştürmenin doğrudan bir yolu, kümeye sabit bir elemanı eklemek ya da kümeden çıkarmaktır. Bölüm 17'de ele alacağımız parite mantığıyla, bu işlem kümenin eleman sayısını $\pm 1$ değiştirir ve paritesini tersine çevirir.

Kümenin özel bir elemanını sabitleyelim; örneğin $x_1$ elemanını seçelim. $A_{\text{çift}}$ kümesinden $A_{\text{tek}}$ kümesine bir $f$ fonksiyonu tanımlayalım:

$$f(S) = \begin{cases} S \cup \{x_1\}, & \text{eğer } x_1 \notin S \\ S \setminus \{x_1\}, & \text{eğer } x_1 \in S \end{cases}$$

Bu kuralı inceleyelim:

- Eğer $|S|$ çift ise, $x_1$'i eklersek ya da çıkarırsak yeni kümenin boyutu $|S| \pm 1$ olur, yani tek sayıya dönüşür. Dolayısıyla $f(S) \in A_{\text{tek}}$ olur.
- Bu işlemin tersi tam olarak kendisidir: $f(f(S)) = S$. Çünkü $x_1$ yoksa ekledik, bir daha uygularsak çıkardık ve eski haline döndük.

Bir fonksiyon kendi kendisinin tersi ise ($f \circ f = I$), o fonksiyon zorunlu olarak birebir ve örtendir (involüsyon). Dolayısıyla $|A_{\text{çift}}| = |A_{\text{tek}}|$ olmak zorundadır. Toplam alt küme sayısı $2^n$ olduğuna göre, her birinin sayısı $2^n/2 = 2^{n-1}$ olur. □

### 10.2.2 Kafes Yolları (Grid Paths) ve Bloklama Eşlemesi

Bölüm 9'da incelediğimiz kafes yolları, kombinatorikte birebir eşlemenin en sık kullanıldığı alanlardandır. $(0,0)$ noktasından $(m,n)$ noktasına sadece sağa ($R : (x+1, y)$) ve yukarı ($U : (x, y+1)$) adımlarla kaç farklı yoldan gidilebilir?

Toplamda $m$ defa sağa ve $n$ defa yukarı adım atmak zorundayız. Toplam adım sayısı $m+n$'dir. Bu adımlardan hangilerinin sağa olacağını seçmek $\binom{m+n}{m}$ farklı yolla yapılabilir. Burada kurulan eşleme şudur: Geçerli her yol, $\{R, U\}$ harflerinden oluşan, $m$ tane $R$ ve $n$ tane $U$ içeren bir dizilime birebir karşılık gelir.

**Örnek 10.7** (André'nin Yansıma İlkesi (Désiré André, 1887) / Catalan Sayıları Sezgisi)**.** $(0,0)$ *noktasından* $(n,n)$ *noktasına giden ve* $y = x$ *doğrusunun kesinlikle üzerine çıkmayan (yani her an* $y \leq x$ *olan) kafes yollarının sayısını bulunuz.*

*Çözüm.* İlk hamle olarak kuralı ihlal eden durumları belirleyelim. Kısıtlamasız tüm yolların sayısı $\binom{2n}{n}$'dir. Kuralı ihlal eden "kötü" yolları çıkarırsak sonuca ulaşabiliriz. Kötü bir yol, yolculuğun bir anında $y = x+1$ doğrusuna değen yoldur.

Bir kötü yol düşünelim. Bu yol $y = x+1$ doğrusuna ilk kez bir $P$ noktasında çarpmış olsun. $P$ noktasından sonraki yol parçasını $y = x+1$ doğrusuna göre **yansıtalım** (yani sonraki tüm $R$ adımlarını $U$, tüm $U$ adımlarını $R$ yapalım).

- Başlangıçta yol $(n,n)$ noktasında bitiyordu ($n$ tane $R$, $n$ tane $U$).
- Çarpma anı $P$'ye kadar yol $k$ tane $R$ ve $k+1$ tane $U$ atmıştır (çünkü $y = x+1$).
- $P$'den sonra kalan adımlar: $n-k$ tane $R$ ve $n-(k+1)$ tane $U$.

- Bu kalan kısmı tersine çevirirsek ($R \leftrightarrow U$), yeni eklenen adımlar $n-(k+1)$ tane $R$ ve $n-k$ tane $U$ olur.

- Yansıyan yeni yolun ulaştığı son nokta:

$$x_{\text{son}} = k + (n-k-1) = n-1$$

$$y_{\text{son}} = (k+1) + (n-k) = n+1$$

Görüldüğü gibi, $y = x+1$ doğrusuna en az bir kez çarpan her yol, yansıtıldığında $(0,0)$'dan $(n-1, n+1)$ noktasına giden bir yola dönüşür.

Bu eşleme tersine de kurulabilir: $(0,0)$'dan $(n-1, n+1)$ noktasına giden her yol, başlangıçta $y \leq x$ iken sonunda $y > x$ olduğu için $y = x+1$ doğrusunu kesmek **zorundadır**. İlk kestiği noktadan sonrasını yansıtırsak orijinal $(n,n)$'e giden kötü yolu geri elde ederiz.

Dolayısıyla:

$$|\text{Kötü Yollar}| = \binom{(n-1)+(n+1)}{n-1} = \binom{2n}{n-1}$$

Geçerli yolların sayısı ise:

$$C_n = \binom{2n}{n} - \binom{2n}{n-1} = \frac{1}{n+1}\binom{2n}{n}$$

Bu sayılara **Catalan Sayıları** denir. Birebir eşleme sayesinde geometrik bir yansıma problemi doğrudan çözüme kavuşturulmuştur. □

**En Sık Yapılan Hata (Birebir Eşleme):** Aday bir $f : A \to B$ dönüşümü kurgulayıp sadece birebir olduğunu sezmek ama $f$'nin hedef kümedeki **her** elemanı kapsadığını (örtenlik) ya da $f(A) \subseteq B$ olduğunu (yani dönüşümün hedef kümenin sınırları içinde kaldığını) kontrol etmeyi unutmaktır. En sağlam kontrol, ters fonksiyon $f^{-1}$'i açıkça yazıp çalıştığını göstermektir.

## 10.3 Kombinatorik İspat Örnekleri

Bir cebirsel özdeşliği kanıtlamanın iki yolu vardır: Formülleri açıp terimleri sadeleştirmek ya da ifadenin iki tarafının da aslında aynı büyüklüğü saydığını göstermek. İkinci yaklaşıma **kombinatorik ispat** denir. Kombinatorik ispatlar ifadenin doğruluğunu göstermekle kalmaz, arkasındaki yapısal nedeni de açıklar.

### 10.3.1 Pascal Özdeşliğine İkinci İspat

Pascal kuralını cebirsel olarak faktöriyellerle ($n!$) ispatlamayı Bölüm 5'ten biliyoruz. Şimdi bu kuralı doğrudan bir kombinatorik sayımla gösterelim.

**Teorem 10.8** (Pascal Özdeşliği)**.** *$1 \leq k \leq n$ tam sayıları için:*

$$\binom{n+1}{k} = \binom{n}{k} + \binom{n}{k-1}$$

*Kanıt.* **Soru:** $n+1$ kişilik bir gruptan $k$ kişilik bir olimpiyat takımı kaç farklı şekilde seçilebilir?

**1. Cevap (Doğrudan Seçim):** $n+1$ elemanlı bir kümeden sırasız olarak $k$ eleman seçiyoruz. Tanım gereği bu seçim:

$$\binom{n+1}{k}$$

farklı şekilde yapılabilir.

**2. Cevap (Özel Bir Kişiye Göre Durum Analizi):** Gruptaki öğrencilerden birini belirleyelim; adı Ayşe olsun. Kurulacak $k$ kişilik takımda Ayşe ya vardır ya da yoktur. Bu iki durum birbirini dışlar:

- **1. Durum (Ayşe takımda yok):** Takımın tamamı Ayşe dışındaki kalan $n$ kişi arasından seçilmelidir. $n$ kişiden $k$ kişi seçileceği için $\binom{n}{k}$ farklı yol vardır.
- **2. Durum (Ayşe takımda var):** Ayşe'nin yeri kesinleştiği için takımın kalan $k-1$ üyesi, gruptaki diğer $n$ kişi arasından seçilmelidir. Bu seçim $\binom{n}{k-1}$ farklı yolla yapılabilir.

Toplama ilkesi gereği toplam takım sayısı bu iki durumun toplamıdır:

$$\binom{n}{k} + \binom{n}{k-1}$$

Her iki yöntem de aynı takım sayısını bulduğundan:

$$\binom{n+1}{k} = \binom{n}{k} + \binom{n}{k-1}$$

özdeşliği kanıtlanmış olur. □

### 10.3.2 Alt Küme Sayısı Özdeşlikleri

**Teorem 10.9.** *Her $n \geq 0$ tam sayısı için:*

$$\sum_{k=0}^{n} \binom{n}{k} = 2^n$$

*Kanıt.* $n$ elemanlı bir $S = \{a_1, a_2, \ldots, a_n\}$ kümesinin tüm alt kümelerinin sayısını iki yoldan hesaplayalım.

**1. Yol (Kümeleri boyutlarına göre ayırarak):** Bir alt kümenin eleman sayısı $k$, 0 ile $n$ arasında herhangi bir değer alabilir. $k$ elemanlı alt kümelerin sayısı $\binom{n}{k}$'dır. Tüm olası boyutları toplarsak:

$$\text{Toplam Alt Küme Sayısı} = \sum_{k=0}^{n} \binom{n}{k}$$

**2. Yol (Elemanların üyelik kararları üzerinden):** Bir alt küme oluşturmak için her $a_i \in S$ elemanı için bağımsız bir karar veririz:

$$a_i \text{ bu alt kümede var mı, yok mu?}$$

Her eleman için 2 bağımsız seçenek (VAR ya da YOK) vardır. Çarpma ilkesi gereği $n$ eleman için toplam seçenek sayısı:

$$\underbrace{2 \times 2 \times \cdots \times 2}_{n \text{ tane}} = 2^n$$

İki sayım da aynı nesneleri saydığından eşitlik sağlanır. □

**Örnek 10.10** (Komite ve Başkan Seçimi)**.** *$n \geq 1$ için aşağıdaki özdeşliği kombinatorik bir model kurarak ispatlayınız:*

$$\sum_{k=1}^{n} k\binom{n}{k} = n2^{n-1}$$

*Çözüm.* Sol taraftaki $k\binom{n}{k}$ teriminin yapısına bakalım: Önce $n$ kişiden $k$ kişilik bir komite seçilmiş ($\binom{n}{k}$), ardından seçilen bu $k$ kişi arasından 1 başkan seçilmiştir ($k$). Dolayısıyla ifadenin tamamı, herhangi bir büyüklükte bir komite ve bu komitenin içinden bir başkan seçme işlemini temsil eder.

Şimdi bu komite ve başkan çiftini iki yoldan seçelim:

**1. Yol (Önce komiteyi, sonra başkanı seç):** Komitenin boyutu $k$ ($1 \leq k \leq n$) olsun.

- $k$ kişilik komite $\binom{n}{k}$ yolla seçilir.
- Seçilen komiteye başkan $k$ farklı yoldan belirlenir.

Komite boyutu $1, 2, \ldots, n$ olabileceği için tüm durumların toplamı:

$$\sum_{k=1}^{n} k\binom{n}{k}$$

**2. Yol (Önce başkanı, sonra komitenin kalanını seç):** Sıralamayı değiştirelim:

- $n$ kişi arasından komite başkanını seçelim: $n$ farklı seçenek vardır.
- Kalan $n-1$ kişinin her birine soralım: "Bu komiteye girmek istiyor musun?"
- Her birinin 2 seçeneği vardır (evet/hayır). Bu da $2^{n-1}$ farklı komite oluşumu demektir.

Çarpma ilkesi gereği toplam seçim sayısı:

$$n \times 2^{n-1}$$

Her iki yöntem de (komite, başkan) çiftlerini saydığı için:

$$\sum_{k=1}^{n} k\binom{n}{k} = n2^{n-1}$$

□

**Örnek 10.11** (Vandermonde Özdeşliği)**.** *$m, n, k \geq 0$ tam sayıları için:*

$$\sum_{i=0}^{k} \binom{m}{i}\binom{n}{k-i} = \binom{m+n}{k}$$

*özdeşliğini kombinatorik olarak kanıtlayınız.*

*Çözüm.* Sağ taraftaki $\binom{m+n}{k}$ ifadesi, toplam $m+n$ kişilik bir gruptan $k$ kişinin seçilmesini anlatır. Sol tarafta ise grup iki parçaya ayrılmış durumdadır. Bu durumu somutlaştıralım:

Bir sınıfta $m$ kız ve $n$ erkek öğrenci olsun. Bu sınıftan $k$ kişilik bir yarışma takımı seçmek istiyoruz.

**1. Yol (Doğrudan seçim):** Toplam $m+n$ öğrenciden $k$ öğrenci seçileceği için toplam durum sayısı:

$$\binom{m+n}{k}$$

**2. Yol (Kız ve erkek sayılarına göre parçalama):** Seçilen $k$ kişilik takımda $i$ tane kız öğrenci ($0 \leq i \leq k$) bulunsun. O halde takımın geri kalan $k-i$ kişisi erkek olmak zorundadır.

- $m$ kız arasından $i$ kız $\binom{m}{i}$ yolla seçilir.
- $n$ erkek arasından $k-i$ erkek $\binom{n}{k-i}$ yolla seçilir.

Çarpma kuralıyla bu bileşim $\binom{m}{i}\binom{n}{k-i}$ yolla oluşturulur. $i$'nin tüm olası $0, 1, \ldots, k$ değerlerini toplarsak:

$$\sum_{i=0}^{k} \binom{m}{i}\binom{n}{k-i}$$

Her iki yol da aynı takımları saydığından eşitlik kanıtlanır. □

### 10.3.3 Ağaç Sayma ve Prüfer Kodlaması: Cayley Formülü

Bilgisayar olimpiyatlarında ve ileride Bölüm 28'de göreceğimiz graf teorisinde ağaç (tree) yapıları önemli bir yer tutar. $n$ etiketli düğüm üzerine çizilebilecek farklı ağaçların sayısını veren formül $n^{n-2}$'dir (Cayley Formülü). Bu formülün Prüfer dizilimi ile ispatı, birebir eşlemenin kombinatorikteki etkili uygulamalarındandır.

**Teorem 10.12** (Cayley Formülü)**.** $\{1, 2, \ldots, n\}$ *etiketli düğümlere sahip farklı ağaçların sayısı* $n^{n-2}$*'dir* $(n \geq 2)$*.*

*İspat Taslağı (Prüfer Eşlemesi).* Bir $T$ ağacını, elemanları $\{1, 2, \ldots, n\}$ kümesinden seçilen $n-2$ uzunluğundaki bir $(p_1, p_2, \ldots, p_{n-2})$ dizisiyle birebir eşleyebiliriz.

**Ağaçtan Diziye ($f : T \to P$):**

1. Ağaçtaki derecesi 1 olan düğümleri (yaprakları) bul.
2. Etiketi en küçük olan yaprağı seç. Bu yaprağın tek komşusunun etiketini diziye yaz.
3. Bu yaprağı ve ona bağlı kenarı ağaçtan sil.
4. Ağaçta sadece 2 düğüm kalana kadar bu işlemi tekrarla. Toplam $n-2$ adımda $n-2$ elemanlı bir dizi oluşur.

**Diziden Ağaca ($f^{-1} : P \to T$):** Her adımda dizide görünmeyen en küçük etiket, o adımda silinen yaprağın ta kendisidir. Dolayısıyla adımları geriye doğru takip ederek her yaprağın hangi düğüme bağlanması gerektiği tek şekilde belirlenir.

Bu işlem $n$ düğümlü her ağacı, $\{1, 2, \ldots, n\}$ elemanlarından oluşan $n-2$ uzunluğundaki dizilerle birebir eşler. Böyle bir dizinin her pozisyonu için $n$ bağımsız seçenek olduğundan, toplam dizi sayısı:

$$\underbrace{n \times n \times \cdots \times n}_{n-2 \text{ tane}} = n^{n-2}$$

olur. Eşleme birebir ve örten olduğundan ağaç sayısı da $n^{n-2}$'dir. □

### 10.3.4 Hokey Sopası Özdeşliği (Hockey-Stick Identity)

**Teorem 10.13** (Hokey Sopası Özdeşliği)**.** $n \geq k$ *pozitif tam sayıları için:*

$$\sum_{i=k}^{n} \binom{i}{k} = \binom{n+1}{k+1}$$

*Kanıt.* Sağ taraftaki $\binom{n+1}{k+1}$ ifadesi, $\{1, 2, \ldots, n+1\}$ kümesinden $k+1$ elemanlı bir alt küme seçmeyi temsil eder. Sol taraftaki toplam ise bir parametreye göre durum ayrımı yapıldığını gösterir. Buradaki doğal ayrım ölçütü, alt kümenin en büyük elemanıdır.

$\{1, 2, \ldots, n+1\}$ kümesinden $k+1$ elemanlı bir $A$ alt kümesi seçelim. $A$ kümesinin en büyük elemanına $M$ diyelim. $A$'da $k+1$ eleman olduğu için en büyük eleman $M$ en az $k+1$ olabilir. En fazla ise $n+1$ olabilir. O halde $M = i+1$ olsun, burada $k \leq i \leq n$.

Eğer en büyük eleman $i+1$ olarak sabitlenirse:

- $i+1$'den büyük hiçbir eleman seçilemez.
- $A$'nın kalan $k$ elemanı, $i+1$'den küçük olan $\{1, 2, \ldots, i\}$ kümesinden seçilmelidir.
- $i$ eleman arasından $k$ eleman $\binom{i}{k}$ yolla seçilir.

Olası tüm en büyük eleman durumlarını ($i = k, k+1, \ldots, n$) toplarsak:

$$\sum_{i=k}^{n} \binom{i}{k} = \binom{n+1}{k+1}$$

özdeşliği doğrudan elde edilir. Pascal üçgeninde bu toplama bakıldığında bir köşegenden başlayıp aniden yön değiştiren bir çizgi görülür; bu şekil bir hokey sopasına benzediği için özdeşlik bu adla anılır. □

### 10.3.5 Özet ve Problem Çözme Rehberi

Kombinatorik bir eşitlik veya sayım problemiyle karşılaşıldığında şu adımlar izlenebilir:

1. **Cebirsel açılımlara takılmayın:** Faktöriyel sadeleştirmeleri uzadığında durup "Burada somut olarak ne seçiliyor?" sorusunu sorun.
2. **İlişki matrisi kurun:** İki farklı küme arasındaki ilişkiler söz konusuysa bir $\{0, 1\}$ matrisi düşünün. Satır toplamları ile sütun toplamlarını eşitlemek (çifte sayım) denklemi belirginleştirir.
3. **Uç elemanı sabitleyin:** Alt küme seçimlerinde "en büyük eleman", "özel bir üye" veya "ilk kural ihlali anı" gibi bir referans noktası belirlemek, toplamları bağımsız parçalara ayırmanın etkili bir yoludur.
4. **Ters fonksiyon testi:** Birebir eşleme kurarken $f^{-1}$ dönüşümünü somut olarak gösterin. Fonksiyonun hedef kümede açıkta eleman bırakıp bırakmadığını küçük değerlerle ($n = 1, 2$) denetleyin.

# Bölüm 11

# Bölünebilme

Bilgisayar biliminde ve algoritma tasarımında tam sayıların birbiriyle kurduğu ilişkiler temel bir yer tutar. Bir döngünün kaç adımda bir tetikleneceğini belirlemekten karma (hash) tablolarına, asallık testlerinden kriptografik anahtar üretimine kadar pek çok alanda tam sayıların katları ve kalanlarıyla karşılaşırız.

Bu bölümde tam sayılar kümesinde bölünebilme bağıntısını, basamak çözümlemelerine dayanan pratik bölünebilme kurallarını ve aritmetiğin temel taşlarından biri olan Bölme Algoritması'nı algoritmik yansımalarıyla ele alacağız.

## 11.1 Bölünebilme ve Gösterim

### 11.1.1 Motivasyon: Hangi Sorunu Çözer?

Elimizde 24 adet özdeş bellek hücresi olduğunu ve bunları eşit büyüklükte bloklara ayırmak istediğimizi varsayalım. Blok boyutunu 3 seçersek $24 = 3 \times 8$ yazabiliriz; hiçbir hücre açıkta kalmaz. Blok boyutunu 4 seçersek $24 = 4 \times 6$ olur; yine açıkta hücre kalmaz. Fakat blok boyutunu 5 seçtiğimizde $24 = 5 \times 4 + 4$ elde ederiz ve 4 hücre tam bir blok oluşturamaz.

İlk iki durumda 24'ün 3'e ve 4'e *tam böründüğünü*, son durumda ise 5'e *tam bölünmediğini* söyleriz. Gerçek dünyada disk bloklarının sektörlere yerleştirilmesi, bellek hizalaması (alignment) veya paralel işlemciler arasında iş yükünün paylaştırılması gibi senaryolar, bir tam sayının diğerine kalansız bölünüp bölünmediği sorusuna dayanır. Bu ilişkiyi cebirsel işlemlerle yürütülebilir biçimde tanımlamak gerekir.

### 11.1.2 Tanım ve Temel Gösterim

**Tanım 11.1** (Bölünebilme)**.** *$a$ ve $b$ iki tam sayı olsun. Eğer $b = a \cdot k$ eşitliğini sağlayan en az bir $k \in \mathbb{Z}$ tam sayısı varsa, $a$,* ***$b$'yi böler*** *(veya $b$,* ***$a$'nın bir***

***katıdır**) denir ve bu durum*

$$a \mid b$$

*biçiminde gösterilir. Eğer böyle bir $k$ tam sayısı yoksa, $a$, $b$**'yi bölmez** denir ve $a \nmid b$ şeklinde yazılır.*

Burada $a$'ya $b$'nin bir *böleni* (veya çarpanı), $b$'ye ise $a$'nın bir *katı* denir.

Somut örneklere bakalım:

- $3 \mid 24$ çünkü $24 = 3 \times 8$ ve $8 \in \mathbb{Z}$'dir.
- $-4 \mid 20$ çünkü $20 = (-4) \times (-5)$ ve $-5 \in \mathbb{Z}$'dir.
- $7 \mid -35$ çünkü $-35 = 7 \times (-5)$ ve $-5 \in \mathbb{Z}$'dir.
- $5 \nmid 24$ çünkü $24 = 5k$ eşitliğini sağlayan bir $k$ tam sayısı yoktur ($k = 4.8 \notin \mathbb{Z}$).

**En Sık Yapılan Hata: Bölme İşareti ile Bölünebilme Çizgisini Karıştırmak**
Düz çizgi ($|$) ile bölü işareti ($/$) veya kesir çizgisi sıkça karıştırılır:

- $a/b$ veya $\frac{a}{b}$ bir **sayıdır** (örneğin $6/2 = 3$ bir sayıdır).
- $a \mid b$ ise doğru ya da yanlış değeri alabilen bir **önermedir** (bağıntı bildirir). $2 \mid 6$ ifadesi "2, 6'yı böler" anlamına gelen *doğru* bir önermedir.
- Terimlerin sırası da farklıdır: $a \mid b$ yazıldığında bölen *solda*, bölünen *sağdadır*. Kesirde ise pay üstte (veya solda), payda altta (veya sağdadır): $\frac{b}{a}$. Dolayısıyla $3 \mid 12$ doğru bir önermeyken $12 \mid 3$ yanlıştır.

### Uç Değerler: Sıfır ve Bir

Bölünebilme tanımını sıfır ve bir için inceleyelim:

1. **Her $a \in \mathbb{Z}$ için $1 \mid a$ ve $-1 \mid a$**: Çünkü $a = 1 \cdot a$ ve $a = (-1) \cdot (-a)$ her zaman yazılabilir.
2. **Her $a \in \mathbb{Z}$ için $a \mid 0$**: Çünkü $0 = a \cdot 0$ olup $0 \in \mathbb{Z}$'dir. Yani her tam sayı sıfırı böler.
3. **0 kimi böler?**: Tanıma göre $0 \mid b$ olması için $b = 0 \cdot k$ olmalıdır. Sağ taraf her zaman 0 olduğundan, bu eşitlik yalnızca $b = 0$ iken sağlanır. Dolayısıyla $0 \mid 0$ teknik olarak doğrudur ($0 = 0 \cdot k$ her $k$ için geçerlidir), fakat $b \neq 0$ ise $0 \nmid b$. Sıfıra bölme işlemi aritmetikte tanımsız olsa da, çarpan ilişkisi üzerinden tanımlanan $0 \mid 0$ ifadesi bu biçimsel kurala uyar; ancak belirsizlik yaratmamak adına aksi belirtilmedikçe bölenin sıfırdan farklı olduğu varsayılır.

### 11.1.3 Bölünebilmenin Temel Özellikleri

Bölünebilme bağıntısı, cebirsel sadeleştirmelerde sıkça başvurulan doğrusal özelliklere sahiptir.

**Teorem 11.2** (Bölünebilmenin Cebirsel Özellikleri)**.** *$a, b, c, d \in \mathbb{Z}$ olsun. Aşağıdaki özellikler geçerlidir:*

1. ***Yansıma (Reflexivity):*** *Her $a$ için $a \mid a$.*
2. ***Geçişme (Transitivity):*** *$a \mid b$ ve $b \mid c$ ise $a \mid c$.*
3. ***Çarpımsal Büyüme:*** *$a \mid b$ ise her $m \in \mathbb{Z}$ için $a \mid bm$.*
4. ***Doğrusal Kombinasyon:*** *$a \mid b$ ve $a \mid c$ ise, her $x, y \in \mathbb{Z}$ için*

   $$a \mid (bx + cy)$$

   *olur. Özel olarak $a \mid (b + c)$ ve $a \mid (b - c)$ sağlanır.*
5. ***Çarpım Kuralı:*** *$a \mid b$ ve $c \mid d$ ise $ac \mid bd$.*
6. ***Sınırlandırma İlkesi:*** *$b \neq 0$ ve $a \mid b$ ise $|a| \leq |b|$.*

*Kanıt.* Özellikleri tanım üzerinden doğrulayalım:

- *Geçişme:* $a \mid b \implies b = ak_1$ ve $b \mid c \implies c = bk_2$ $(k_1, k_2 \in \mathbb{Z})$. O halde $c = (ak_1)k_2 = a(k_1k_2)$. $k_1k_2 \in \mathbb{Z}$ olduğundan $a \mid c$.
- *Doğrusal Kombinasyon:* $b = ak_1$ ve $c = ak_2$ ise,

  $$bx + cy = (ak_1)x + (ak_2)y = a(k_1x + k_2y).$$

  $x, y, k_1, k_2$ tam sayı olduğundan $k_1x + k_2y$ de bir tam sayıdır. Tanım gereği $a \mid (bx + cy)$.
- *Sınırlandırma:* $b = ak$ ve $b \neq 0$ ise $k \neq 0$'dır. $k$ bir tam sayı olduğundan $|k| \geq 1$'dir. Eşitliğin her iki tarafının mutlak değerini alırsak:

  $$|b| = |a| \cdot |k| \geq |a| \cdot 1 = |a|.$$

□

Sınırlandırma ilkesi, sonsuz bir arama uzayını sonlu bir aralığa indirmek için en etkili araçlardan biridir. Bir $a$ bilinmeyeninin sabit bir $b$ sayısını böldüğü belirlendiğinde, $a$'nın alabileceği değerler $b$'nin sonlu sayıdaki bölenleriyle kısıtlanır.

### 11.1.4 Çözümlü Örnekler

**Örnek 11.3.** *Her $n$ tam sayısı için $2 \mid (n^2+n)$ olduğunu gösteriniz.*

*Çözüm.* İfadeyi çarpanlarına ayırırsak $n^2+n = n(n+1)$ elde ederiz. Bölüm 17'de ayrıntılı inceleyeceğimiz parite (teklik-çiftlik) durumlarını ele alalım:

- **Durum 1:** $n$ çift olsun. Bir $k \in \mathbb{Z}$ için $n = 2k$ yazılabilir.

  $$n(n+1) = 2k(2k+1) = 2 \cdot [k(2k+1)]$$

  $k(2k+1)$ bir tam sayı olduğundan $2 \mid n(n+1)$'dir.

- **Durum 2:** $n$ tek olsun. Bir $k \in \mathbb{Z}$ için $n = 2k+1$ yazılabilir.

  $$n(n+1) = (2k+1)(2k+2) = (2k+1) \cdot 2(k+1) = 2 \cdot [(2k+1)(k+1)]$$

  Burada da çarpım 2'nin tam katıdır.

Her iki durumda da $2 \mid n(n+1)$ elde edilir. □

**Örnek 11.4.** *$\frac{n+3}{n-1}$ kesrini bir tam sayı yapan tüm $n \in \mathbb{Z}$ değerlerini bulunuz.*

*Çözüm.* Kesrin tam sayı olması, paydanın payı kalansız bölmesini gerektirir: $(n-1) \mid (n+3)$. Doğrusal kombinasyon özelliğinden yararlanarak bölünen kısımdaki $n$ değişkenini eleyebiliriz.

$(n-1) \mid (n-1)$ olduğu açıktır. İki terimin farkını alırsak:

$$(n-1) \mid \Big((n+3) - 1 \cdot (n-1)\Big) \implies (n-1) \mid 4.$$

Böylece değişken elenmiş ve $n-1$'in 4'ün bir böleni olması gerektiği sonucuna varılmıştır. 4'ün tam sayı bölenleri $\pm 1, \pm 2, \pm 4$'tür:

$$\begin{aligned} n-1 = 1 &\implies n = 2 \\ n-1 = -1 &\implies n = 0 \\ n-1 = 2 &\implies n = 3 \\ n-1 = -2 &\implies n = -1 \\ n-1 = 4 &\implies n = 5 \\ n-1 = -4 &\implies n = -3 \end{aligned}$$

Çözüm kümesi $n \in \{-3, -1, 0, 2, 3, 5\}$ olarak bulunur. □

**Örnek 11.5.** *$a, b \in \mathbb{Z}$ olmak üzere, $13 \mid (3a+4b)$ olduğu veriliyor. $13 \mid (7a+5b)$ olduğunu kanıtlayınız.*

*Çözüm.* $A = 3a + 4b$ ve $B = 7a + 5b$ diyelim. $13 \mid A$ verildiğinde $13 \mid B$ olduğunu göstermek için $B$'yi $A$ ve 13'ün katları cinsinden ifade etmeyi deneriz.

$A$'yı 2 ile çarpıp $B$ ile toplayalım:

$$2A + B = 2(3a + 4b) + (7a + 5b) = (6a + 8b) + (7a + 5b) = 13a + 13b = 13(a + b).$$

Buradan $B$ çekildiğinde:

$$B = 13(a + b) - 2A$$

elde edilir. Hipotezden $13 \mid A$ olduğundan $13 \mid 2A$'dır. Doğal olarak $13 \mid 13(a + b)$ de geçerlidir. Doğrusal kombinasyon özelliği gereğince 13 bu iki terimin farkını da böler:

$$13 \mid \Big(13(a + b) - 2A\Big) \implies 13 \mid B \implies 13 \mid (7a + 5b).$$

□

**Örnek 11.6.** *$n$ pozitif bir tam sayı olmak üzere, $(2n+1) \mid (n^2+5)$ şartını sağlayan tüm $n$ değerlerini bulunuz.*

*Çözüm.* Bölen doğrusal, bölünen ise kareseldir. Bölünen terimden $n$'i aşama aşama yok etmek için ifadeyi 2 ile genişleterek $2n$ katsayısını yakalayalım.

$(2n + 1) \mid (n^2 + 5)$ ise çarpımsal büyüme gereğince:

$$(2n + 1) \mid 2(n^2 + 5) \implies (2n + 1) \mid (2n^2 + 10).$$

$2n^2$ terimini yok etmek için bu ifadeden $n(2n + 1)$ çıkaralım:

$$(2n + 1) \mid \Big((2n^2 + 10) - n(2n + 1)\Big) \implies (2n + 1) \mid (-n + 10).$$

Kalan terimde $n$ katsayısını 2 yapmak için tekrar 2 ile çarpalım:

$$(2n + 1) \mid 2(-n + 10) \implies (2n + 1) \mid (-2n + 20).$$

Bu ifadeye $(2n + 1)$ eklersek:

$$(2n + 1) \mid \Big((-2n + 20) + (2n + 1)\Big) \implies (2n + 1) \mid 21.$$

Böylece değişken tamamen ortadan kalkar. $n$ pozitif bir tam sayı olduğundan $2n + 1 \geq 3$'tür. Dolayısıyla $2n + 1$, 21'in 1'den büyük pozitif bölenlerinden biri olmalıdır:

$$2n + 1 \in \{3, 7, 21\}.$$

Bu durumları çözersek:

$$\begin{aligned} 2n + 1 = 3 &\implies n = 1, \\ 2n + 1 = 7 &\implies n = 3, \\ 2n + 1 = 21 &\implies n = 10. \end{aligned}$$

İfadeyi 2 ile çarptığımız için elde edilen adayları ilk bağıntıda sınayalım:

- $n = 1$: $2(1) + 1 = 3$ ve $1^2 + 5 = 6$. $3 \mid 6$ sağlanır.
- $n = 3$: $2(3) + 1 = 7$ ve $3^2 + 5 = 14$. $7 \mid 14$ sağlanır.
- $n = 10$: $2(10) + 1 = 21$ ve $10^2 + 5 = 105$. $105 = 21 \times 5$ olduğundan $21 \mid 105$ sağlanır.

Tüm çözümler $n \in \{1, 3, 10\}$ olarak belirlenir. □

### 11.1.5 Algoritmik Bakış: Bölünebilirlik Testi

Programlamada $a$'nın $b$'yi bölüp bölmediğini kontrol etmek için mod operatöründen ("%") yararlanılır. $b$'nin $a$'ya bölümünden kalan 0 ise $a \mid b$'dir.

```
#include <stdbool.h>

// a, b'yi boler mi? (a != 0 varsayimiyla)
bool boler_mi(long long a, long long b) {
    if (a == 0) {
        return b == 0;
    }
    return (b % a) == 0;
}
```

Bu kontrol $O(1)$ zamanda yürütülür. Donanım seviyesinde işlemci tek bir bölme talimatı ("idiv") ile hem bölümü hem de kalanı aynı anda hesaplar.

## 11.2 Bölünebilme Kuralları

### 11.2.1 Motivasyon: Hangi Sorunu Çözer?

Elimizde 50 basamaklı bir sayı olduğunu varsayalım:

$$N = 74829104857201948572019485720194857201948572019488$$

Bu sayının 2, 3, 4 ya da 9'a bölünüp bölünmediğini belirlemek için uzun bölme yapmak gereksiz işlem maliyeti doğurur.

Günlük matematikte başvurduğumuz "son basamağı çiftse 2'ye bölünür" veya "rakamları toplamı 3'ün katıysa 3'e bölünür" gibi kurallar, kullandığımız **10 tabanlı sayı sisteminin** cebirsel özelliklerinden kaynaklanır. 10'un çarpanları ile kuvvetlerinin belirli sayılara göre kalanları bu kuralların temelini oluşturur.

### 11.2.2 Onluk Basamak Açılımı

Bir $N$ pozitif tam sayısının 10 tabanındaki basamakları sağdan sola doğru $d_0, d_1, d_2, \ldots, d_k$ olsun ($0 \le d_i \le 9$ ve $d_k \neq 0$). Sayının değeri:

$$N = \overline{d_k d_{k-1} \ldots d_1 d_0} = d_k 10^k + d_{k-1} 10^{k-1} + \cdots + d_2 10^2 + d_1 10^1 + d_0 10^0$$

biçiminde ifade edilir. Bölünebilme kuralları, $10^m$ terimlerini incelenen bölen cinsinden ayrıştırmaya dayanır.

### 11.2.3 Kurallar ve İspatları

#### 2, 5 ve 10 ile Bölünebilme (Son Basamak)

$10 = 2 \times 5$ olduğundan, her $m \geq 1$ için $10^m = 10 \cdot 10^{m-1}$ sayısı 2, 5 ve 10'un katıdır. Sayıyı şu şekilde ayrıştıralım:

$$N = 10 \cdot (d_k 10^{k-1} + \cdots + d_1) + d_0.$$

Parantez içindeki kısım 10 ile çarpıldığından 2, 5 ve 10 ile kalansız bölünür. Dolayısıyla $N$'nin bu sayılara bölünebilmesi birler basamağı $d_0$'a bağlıdır:

- $2 \mid N \iff 2 \mid d_0 \iff d_0 \in \{0, 2, 4, 6, 8\}$.
- $5 \mid N \iff 5 \mid d_0 \iff d_0 \in \{0, 5\}$.
- $10 \mid N \iff 10 \mid d_0 \iff d_0 = 0$.

#### 4, 25 ve 100 ile Bölünebilme (Son İki Basamak)

$100 = 10^2 = 4 \times 25$ olduğundan, her $m \geq 2$ için $10^m$ terimi 4, 25 ve 100'e kalansız bölünür. Sayıyı son iki basamağından ayıralım:

$$N = 100 \cdot (d_k 10^{k-2} + \cdots + d_2) + (10d_1 + d_0).$$

100 çarpanını içeren blok bu sayılara bölündüğünden:

- $4 \mid N \iff 4 \mid \overline{d_1 d_0}$.
- $25 \mid N \iff 25 \mid \overline{d_1 d_0} \iff \overline{d_1 d_0} \in \{00, 25, 50, 75\}$.

Bu yaklaşım genelleştirilebilir: $2^m$ ve $5^m$ için sayının son $m$ basamağına bakmak yeterlidir. Örneğin $8 = 2^3$ için son üç basamak incelenir ($8 \mid N \iff 8 \mid \overline{d_2 d_1 d_0}$).

#### 3 ve 9 ile Bölünebilme (Rakamlar Toplamı)

10'un kuvvetlerini inceleyelim:

$$\begin{aligned} 10 &= 9 + 1 \\ 100 &= 99 + 1 \\ 1000 &= 999 + 1 \\ 10^m &= \underbrace{99\ldots9}_{m \text{ tane}} + 1 = 9 \cdot \underbrace{11\ldots1}_{m \text{ tane}} + 1. \end{aligned}$$

Her $10^m$ sayısı 9'un bir katı ile 1'in toplamıdır. $N$ sayısının basamak açılımında bu eşitliği yerine yazalım:

$$\begin{aligned} N &= \sum_{i=0}^{k} d_i 10^i = \sum_{i=0}^{k} d_i(9K_i + 1) \quad (\text{burada } K_i = \underbrace{11\ldots1}_{i \text{ tane}}, K_0 = 0) \\ &= \sum_{i=0}^{k} 9K_i d_i + \sum_{i=0}^{k} d_i \\ &= 9 \cdot \left( \sum_{i=0}^{k} K_i d_i \right) + \sum_{i=0}^{k} d_i. \end{aligned}$$

İlk terim 9'un katıdır. $S(N) = \sum_{i=0}^{k} d_i$ sayının rakamları toplamı olmak üzere:

$$N = 9M + S(N)$$

yazılabilir. Bu eşitlik şu sonuçları verir:

- $9 \mid N \iff 9 \mid S(N)$.
- $3 \mid N \iff 3 \mid S(N)$ (çünkü $9M$ terimi 3'ün de katıdır).
- Bir sayının 9'a (veya 3'e) bölümünden kalan, rakamları toplamının 9'a (veya 3'e) bölümünden kalana eşittir.

### 11 ile Bölünebilme (Alternatif Toplam)

10 sayısı $11 - 1$ biçiminde yazılabilir. Kuvvetleri alındığında:

$$\begin{aligned} 10^0 &= 1 \\ 10^1 &= 11 - 1 \\ 10^2 &= 99 + 1 = 11 \times 9 + 1 \\ 10^3 &= 1001 - 1 = 11 \times 91 - 1 \\ 10^4 &= 9999 + 1 = 11 \times 909 + 1 \end{aligned}$$

Bölüm 6'da incelediğimiz binom açılımından:

$$10^m = (11 - 1)^m = 11 \cdot L_m + (-1)^m$$

yazılabilir. $10^m$'in 11'e bölümünden kalan $m$ çift ise $+1$, $m$ tek ise $-1$'dir. Bu ifadeyi basamak açılımına uygularsak:

$$\begin{aligned} N &= \sum_{i=0}^{k} d_i 10^i = \sum_{i=0}^{k} d_i \Big( 11L_i + (-1)^i \Big) \\ &= 11 \cdot \left( \sum_{i=0}^{k} L_i d_i \right) + \sum_{i=0}^{k} (-1)^i d_i \\ &= 11M' + \Big( d_0 - d_1 + d_2 - d_3 + \cdots + (-1)^k d_k \Big). \end{aligned}$$

Parantez içindeki ifade sayının **işaretli basamak toplamıdır**:

$$\mathrm{Alt}(N) = d_0 - d_1 + d_2 - d_3 + \dots$$

Dolayısıyla:

$$11 \mid N \iff 11 \mid \mathrm{Alt}(N).$$

**En Sık Yapılan Hata: 11 Kuralında İşaret Sırasını Karıştırmak**
İşaretleme sağdan sola yapılırken **birler basamağı daima pozitif (+)** seçilmelidir:

$$\cdots - d_3 + d_2 - d_1 + d_0$$

Soldan sağa başlandığında basamak sayısı çift ise işaretler ters döner. Her ne kadar $11 \mid X \iff 11 \mid (-X)$ olduğundan sadece bölünebilirlik testinde işaret yönü sonucu etkilemese de, *11'e bölümden kalanı bulurken* hatalı sonuca yol açar.

### 7, 11 ve 13 İçin Blok Kuralı: 1001 Çarpanı

1001 sayısının asal çarpanları şunlardır:

$$1001 = 7 \times 11 \times 13.$$

Buna göre $10^3 = 1000 = 1001 - 1$ sayısı 7, 11 ve 13 modüllerinde $-1$ kalanını verir. Tıpkı 11 kuralında basamakların tek tek $+/-$ şeklinde işaretlenmesi gibi, sayı **sağdan sola doğru üçerli bloklara** ayrılıp $+/-$ ile toplandığında elde edilen sonuç 7, 11 ve 13 ile bölünebilmeyi tek seferde sınar.

Örneğin $N = \overline{A_2A_1A_0}$ (üçer basamaklı bloklar) biçimindeyse:

$$N \equiv A_0 - A_1 + A_2 \pmod{7, 11, 13}.$$

## 11.2.4 Çözümlü Örnekler

**Örnek 11.7.** *Altı basamaklı $N = \overline{52x41y}$ sayısı 72 ile tam bölünebildiğine göre, $x$ ve $y$ rakamlarının alabileceği değerleri bulunuz.*

*Çözüm.* $72 = 8 \times 9$ olup 8 ve 9 aralarında asaldır. Sayının 72'ye bölünmesi hem 8'e hem de 9'a bölünmesini gerektirir. 8 ile bölünebilme son üç basamağa ($\overline{41y}$) bağlı olduğundan önce $y$ belirlenebilir.

**Adım 1: 8 ile bölünebilme.** Kural gereği $8 \mid \overline{41y}$ olmalıdır:

$$\overline{41y} = 400 + 10 + y = 8 \times 50 + (10 + y) = 8 \times 51 + (y + 2).$$

Buradan $8 \mid (y+2)$ elde edilir. $0 \le y \le 9$ olduğundan $y+2 \in \{2,3,\ldots,11\}$ aralığındadır. Bu aralıkta 8'in katı olan tek değer 8'dir:

$$y+2 = 8 \implies y = 6.$$

Sayımız $\overline{52x416}$ halini alır.

**Adım 2: 9 ile bölünebilme.** Rakamlar toplamı 9'un katı olmalıdır:

$$S(N) = 5+2+x+4+1+6 = 18+x.$$

$9 \mid (18+x)$ olması için $9 \mid x$ gereklidir. $x$ bir rakam olduğundan $x = 0$ veya $x = 9$ olabilir.

Çözüm çiftleri:

$$(x,y) \in \{(0,6),(9,6)\}.$$

□

**Örnek 11.8.** *Basamak sayısı çift olan ve rakamları simetrik dizilen tüm palindrom sayıların (örneğin 1221, 743347) 11 ile kalansız bölündüğünü kanıtlayınız.*

*Çözüm.* Palindrom sayının basamak sayısı $2k$ olsun. Simetri özelliği gereği baştan $i$'nci basamak ile sondan $i$'nci basamak birbirine eşittir.

Sayıyı basamaklarıyla yazalım:

$$N = \overline{a_k a_{k-1} \ldots a_2 a_1 a_1 a_2 \ldots a_{k-1} a_k}.$$

Sayının işaretli toplamını birler basamağından başlayarak oluşturalım:

$$\begin{aligned} \mathrm{Alt}(N) &= d_0 - d_1 + d_2 - d_3 + \cdots + (-1)^{2k-1} d_{2k-1} \\ &= a_k - a_{k-1} + a_{k-2} - \cdots + (-1)^{k-1} a_1 - (-1)^{k-1} a_1 + \cdots - a_k. \end{aligned}$$

Daha açık görmek için $2k = 4$ durumuna bakalım. $\overline{a_2 a_1 a_1 a_2}$ sayısı için:

$$\mathrm{Alt}(N) = a_2 - a_1 + a_1 - a_2 = (a_2 - a_2) + (-a_1 + a_1) = 0.$$

Genel durumda her $m \in \{1,2,\ldots,k\}$ için $a_m$ rakamı iki kez geçer. Basamakları birler basamağından başlayarak numaralandıralım (birler basamağının indeksi 0 olsun):

- Sağ yarıda sondan $m$'nci basamak olarak yer alır; indeksi $k-m$ olduğundan işareti $(-1)^{k-m}$'dir.
- Sol yarıda sondan $k-1+m$'nci basamak olarak yer alır; işareti $(-1)^{k-1+m}$'dir.

$k-1+m$ ile $k-m$ ardışık tam sayılar olduğundan bu iki işaret zıttır; dolayısıyla $a_m$'nin iki katkısı birbirini sıfırlar:

$$\mathrm{Alt}(N) = \sum_{m=1}^{k} a_m\Big((-1)^{k-m} + (-1)^{k-1+m}\Big) = \sum_{m=1}^{k} a_m \cdot 0 = 0.$$

$11 \mid 0$ olduğundan çift basamaklı tüm palindrom sayılar 11'e tam bölünür. □

**Örnek 11.9.** *Yalnızca 1 rakamı kullanılarak yazılan* $R_n = \underbrace{11\ldots1}_{n \text{ tane}}$ *sayılarına repunit denir.* $R_n$*'in 27 ile bölünebilmesi için gerek ve yeter şartın* $27 \mid n$ *olduğunu gösteriniz.*

*Çözüm.* $R_n$ sayısının rakamları toplamı $n \times 1 = n$'dir. Buradan $9 \mid R_n \iff 9 \mid n$ olduğu bilinir. 27 için durumu basamak bloklama yöntemiyle inceleyelim.

$n$ sayısı 3'ün katı olduğunda $R_n$ sayısını üçerli bloklara ayıralım: $R_3 = 111 = 3 \times 37$. $R_9 = \underbrace{111}_{R_3}\underbrace{111}_{R_3}\underbrace{111}_{R_3} = R_3 \times 1001001$. Burada 1001001'in rakamları toplamı $1+0+0+1+0+0+1 = 3$ olduğundan 3'e bölünür, fakat 9'a bölünmez. $R_{27}$ için:

$$R_{27} = R_9 \times (10^{18} + 10^9 + 1)$$

yazılabilir. Parantez içindeki sayının rakamları toplamı 3'tür; dolayısıyla 3'e bölünür ama 9'a bölünmez.

$R_n$'deki 3 çarpanlarının sayısı, $n$'in içerdiği 3 çarpanlarına bağlıdır:

- $R_n$'in 3'e bölünmesi için rakamlar toplamı olan $n$'in 3'e bölünmesi gerekir $(3 \mid n)$.
- $R_n$'in 9'a bölünmesi için rakamlar toplamı olan $n$'in 9'a bölünmesi gerekir $(9 \mid n)$.
- $n = 9k$ olduğunda $R_n$ sayısı $R_9$'un katıdır. $R_9 = 111111111 = 9 \times 12345679$'dur. 12345679'un rakamları toplamı 37 olup 3'e bölünmez. Dolayısıyla $R_9$ sayısı 9'a bölünür, fakat 27'ye bölünmez.
- $R_{9k}$ sayısını $R_9$ blokları cinsinden yazarsak:

  $$R_{9k} = R_9 \times \Big(10^{9(k-1)} + 10^{9(k-2)} + \cdots + 10^9 + 1\Big).$$

  $R_9$'un çarpanlarında $3^2$ bulunur ($3^3$ bulunmaz). $R_{9k}$'nin 27'ye ($3^3$) bölünebilmesi için parantezli ifadenin 3'e bölünmesi gerekir. Sağdaki ifade $k$ adet terimin toplamıdır ve her $10^{9m} \equiv 1 \pmod 3$ sağlandığından, parantez 3'e bölündüğünde $k$ kalanını verir:

  $$\text{Parantez} \equiv \underbrace{1+1+\cdots+1}_{k \text{ tane}} = k \pmod 3.$$

  İfadenin 3 ile bölünmesi için $3 \mid k$ olmalıdır.

$n = 9k$ ve $3 \mid k$ olduğunda $n = 9(3m) = 27m$, yani $27 \mid n$ elde edilir. □

### 11.2.5 Algoritmik Uygulama: Büyük Sayılarda Mod Hesabı

Büyük boyutlu sayılar programlamada dizgi (string) olarak saklanabilir. Böyle bir sayının küçük bir $M$ tam sayısına bölümünden kalanı Horner yöntemiyle döngü içinde hesaplayabiliriz:

```
FUNCTION largeNumberMod(numberStr, M)
    remainder = 0
    FOR EACH char IN numberStr DO
        digit = TO_INTEGER(char)
        remainder = (remainder * 10 + digit) MOD M
    END FOR
    RETURN remainder
END FUNCTION

number = "74829104857201948572019485720194857201948572019488"
OUTPUT "Remainder: ", largeNumberMod(number, 11)
```

Bu algoritma basamak sayısı $L$ olmak üzere $O(L)$ zamanda çalışır ve tam sayı taşması yaratmaz.

## 11.3 Bölme Algoritması

### 11.3.1 Motivasyon: Hangi Sorunu Çözer?

Şimdiye kadar kalanın 0 olduğu durumları inceledik. Ancak iki tam sayının bölümü her zaman tam sayı üretmez. 23 elmayı 5 kişiye paylaştırdığımızda kişi başı 4 elma düşer ve 3 elma artar: $23 = 5 \times 4 + 3$.

Tam sayılar kümesinde bölme işleminin bir tam sayı **bölüm** ve negatif olmayan bir **kalan** üretecek şekilde tek bir biçimde yapılabileceğini garanti eden teorem **Bölme Algoritması**dır (Division Algorithm). Adında "algoritma" geçmesine karşın bu ifade temel bir varlık ve teklik teoremidir.

### 11.3.2 Teorem ve İspatı

**Teorem 11.10** (Bölme Algoritması)**.** *$a$ ve $b$ iki tam sayı ve $b > 0$ olsun. Bu durumda,*

$$a = b \cdot q + r \quad ve \quad 0 \le r < b$$

*şartlarını sağlayan **bir ve yalnız bir** $(q, r)$ tam sayı ikilisi vardır.*

Burada $a$'ya *bölünen*, $b$'ye *bölen*, $q$'ya *bölüm* (quotient), $r$'ye ise *kalan* (remainder) denir.

*Kanıt.* İspat iki adımdan oluşur: Varlık ve Teklik. Bu ispat, Bölüm 19'da ele alacağımız **İyi Sıralılık İlkesi**'nin (negatif olmayan tam sayılardan oluşan boş olmayan her kümenin en küçük bir elemanı vardır) doğrudan bir uygulamasıdır.

**1. Varlık:** Şu kümeyi kuralım:

$$S = \{a - bk : k \in \mathbb{Z} \text{ ve } a - bk \geq 0\}.$$

Bu küme, $a$'dan $b$'nin katları çıkarıldığında elde edilen negatif olmayan tam sayılardan oluşur. $S$'nin boş olmadığını gösterelim:

- $a \geq 0$ ise $k = 0$ seçildiğinde $a - b(0) = a \geq 0$ olur, yani $a \in S$.
- $a < 0$ ise $k = a$ seçilebilir. $b \geq 1$ olduğundan $b \cdot a \leq a$ ve $a - ba = a(1-b) \geq 0$ elde edilir. Buradan $a - ba \in S$.

Her iki durumda da $S$ boş değildir ve elemanları negatif olmayan tam sayılardır.

İyi Sıralama İlkesi uyarınca $S$ kümesinin **en küçük bir elemanı** bulunur. Bu elemana $r$ diyelim. $r \in S$ olduğundan bir $q \in \mathbb{Z}$ için:

$$r = a - bq \implies a = bq + r \quad (r \geq 0).$$

$r < b$ olduğunu göstermek için aksini varsayalım: $r \geq b$ olsun. $r' = r - b$ sayısını ele alalım. $r \geq b$ olduğundan $r' \geq 0$'dır ve:

$$r' = (a - bq) - b = a - b(q + 1).$$

Bu durumda $r' \in S$ olur. Fakat $b > 0$ olduğundan $r' = r - b < r$ elde edilir. Bu ise $r$'nin $S$'deki en küçük eleman olmasıyla çelişir. Dolayısıyla $r < b$ sağlanmalıdır.

**2. Teklik:** İki farklı gösterim bulunduğunu varsayalım:

$$a = bq_1 + r_1 \quad (0 \leq r_1 < b)$$

$$a = bq_2 + r_2 \quad (0 \leq r_2 < b).$$

Taraf tarafa çıkarırsak:

$$0 = b(q_1 - q_2) + (r_1 - r_2) \implies b(q_2 - q_1) = r_1 - r_2.$$

Buradan $b \mid (r_1 - r_2)$ gelir. Ancak $0 \leq r_1 < b$ ve $0 \leq r_2 < b$ aralıkları gereğince:

$$-b < r_1 - r_2 < b.$$

Bu açık aralıkta $b$'nin katı olan tek sayı 0'dır. Dolayısıyla $r_1 - r_2 = 0 \implies r_1 = r_2$ olmalıdır. Eşitlikte yerine yazıldığında $b(q_2 - q_1) = 0$ ve $b > 0$ olduğundan $q_1 = q_2$ çıkar. Gösterim tektir. □

### 11.3.3 Negatif Sayılarda Bölme ve Alt Taban İlişkisi

Teoremde $b > 0$ koşulu aranırken $a$ pozitif, negatif ya da sıfır olabilir. $a$ negatif olduğunda $0 \leq r < b$ şartına dikkat edilmelidir.

Örneğin $a = -23$ ve $b = 5$ için:

- $-23 = 5 \times (-4) - 3$ yazıldığında $r = -3$ olur; kalan negatif olamayacağından bu yazım Bölme Algoritması'na uymaz.
- Doğru yazım için bölüm bir azaltılır ve kalan pozitif yapılır:

$$-23 = 5 \times (-5) + 2.$$

  Burada $q = -5$ ve $r = 2$ olup $0 \leq 2 < 5$ şartı sağlanır.

Bölüm ve kalan, **alt taban** (floor, $\lfloor x \rfloor$) fonksiyonu aracılığıyla tanımlanabilir:

$$q = \left\lfloor \frac{a}{b} \right\rfloor, \qquad r = a - b \left\lfloor \frac{a}{b} \right\rfloor.$$

$-23/5 = -4.6$ olduğundan $q = \lfloor -4.6 \rfloor = -5$ ve $r = -23 - 5(-5) = 2$ bulunur.

**En Sık Yapılan Hata: C/C++ ile Python'da Negatif Mod İşlemi Farkı**
Yarışma programlamasında dillerin "%" operatörünü işleme biçimi farklılık gösterir:

- **C / C++ (C99 standardı):** Bölme işlemini sıfıra doğru yuvarlar (truncate). Dolayısıyla "-23 / 5" ifadesi "-4", "-23 % 5" ifadesi ise "-3" sonucunu verir. Matematiksel tanımın aksine kalan negatif çıkabilir.
- **Python:** Taban fonksiyonunu ($\lfloor \cdot \rfloor$) esas alır. "-23 // 5" sonucu "-5", "-23 % 5" sonucu ise "+2" çıkar.

C/C++ ortamında matematiksel kalanı elde etmek için şu kalıp tercih edilir:

$$\text{mat\_mod}(a, b) = ((a\%b) + b)\%b$$

### 11.3.4 Tam Sayıların Sınıflandırılması

Bölme Algoritması, tam sayılar kümesini ($\mathbb{Z}$) bir $b$ bölenine göre sonlu sayıda ayrık kalan sınıfına böler:

- $b = 2$ için her tam sayı $2k$ (çift) ya da $2k + 1$ (tek) biçimindedir.
- $b = 3$ için her tam sayı $3k$, $3k + 1$ veya $3k + 2$ biçimindedir.

- $b = 4$ için her tam sayı $4k, 4k+1, 4k+2, 4k+3$ biçimindedir.

İspatlarda tüm tam sayıları tek tek incelemek yerine bu sonlu sayıdaki kalan sınıflarını ele almak yeterli olur.

### 11.3.5 Çözümlü Örnekler

**Örnek 11.11.** *Herhangi bir tam sayının karesinin 3 ile bölümünden kalanın yalnızca 0 veya 1 olabileceğini (asla 2 olamayacağını) kanıtlayınız.*

*Çözüm.* Bölme Algoritması'na göre her $n$ tam sayısı $3k$, $3k+1$ veya $3k+2$ biçimindedir. Bu durumların karelerini inceleyelim:

- **Durum 1 ($n = 3k$):**

$$n^2 = (3k)^2 = 9k^2 = 3(3k^2).$$

  Kalan 0'dır.

- **Durum 2 ($n = 3k+1$):**

$$n^2 = (3k+1)^2 = 9k^2 + 6k + 1 = 3(3k^2 + 2k) + 1.$$

  Kalan 1'dir.

- **Durum 3 ($n = 3k+2$):**

  $n^2 = (3k+2)^2 = 9k^2 + 12k + 4 = 9k^2 + 12k + 3 + 1 = 3(3k^2 + 4k + 1) + 1.$

  Kalan yine 1'dir.

Hiçbir durumda kalan 2 olamaz; kareler mod 3'te yalnızca 0 veya 1 değerini alabilir. □

**Örnek 11.12.** $x^2 + y^2 = 2023$ *denklemini sağlayan* $x, y \in \mathbb{Z}$ *tam sayılarının bulunmadığını kanıtlayınız.*

*Çözüm.* Tam sayı katsayılı denklemlerde kalan analizi pratik bir eleme sağlar. Tam karelerin 4 ile bölümünden kalanlarına bakalım:

$$(2k)^2 = 4k^2 \implies \text{kalan } 0,$$

$$(2k+1)^2 = 4k^2 + 4k + 1 = 4(k^2 + k) + 1 \implies \text{kalan } 1.$$

Bir tam karenin 4 ile bölümünden kalan yalnızca 0 ya da 1 olabilir.

Sol taraftaki $x^2 + y^2$ ifadesinin 4 ile bölümünden kalan olası değerler:

- $0 + 0 = 0$,

- $0 + 1 = 1$,
- $1 + 1 = 2$.

İki karenin toplamı 4 ile bölündüğünde asla 3 kalanını veremez.

Sağ taraftaki 2023 sayısına Bölme Algoritması uygulandığında:

$$2023 = 4 \times 505 + 3$$

elde edilir; yani kalan 3'tür. Sol taraf 4 modunda 3 değerini alamazken sağ taraf 3'tür. Bu çelişki nedeniyle denklemi sağlayan hiçbir $(x, y)$ tam sayı çifti yoktur. □

**Örnek 11.13.** *Ardışık $n$ tane tam sayının çarpımının her zaman $n!$ ile kalansız bölündüğünü gösteriniz.*

*Çözüm.* Ardışık $n$ tam sayıyı $k+1, k+2, \ldots, k+n$ olarak gösterelim. Çarpımları:

$$P = (k+1)(k+2)\ldots(k+n)$$

olur. $n! \mid P$ olduğunu göstermek, $\frac{P}{n!}$ ifadesinin tam sayı olduğunu göstermekle eşdeğerdir. Bölüm 5'te tanımladığımız kombinasyon bağıntısını hatırlayalım.

$k \geq 0$ için:

$$\frac{P}{n!} = \frac{(k+1)(k+2)\ldots(k+n)}{n!} = \frac{(n+k)!}{n!\,k!} = \binom{n+k}{n}.$$

$\binom{n+k}{n}$ katsayısı, $(n+k)$ elemanlı bir kümeden seçilen $n$ elemanlı alt küme sayısını temsil ettiğinden bir tam sayıdır.

Dolayısıyla $\frac{P}{n!} \in \mathbb{Z}$ olup $n! \mid P$ sağlanır. Sayılar negatif veya sıfır içeriyorsa:

- Çarpımda 0 bulunuyorsa $P = 0$ olur ve $n! \mid 0$ geçerlidir.
- Terimlerin tümü negatif ise $(-1)^n$ çarpanı paranteze alınarak pozitif duruma indirgenir.

Böylece ardışık $n$ tam sayının çarpımının daima $n!$ ile bölündüğü görülür. □

**Örnek 11.14.** *Her $n$ tam sayısı için $n^3 - n$ sayısının 6 ile tam bölündüğünü kanıtlayınız.*

*Çözüm.* İfadeyi çarpanlarına ayıralım:

$$n^3 - n = n(n^2 - 1) = (n-1)\,n\,(n+1).$$

Elde edilen çarpım, ardışık üç tam sayının çarpımıdır. Yukarıda kanıtladığımız sonuca göre ardışık $k$ tam sayının çarpımı daima $k!$ ile bölünür; $k = 3$ için bu $3! = 6$ demektir. Dolayısıyla $(n-1)n(n+1)$ çarpımı 6 ile tam bölünür ve istenen gösterilmiş olur. □

### 11.3.6 C/C++ İçin Güvenli Mod Fonksiyonu

Negatif sayılarla çalışırken Bölme Algoritması'nın matematiksel tanımını korumak adına aşağıdaki fonksiyon yapısı tercih edilir:

```
// Matematiksel bolme algoritmasi: 0 <= kalan < b
long long guvenli_mod(long long a, long long b) {
    long long r = a % b;
    if (r < 0) {
        r += b;
    }
    return r;
}

// Bolum q'yu hesaplamak icin: a = b*q + r
long long guvenli_bolum(long long a, long long b) {
    long long r = guvenli_mod(a, b);
    return (a - r) / b;
}
```

Bu fonksiyonlar sayesinde $a = -23$ ve $b = 5$ girildiğinde `guvenli_mod(-23, 5)` çağrısı `2`, `guvenli_bolum(-23, 5)` çağrısı ise `-5` değerini üretir.

# Bölüm 12

# Asal Sayılar

Tam sayılar dünyasını bir kimya laboratuvarına benzetirsek asal sayılar, bu evrenin bölünemez en temel "kimyasal elementleridir". Su molekülünü hidrojen ve oksijene, tuzu sodyum ve klora ayrıştırabildiğimiz gibi 60 sayısını da bileşenlerine ayırabiliriz: $60 = 2 \cdot 2 \cdot 3 \cdot 5$. Ancak 2, 3 ve 5 sayılarına ulaştığımızda artık daha küçük çarpanlara inemeyiz. Sayılar teorisindeki teoremler, algoritmik yapılar ve modern kriptografik protokoller, bu temel yapı taşlarının dağılımına ve özelliklerine dayanır.

Bölüm 11'de bölünebilme bağıntısını, kalan kavramını ve temel aritmetik özelliklerini incelemiştik. Şimdi bu yapının temel taşı olan asal sayıları, aralarında asallık ilişkisini ve her tam sayıya benzersiz bir çarpan kimliği kazandıran Aritmetiğin Temel Teoremi'ni ayrıntılarıyla ele alacağız.

## 12.1 Asal Sayılar ve Eratosthenes Kalburu

Bir problemi incelerken eldeki sayının daha küçük çarpanlara ayrılıp ayrılamayacağını belirlemek çoğu zaman ilk adımdır. Bir sayının asallığını nasıl test ederiz? Tek bir sayının asallığı ile belirli bir aralıktaki binlerce sayının asallığını denetlemek arasında algoritmik açıdan nasıl bir fark vardır? Bu sorular, bizi antik döneme uzanan klasik bir algoritmaya götürür.

### 12.1.1 Asal Sayı Tanımı ve Sezgisel Yaklaşım

1'den büyük her tam sayı en az iki pozitif bölene sahiptir: 1 ve kendisi. Bir sayının bu zorunlu iki bölen dışında hiçbir pozitif böleni yoksa, o sayı çarpanlarına daha fazla ayrılamaz.

**Tanım 12.1** (Asal Sayı ve Bileşik Sayı)**.** *1'den büyük bir $p$ tamsayısının pozitif bölenleri yalnızca 1 ve $p$ ise, $p$'ye bir* ***asal sayı*** *denir. 1'den büyük olup asal olmayan tamsayılara ise* ***bileşik sayı*** *(kompozit sayı) adı verilir.*

**Örnek 12.2.** *İlk birkaç asal sayıyı yazalım:*

$$2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, \ldots$$

*Burada* 2 *sayısı özel bir yere sahiptir: hem en küçük asal sayıdır hem de* ***çift olan yegâne asal sayıdır****.* 2*'den büyük tüm çift sayılar* 2*'ye bölünebildiğinden bileşik sayıdır.*

**En sık yapılan hata:** 1 sayısını asal kabul etmek sık rastlanan bir yanılgıdır. 1 sayısı ne asaldır ne de bileşiktir; bir "birim"dir (unit). 1'in asal kabul edilmemesinin en temel gerekçesi, birazdan göreceğimiz *Aritmetiğin Temel Teoremi*'nin tekliğini korumaktır. Eğer 1 asal olsaydı, örneğin 6 sayısı $2 \cdot 3$ olarak da, $1 \cdot 2 \cdot 3$ veya $1^5 \cdot 2 \cdot 3$ olarak da asal çarpanlarına ayrılabilirdi ve çarpanlara ayırmanın tekliği bozulurdu.

### 12.1.2 Bir Sayının Asallığını Test Etme: Deneme Bölmesi

$n = 101$ sayısının asal olup olmadığını nasıl anlarız? En kaba yaklaşım, 2'den 100'e kadar olan tüm tam sayıları tek tek deneyip 101'i bölüp bölmediğine bakmaktır. Ancak bu işlem büyük sayılar için oldukça yavaştır.

Bir $n$ sayısı bileşik ise, $1 < a \leq b < n$ olmak üzere $n = a \cdot b$ şeklinde iki tam sayının çarpımı olarak yazılabilir. Şimdi şu gözlemi yapalım: $a$ ve $b$ çarpanlarının her ikisi birden $\sqrt{n}$'den büyük olabilir mi? Eğer $a > \sqrt{n}$ ve $b > \sqrt{n}$ olsaydı:

$$a \cdot b > \sqrt{n} \cdot \sqrt{n} = n$$

elde edilirdi ki bu bir çelişkidir ($a \cdot b = n$ idi). Dolayısıyla çarpanlardan en az biri $\sqrt{n}$ değerine eşit ya da bundan küçük olmak zorundadır.

**Teorem 12.3** (Karekök Sınırı Teoremi)**.** $n > 1$ *bir bileşik sayı ise,* $n$*'nin* $1 < p \leq \sqrt{n}$ *koşulunu sağlayan en az bir* $p$ *asal böleni vardır.*

*Kanıt.* $n$ bileşik olduğundan $n = a \cdot b$ olacak şekilde $1 < a \leq b < n$ tamsayıları vardır. Yukarıdaki eşitsizlikten ötürü $a \leq \sqrt{n}$ olmak zorundadır. Aritmetiğin Temel Teoremi gereği (veya 1'den büyük her sayının en küçük böleninin asal olması ilkesinden), $a$'nın en az bir $p$ asal böleni vardır. $p \mid a$ ve $a \mid n$ olduğundan $p \mid n$ olur. Ayrıca $p \leq a \leq \sqrt{n}$ sağlandığından, $n$'nin $\sqrt{n}$'den küçük ya da eşit bir $p$ asal böleni mutlaka mevcuttur. □

Bu teorem hesaplama açısından önemli bir hızlanma sağlar: Bir sayının asal olup olmadığını sınamak için $n - 1$'e kadar değil, yalnızca $\lfloor\sqrt{n}\rfloor$'e kadar olan asal sayıları denemek yeterlidir.

**Örnek 12.4.** $n = 101$ *sayısının asal olup olmadığını inceleyelim.*

*Çözüm.* $\sqrt{101} \approx 10{,}049$ olduğundan, yalnızca 10'a kadar olan asalları kontrol etmemiz yeterlidir: Bu asallar $2, 3, 5, 7$'dir.

- 101 çift değildir, $2 \nmid 101$.
- Rakamlar toplamı $1 + 0 + 1 = 2$ olup 3'ün katı değildir, $3 \nmid 101$.
- Son basamağı 0 veya 5 değildir, $5 \nmid 101$.
- $101 = 7 \cdot 14 + 3$ olduğundan $7 \nmid 101$.

$\sqrt{101}$'e kadar hiçbir asal 101'i bölmediğine göre 101 kesinlikle asaldır. Yalnızca 4 bölme işlemi ile sonuca ulaştık. □

Aşağıdaki algoritma tek bir sayının asallığını $O(\sqrt{n})$ zaman karmaşıklığında denetler:

```
FUNCTION isPrime(n)
    IF n <= 1 THEN
        RETURN False
    END IF
    IF n <= 3 THEN
        RETURN True
    END IF
    IF n MOD 2 = 0 OR n MOD 3 = 0 THEN
        RETURN False
    END IF

    SET i = 5
    WHILE i * i <= n DO
        IF n MOD i = 0 OR n MOD (i + 2) = 0 THEN
            RETURN False
        END IF
        i = i + 6
    END WHILE

    RETURN True
END FUNCTION
```

### 12.1.3 Aralıktaki Asalları Bulma: Eratosthenes Kalburu

Tek bir sayının değil de 1'den $N$'e kadar olan tüm sayıların asallığına ihtiyaç duyuluyorsa her sayı için ayrı ayrı deneme bölmesi yapmak $O(N\sqrt{N})$ zaman alır. $N = 10^7$ gibi büyük değerler için bu yaklaşım verimsizdir.

Kirenesli Eratosthenes, sayıları tek tek test etmek yerine tersine bir yöntem kurdu: Sayıları bir listeye yazıp, asal olduğu bilinen sayıların katlarını listeden eleyerek geriye yalnızca asalları bırakmak.

**Adım Adım Kalbur Mekanizması ($N = 30$ için):**

1. 2'den 30'a kadar tüm sayıları yazalım.

2. Karşılaştığımız ilk silinmemiş sayı 2'dir; 2 asaldır. 2'nin 2'den büyük tüm katlarını $(4, 6, 8, \ldots, 30)$ çizelim.

3. Sıradaki silinmemiş sayı 3'tür; 3 asaldır. 3'ün katlarını çizelim $(6, 9, 12, \ldots)$. Burada 6 sayısı 2'nin katı olarak zaten çizilmiştir; bu nedenle çizmeye doğrudan $3^2 = 9$'dan başlayabiliriz.

4. Sıradaki silinmemiş sayı 5'tir; 5 asaldır. $5^2 = 25 \leq 30$ olduğundan 25'i de çizeriz.

5. Bir sonraki sayı 7'dir. $7^2 = 49 > 30$ olduğundan işlem tamamlanır. Silinmeden kalan tüm sayılar $(2, 3, 5, 7, 11, 13, 17, 19, 23, 29)$ asaldır.

**Teorem 12.5** (Eratosthenes Kalburunun Karmaşıklığı)**.** *$N$ sayısına kadar olan asal sayıları Eratosthenes kalburu ile bulmanın zaman karmaşıklığı $O(N \log \log N)$'dir.*

*Sezgi ve Taslak.* Her $p$ asalı için yapılan eleme işlemi sayısı yaklaşık $\frac{N}{p}$ kadardır. Dolayısıyla toplam işlem sayısı:

$$\sum_{p \leq N, p \text{ asal}} \frac{N}{p} = N \sum_{p \leq N} \frac{1}{p}$$

Mertens'in ikinci teoremi uyarınca, asal sayıların çarpmaya göre terslerinin toplamı $\sum_{p \leq N} \frac{1}{p} = \ln(\ln N) + O(1)$ şeklinde büyür. Bu nedenle kalburun karmaşıklığı $O(N \log \log N)$ olur. Bu süre, pratikte $O(N)$ doğrusal zamana oldukça yakındır. □

```
#include <stdio.h>
#include <stdbool.h>
#define MAXN 10000000

bool asal[MAXN + 1];

void eratosthenes_kalburu(int n) {
    for (int i = 2; i <= n; i++)
        asal[i] = true;

    for (int p = 2; p * p <= n; p++) {
        if (asal[p]) {
            // p'nin katlarini p*p'den baslayarak eliyoruz
            for (int i = p * p; i <= n; i += p)
                asal[i] = false;
        }
    }
}
```

### 12.1.4 Çözümlü Örnekler

**Örnek 12.6.** *$p$, $p+2$ ve $p+4$ sayılarının üçünün birden asal olmasını sağlayan tüm $p$ asal sayılarını bulunuz.*

*Çözüm.* Küçük asal değerleri deneyerek bir örüntü arayalım:

- $p = 2 \implies 2, 4, 6$ (4 ve 6 asal değil).
- $p = 3 \implies 3, 5, 7$ (Üçü de asaldır; bir çözüm bulundu).
- $p = 5 \implies 5, 7, 9$ ($9 = 3 \cdot 3$ asal değil).
- $p = 7 \implies 7, 9, 11$ (9 asal değil).

Her denemede sayılardan biri 3'ün katı çıkmaktadır. Ardışık tek sayılar, Bölüm 14'te ayrıntılandıracağımız modüler aritmetik yaklaşımıyla 3 modunda incelenmelidir.

Herhangi bir $p$ tam sayısının 3'e bölümünden kalan $0, 1$ veya 2 olabilir.

1. $p \equiv 0 \pmod 3$: $p$ bir asal sayı olduğundan ve 3'e bölündüğünden zorunlu olarak $p = 3$ olmalıdır. Bu durumda $p + 2 = 5$ ve $p + 4 = 7$ olup her üç sayı da asaldır.

2. $p \equiv 1 \pmod 3$: Bu durumda $p + 2 \equiv 1 + 2 \equiv 0 \pmod 3$ olur. $p > 1$ olduğundan $p + 2 > 3$'tür ve 3'e bölünen 3'ten büyük bir sayı asal olamaz ($p + 2$ bileşiktir).

3. $p \equiv 2 \pmod 3$: Bu durumda $p + 4 \equiv 2 + 4 \equiv 0 \pmod 3$ olur. Benzer şekilde $p + 4 > 3$ olduğundan $p + 4$ bileşiktir.

Sonuç olarak koşulu sağlayan tek asal sayı $p = 3$'tür. □

**Örnek 12.7.** *$n > 1$ bir tam sayı olmak üzere, $n^4 + 4$ sayısının asal olamayacağını kanıtlayınız.*

*Çözüm.* Bir ifadenin asal olmadığını göstermek için onu 1'den büyük iki tam sayının çarpımı şeklinde yazabiliriz. $n^4 + 4$ ifadesi Sophie Germain özdeşliğiyle tam kareye tamamlanabilir. İfadeye $4n^2$ terimini ekleyip çıkaralım:

$$\begin{aligned} n^4 + 4 &= (n^4 + 4n^2 + 4) - 4n^2 \\ &= (n^2 + 2)^2 - (2n)^2 \end{aligned}$$

İki kare farkı özdeşliğini ($A^2 - B^2 = (A - B)(A + B)$) uygularsak:

$$n^4 + 4 = (n^2 - 2n + 2)(n^2 + 2n + 2)$$

çarpanlarını elde ederiz. Bu çarpanların büyüklüklerini inceleyelim:

$$n^2 - 2n + 2 = (n-1)^2 + 1$$

$n > 1$ olduğundan $(n-1)^2 \geq 1$ ve dolayısıyla $(n-1)^2 + 1 \geq 2 > 1$'dir. Diğer çarpan da açıkça $n^2 + 2n + 2 > 2$'dir. $n^4 + 4$ sayısı, 1'den büyük iki tam sayının çarpımı olarak yazılabildiğinden hiçbir $n > 1$ tam sayısı için asal olamaz. □

**Örnek 12.8** (Öklid Teoremi)**.** *Asal sayıların sonsuz çoklukta olduğunu kanıtlayınız.*

*Çözüm.* Bölüm 2'de gördüğümüz çelişkiyle ispat yöntemini kullanalım. Asal sayıların sonlu sayıda, tam olarak $k$ tane olduğunu varsayalım. Bu asalları sıralayalım:

$$p_1 = 2, p_2 = 3, p_3 = 5, \ldots, p_k$$

Şimdi tüm bu asalların çarpımının 1 fazlası olan bir $N$ sayısı tanımlayalım:

$$N = (p_1 \cdot p_2 \ldots p_k) + 1$$

$N > 1$ olduğundan $N$'nin en az bir asal böleni bulunmalıdır. Listemiz tüm asalları içerdiğinden bu bölen, listedeki asallardan biri (örneğin $p_i$, $1 \leq i \leq k$) olmak zorundadır. O halde $p_i \mid N$ olur.

Diğer taraftan $p_i$ sayısı $p_1 \cdot p_2 \ldots p_k$ çarpımının çarpanlarından biridir:

$$p_i \mid (p_1 \cdot p_2 \ldots p_k)$$

Bölünebilmenin temel kuralı gereği, bir sayı iki tam sayıyı da bölüyorsa farklarını da böler:

$$p_i \mid \Big(N - (p_1 \cdot p_2 \ldots p_k)\Big) \implies p_i \mid 1$$

Fakat 1'i bölen hiçbir asal sayı yoktur ($p_i \geq 2$). Bu çelişki, asal sayıların sonlu olduğu varsayımının yanlış olduğunu gösterir; asal sayılar kümesi sonsuzdur. □

## 12.2 Aralarında Asal Sayılar

İki sayının 1 dışında ortak bir asal çarpana sahip olmaması, sayılar teorisinin en temel kavramları arasındadır. Kesirlerin en yalın haline sadeleştirilmesinden Bölüm 14'te göreceğimiz Çin Kalan Teoremi'ne kadar pek çok yapı bu özelliğe dayanır.

### 12.2.1 Tanım ve Temel Özellikler

**Tanım 12.9** (Aralarında Asal)**.** *$a$ ve $b$ sıfırdan farklı iki tamsayı olsun. Eğer $a$ ve $b$'nin 1'den büyük hiçbir pozitif ortak böleni yoksa, yani en büyük ortak bölenleri* $\gcd(a, b) = 1$ *ise, $a$ ve $b$ sayılarına* ***aralarında asal*** *(coprime veya relatively prime) denir.*

**En sık yapılan hata:** Aralarında asal olan sayıların kendilerinin de asal olması gerektiği yanılgısıdır. Örneğin:

$$a = 8 = 2^3 \quad \text{ve} \quad b = 9 = 3^2$$

Her iki sayı da bileşiktir; ancak pozitif ortak bölenleri yalnızca 1 olduğundan $\gcd(8, 9) = 1$'dir, yani 8 ile 9 aralarında asaldır.

### 12.2.2 İkili ve Kümesel Aralarında Asallık Ayrımı

Üç veya daha fazla sayı söz konusu olduğunda iki farklı aralarında asallık düzeyi ayırt edilmelidir:

1. **Birlikte (kümesel) aralarında asal:** $\gcd(a_1, a_2, \ldots, a_k) = 1$.
2. **İkişer ikişer (pairwise) aralarında asal:** Her $i \neq j$ için $\gcd(a_i, a_j) = 1$.

**Örnek 12.10.** $6, 10, 15$ *sayılarını ele alalım:*

$$\gcd(6, 10, 15) = 1$$

*Dolayısıyla bu üç sayı birlikte aralarında asaldır. Fakat ikili ortak bölenlere baktığımızda:*

$$\gcd(6, 10) = 2 \neq 1, \quad \gcd(6, 15) = 3 \neq 1, \quad \gcd(10, 15) = 5 \neq 1$$

*Bu sayılar ikişer ikişer aralarında asal* ***değildir****. İkişer ikişer aralarında asallık, birlikte aralarında asallığa kıyasla daha güçlü bir koşuldur.*

### 12.2.3 Temel Teoremler

Aralarında asallık analizlerinde Öklid Lemması ve türevleri temel araçlardır. EBOB hesaplarına ve Bölüm 13'te ele alacağımız Öklid algoritmasına zemin hazırlayan bu sonuçları netleştirelim.

**Teorem 12.11** (Öklid Lemması)**.** *$a, b, c$ tamsayılar olsun. Eğer $a \mid bc$ ve $\gcd(a, b) = 1$ ise, o halde $a \mid c$ olmak zorundadır.*

*Sezgi.* $a$ sayısı $bc$ çarpımını bölmektedir; yani çarpım $a$'nın tüm asal çarpanlarını içerir. Ancak $\gcd(a, b) = 1$ olduğundan $a$, bu çarpanların hiçbirini $b$'den alamaz. Dolayısıyla $a$'nın gerektirdiği tüm çarpanlar $c$'de bulunmalıdır; yani $a \mid c$. □

**Teorem 12.12** (Kuvvetlerin Aralarında Asallığı)**.** *$a$ ve $b$ pozitif tamsayılar olmak üzere $\gcd(a, b) = 1$ ise:*

1. *Her $m, n \geq 1$ için $\gcd(a^m, b^n) = 1$'dir.*

2. *Eğer $a \cdot b = c^k$ olacak şekilde bir $c$ tamsayısı ve $k \geq 1$ pozitif tamsayısı varsa, $a$ ve $b$ sayılarının her biri birer tam $k$. kuvvet olmak zorundadır. Yani $a = x^k$ ve $b = y^k$ olacak biçimde $x, y$ tamsayıları vardır.*

Bu ikinci özellik, tam sayı katsayılı denklemlerde çarpanları ayırırken sıkça kullanılır. Çarpımları bir tam kare (veya tam küp) olan aralarında asal iki pozitif tam sayı varsa, her bir çarpan da ayrı ayrı bir tam kare (veya tam küp) olmak zorundadır.

### 12.2.4 Çözümlü Örnekler

**Örnek 12.13.** *Her $n$ pozitif tamsayısı için $n$ ile $n+1$ sayılarının aralarında asal olduğunu kanıtlayınız.*

*Çözüm.* İki sayının ortak bölenini ararken farklarını incelemek böleni sınırlandırmanın pratik bir yoludur.

$d = \gcd(n, n+1)$ olsun. En büyük ortak bölen tanımı gereği $d \mid n$ ve $d \mid (n+1)$ olmalıdır. Bölüm 11'den bildiğimiz üzere $d$, bu sayıların doğrusal kombinasyonlarını da böler:

$$d \mid \big((n+1) - n\big) \implies d \mid 1$$

Pozitif bir bölen olan $d$, 1'i bölüyorsa $d = 1$ olmak zorundadır. Demek ki ardışık iki pozitif tam sayı daima aralarında asaldır. □

**Örnek 12.14** (Uluslararası Matematik Olimpiyatı 1959, Problem 1)**.** *Her $n$ pozitif tamsayısı için*

$$\frac{21n+4}{14n+3}$$

*kesrinin sadeleştirilemez olduğunu gösteriniz.*

Bu klasik problem, pay ve paydası aralarında asal olan kesirlerin sadeleşemeyeceği gözlemine dayanır. Soruyu, Öklid algoritmasını kurduktan sonra Bölüm 13'te tam olarak ele alacağız.

**Örnek 12.15.** $\gcd(a, b) = 1$ *ise* $\gcd(a+b, a \cdot b) = 1$ *olduğunu kanıtlayınız.*

*Çözüm.* Karşıt varsayımla başlayalım: Ortak bir $p$ asal böleni bulunduğunu varsayıp bunun $\gcd(a, b) = 1$ kabulüyle çeliştiğini gösterelim.

Varsayalım ki $\gcd(a+b, ab) > 1$ olsun. Bu durumda $a+b$ ve $ab$ sayılarını bölen bir $p$ asal sayısı vardır:

$$p \mid (a+b) \quad \text{ve} \quad p \mid ab$$

$p$ bir asal sayı ve $p \mid ab$ olduğundan, Öklid Lemması uyarınca $p \mid a$ veya $p \mid b$ olmalıdır.

- Eğer $p \mid a$ ise: $p \mid (a+b)$ olduğundan $b = (a+b) - a$ sayısını da böler; yani $p \mid b$ olur.

- Eğer $p \mid b$ ise: benzer biçimde $a = (a+b) - b$ olduğundan $p \mid a$ olur.

Her iki durumda da $p \mid a$ ve $p \mid b$ elde edilir. Bu ise $p$'nin $a$ ve $b$'nin bir ortak böleni olduğu anlamına gelir:

$$p \leq \gcd(a, b) = 1$$

Bu sonuç $p \geq 2$ asallığı ile çelişir. O halde $\gcd(a+b, ab) = 1$ olmak zorundadır. □

**Örnek 12.16.** $\{1, 2, 3, \ldots, 2n\}$ *kümesinden rastgele* $n+1$ *tane tamsayı seçiliyor. Seçilen bu sayılar arasında en az iki tanesinin aralarında asal olduğunu kanıtlayınız.*

*Çözüm.* Aralarında asal en temel sayı çiftleri ardışık tam sayılardır ($\gcd(k, k+1) = 1$). Dolayısıyla seçilen sayılar arasında ardışık iki sayının varlığını garanti etmek yeterlidir. Bu yapı Bölüm 7'de incelediğimiz Güvercin Yuvası İlkesi ile doğrudan kurulabilir.

Kümeyi ardışık elemanlardan oluşan $n$ ayrık ikiliye ayıralım:

$$Y_1 = \{1, 2\}, \quad Y_2 = \{3, 4\}, \quad Y_3 = \{5, 6\}, \quad \ldots, \quad Y_n = \{2n-1, 2n\}$$

Toplam $n$ kümemiz (yuva) vardır. Bu yuvalardan $n+1$ eleman (güvercin) seçildiğinde Güvercin Yuvası İlkesi gereği en az iki eleman aynı ikiliye ait olmak zorundadır. Aynı ikilideki sayılar $\{2k-1, 2k\}$ biçiminde ardışık iki tam sayıdır ve $\gcd(2k-1, 2k) = 1$'dir. Böylece seçilen sayılar arasında mutlaka aralarında asal en az iki sayı bulunur. □

## 12.3 Aritmetiğin Temel Teoremi ve Asal Çarpanlara Ayırma

Asal sayıların önemi, tüm pozitif tam sayılar kümesini benzersiz biçimde inşa etme gücünden gelir.

### 12.3.1 Aritmetiğin Temel Teoremi

**Teorem 12.17** (Aritmetiğin Temel Teoremi)**.** 1*'den büyük her* $n$ *pozitif tamsayısı, asal sayıların bir çarpımı olarak yazılabilir. Dahası, çarpanların yazılış sırası dikkate alınmazsa, bu gösterim* ***benzersizdir (tektir)****.*

*İspat Taslağı.* Teorem iki iddiadan oluşur: Varlık ve Teklik.

1. **Varlık (Her sayı asal çarpımı olarak yazılabilir):** Kuvvetli tümevarım uygulayalım. $n = 2$ bir asaldır, dolayısıyla tek bir asalın çarpımıdır (doğru). $2 \leq k < n$ koşulunu sağlayan tüm tamsayıların asal çarpımı olarak yazılabildiğini varsayalım. $n$ sayısı asal ise iddia doğrudur. Eğer $n$ bileşik ise,

$1 < a, b < n$ olmak üzere $n = a \cdot b$ yazılabilir. Tümevarım varsayımı gereği hem $a$ hem de $b$ asal sayıların çarpımı olarak yazılabilir. Bu iki çarpımın yan yana getirilmesiyle $n$ de asalların çarpımı olarak ifade edilmiş olur.

2. **Teklik (Bu gösterim sırası hariç tektir):** Tersini varsayalım; gösterimi tek olmayan en az bir tamsayı bulunsun. Pozitif tamsayıların iyi sıralılık ilkesi gereği, iki farklı asal çarpan gösterimine sahip en küçük sayı $n$ olsun:

$$n = p_1 p_2 \dots p_r = q_1 q_2 \dots q_s$$

Burada $p_i$ ve $q_j$ asal sayılardır. $p_1 \mid n$ olduğundan $p_1 \mid (q_1 q_2 \dots q_s)$ olur. Öklid lemması gereği $p_1$, $q$ asallarından en az birini bölmek zorundadır. Sıralamayı düzenleyerek $p_1 \mid q_1$ olduğunu kabul edelim. $p_1$ ve $q_1$ birer asal sayı olduğundan bu ancak $p_1 = q_1$ olmasıyla mümkündür! Her iki tarafı bu ortak asala bölersek:

$$\frac{n}{p_1} = p_2 \dots p_r = q_2 \dots q_s$$

elde edilir. Fakat $\frac{n}{p_1} < n$ sayısı da iki farklı asal çarpan açılımına sahip olur. Bu durum $n$'nin "en küçük" böylesi sayı olması kabulümüzle çelişir. Dolayısıyla çarpanlara ayrılış tektir.

□

### 12.3.2 Kanonik Asal Çarpanlar Açılımı

Her $n > 1$ tam sayısı, aynı asallar kuvvet olarak toplanarak tek biçimde ifade edilir:

$$n = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_k^{\alpha_k}$$

Burada $p_1 < p_2 < \dots < p_k$ farklı asal sayılar, $\alpha_i \geq 1$ pozitif tam sayı üslerdir. Bu gösterime $n$'nin **kanonik asal açılımı** denir.

**Örnek 12.18.** 360 *sayısının kanonik açılımı:*

$$360 = 2^3 \cdot 3^2 \cdot 5^1$$

### 12.3.3 Bölen Sayısı Formülü: $d(n)$

$n$ sayısının pozitif bölenlerini inceleyelim. Bir $d$ tam sayısının $n$'yi bölebilmesi için, $d$'nin asal çarpanlarının $n$'nin asal çarpanları arasından seçilmesi ve üslerinin $n$'dekileri aşmaması gerekir:

$$d = p_1^{\beta_1} p_2^{\beta_2} \dots p_k^{\beta_k}, \quad 0 \leq \beta_i \leq \alpha_i$$

**Kombinatorik Çıkarım:** Her bir $\beta_i$ üssünü belirlemek bağımsız birer seçim adımıdır:

- $\beta_1$ için $\{0, 1, 2, \ldots, \alpha_1\}$ olmak üzere $\alpha_1 + 1$ farklı seçenek vardır.
- $\beta_2$ için $\{0, 1, 2, \ldots, \alpha_2\}$ olmak üzere $\alpha_2 + 1$ farklı seçenek vardır.
- ...
- $\beta_k$ için $\{0, 1, 2, \ldots, \alpha_k\}$ olmak üzere $\alpha_k + 1$ farklı seçenek vardır.

Bölüm 3'te gördüğümüz çarpma kuralı uyarınca bu seçenekleri çarparız.

**Teorem 12.19** (Pozitif Bölen Sayısı Formülü)**.** *$n = p_1^{\alpha_1} p_2^{\alpha_2} \ldots p_k^{\alpha_k}$ kanonik açılımına sahip bir $n$ tamsayısının pozitif bölen sayısı $d(n)$ (veya $\tau(n)$):*

$$d(n) = (\alpha_1 + 1)(\alpha_2 + 1) \ldots (\alpha_k + 1)$$

*formülü ile hesaplanır.*

**Örnek 12.20.** $360 = 2^3 \cdot 3^2 \cdot 5^1$ *sayısının pozitif bölen sayısını bulalım:*

$$d(360) = (3 + 1)(2 + 1)(1 + 1) = 4 \cdot 3 \cdot 2 = 24$$

$360$*'ın tam olarak* $24$ *farklı pozitif böleni vardır.*

### 12.3.4 Pozitif Bölenlerin Toplamı: $\sigma(n)$

Tüm pozitif bölenlerin toplamı $\sigma(n)$ ile gösterilir. Bu toplam, şu çarpımın parantezlerinin açılmış halidir:

$$\sigma(n) = \left(1 + p_1 + p_1^2 + \cdots + p_1^{\alpha_1}\right)\left(1 + p_2 + p_2^2 + \cdots + p_2^{\alpha_2}\right) \ldots \left(1 + p_k + \cdots + p_k^{\alpha_k}\right)$$

Geometrik dizi toplam formülü ($1 + p + \cdots + p^{\alpha} = \frac{p^{\alpha+1}-1}{p-1}$) kullanılırsa:

**Teorem 12.21** (Bölenler Toplamı Formülü)**.**

$$\sigma(n) = \prod_{i=1}^{k} \frac{p_i^{\alpha_i+1} - 1}{p_i - 1}$$

### 12.3.5 Pozitif Bölenlerin Çarpımı

$n$'nin tüm pozitif bölenlerini küçükten büyüğe $d_1, d_2, \ldots, d_m$ olarak sıralayalım ($m = d(n)$). Her $d$ böleni için $\frac{n}{d}$ de $n$'nin bir bölenidir. Bölenleri baştan ve sondan eşleştirdiğimizde:

$$d_1 \cdot d_m = n, \quad d_2 \cdot d_{m-1} = n, \quad \ldots$$

Tüm $m$ bölen çarpıldığında her çift $n$ değerini verir:

**Teorem 12.22** (Bölenler Çarpımı)**.** *$n$ sayısının pozitif bölenlerinin çarpımı:*

$$P(n) = n^{\frac{d(n)}{2}} = \sqrt{n^{d(n)}}$$

### 12.3.6 Legendre Formülü ($n!$ İçindeki Asal Üsleri)

Faktöriyelli ifadelerin çarpan analizinde Legendre formülü temel bir rol oynar. $n! = 1 \cdot 2 \cdot 3 \dots n$ çarpımının asal açılımında bir $p$ asalının üssü şu teoremle hesaplanır:

**Teorem 12.23** (Legendre Formülü)**.** *$n!$ sayısının asal çarpanlarına ayrılmış biçimindeki $p$ asalının en büyük üssü $v_p(n!)$:*

$$v_p(n!) = \sum_{j=1}^{\infty} \left\lfloor \frac{n}{p^j} \right\rfloor = \left\lfloor \frac{n}{p} \right\rfloor + \left\lfloor \frac{n}{p^2} \right\rfloor + \left\lfloor \frac{n}{p^3} \right\rfloor + \dots$$

*ile verilir. Buradaki toplam, $p^j > n$ olduğunda terimler sıfırlandığı için sonludur.*

*Sezgi.* 1'den $n$'ye kadar olan sayılar arasında:

- $p$'nin katı olan $\lfloor \frac{n}{p} \rfloor$ tane sayı vardır. Bunların her biri çarpıma en az bir adet $p$ çarpanı kazandırır.
- $p^2$'nin katı olan $\lfloor \frac{n}{p^2} \rfloor$ tane sayı vardır. Bunlar ikinci bir $p$ çarpanı daha kazandırır.
- $p^3$'ün katı olan sayılar üçüncü bir $p$ çarpanı kazandırır ve süreç bu şekilde devam eder.

Tüm bu katkılar toplandığında formül doğrudan elde edilir. □

**Örnek 12.24.** $100!$ *sayısının sonunda kaç tane sıfır vardır?*

*Çözüm.* Bir sayının sonundaki sıfır sayısı, çarpanlarındaki $10 = 2 \cdot 5$ adedine, yani $\min(v_2(100!), v_5(100!))$ değerine eşittir. Faktöriyel çarpımında 2 çarpanı her iki adımda bir gelirken 5 çarpanı beş adımda bir geldiğinden, 2'nin kuvveti her zaman 5'in kuvvetinden büyüktür. Dolayısıyla yalnızca $v_5(100!)$ değerini hesaplamak yeterlidir:

$$v_5(100!) = \left\lfloor \frac{100}{5} \right\rfloor + \left\lfloor \frac{100}{25} \right\rfloor + \left\lfloor \frac{100}{125} \right\rfloor = 20 + 4 + 0 = 24$$

Demek ki $100!$ sayısının sonunda tam 24 adet sıfır bulunur. □

### 12.3.7 Algoritmik Uygulama: Hızlı Asal Çarpanlara Ayırma

Tek bir $n$ sayısını $O(\sqrt{n})$ sürede asal çarpanlarına ayıran algoritma:

```
FUNCTION primeFactors(n)
    factors = EMPTY_MAP
    d = 2
    WHILE d * d <= n DO
        IF n MOD d == 0 THEN
```

```
            count = 0
            WHILE n MOD d == 0 DO
                count = count + 1
                n = n DIV d
            END WHILE
            factors[d] = count
        END IF
        d = d + 1
    END WHILE
    IF n > 1 THEN
        factors[n] = 1
    END IF
    RETURN factors
END FUNCTION
```

**En Küçük Asal Bölen (SPF) Yaklaşımı:** 1'den $N$'e kadar çok sayıda sayı için art arda çarpanlara ayırma sorgusu yapılacaksa (örneğin $10^5$ sorgu), her sayı için $O(\sqrt{n})$ deneme yapmak toplamda maliyetlidir. Bunun yerine Eratosthenes kalburu uyarlanarak her sayının **en küçük asal böleni** (Smallest Prime Factor - SPF) $O(N \log \log N)$ sürede önceden hesaplanabilir. Ardından herhangi bir $n$ sayısı $O(\log n)$ adımda çarpanlarına ayrılabilir:

```
#define MAXN 1000000
int spf[MAXN + 1];

void spf_hesapla(int n) {
    for (int i = 1; i <= n; i++) spf[i] = i;
    for (int i = 2; i * i <= n; i++) {
        if (spf[i] == i) { // i asal
            for (int j = i * i; j <= n; j += i) {
                if (spf[j] == j)
                    spf[j] = i; // j'nin en kucuk asal boleni i'dir
            }
        }
    }
}

// n'i O(log n) surede carpanlarina ayirma:
void carpani_yazdir(int n) {
    while (n > 1) {
        int p = spf[n];
        int kuvvet = 0;
        while (n % p == 0) {
            kuvvet++;
            n /= p;
        }
        printf("%d^%d ", p, kuvvet);
    }
    printf("\n");
```

```
}
```

### 12.3.8 Çözümlü Örnekler

**Örnek 12.25.** *Tam olarak* 12 *tane pozitif böleni olan en küçük pozitif tamsayı kaçtır?*

*Çözüm.* Aradığımız sayı $n = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_k^{\alpha_k}$ olsun. Pozitif bölen sayısı formülünden:

$$d(n) = (\alpha_1 + 1)(\alpha_2 + 1) \dots (\alpha_k + 1) = 12$$

Sayının en küçük olmasını sağlamak için:

1. En küçük asalları taban olarak seçmeliyiz $(2, 3, 5, \dots)$.
2. Büyük üsleri daha küçük asal tabanlara dağıtmalıyız ($p_1 < p_2 < \dots$ iken $\alpha_1 \geq \alpha_2 \geq \dots$).

12'nin çarpan gruplarını inceleyelim:

- $12 = 12 \implies \alpha_1 + 1 = 12 \implies \alpha_1 = 11$. Aday: $2^{11} = 2048$.
- $12 = 6 \cdot 2 \implies \alpha_1 = 5, \alpha_2 = 1$. Aday: $2^5 \cdot 3^1 = 32 \cdot 3 = 96$.
- $12 = 4 \cdot 3 \implies \alpha_1 = 3, \alpha_2 = 2$. Aday: $2^3 \cdot 3^2 = 8 \cdot 9 = 72$.
- $12 = 3 \cdot 2 \cdot 2 \implies \alpha_1 = 2, \alpha_2 = 1, \alpha_3 = 1$. Aday: $2^2 \cdot 3^1 \cdot 5^1 = 4 \cdot 3 \cdot 5 = 60$.

Bulunan adaylar $(2048, 96, 72, 60)$ arasında en küçüğü 60'tır. Gerçekten de $60 = 2^2 \cdot 3^1 \cdot 5^1$ için $d(60) = (2+1)(1+1)(1+1) = 12$ olur. □

**Örnek 12.26.** *$n$ pozitif bir tamsayı olmak üzere, $d(n)$ tek sayıdır ancak ve ancak $n$ bir tam karedir. Kanıtlayınız.*

*Çözüm.* **1. Yol (Cebirsel Açılım):** $n = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_k^{\alpha_k}$ olsun.

$$d(n) = (\alpha_1 + 1)(\alpha_2 + 1) \dots (\alpha_k + 1)$$

Bir çarpımın tek sayı olması ancak ve ancak tüm çarpanlarının tek sayı olmasıyla mümkündür:

$$\alpha_i + 1 \text{ tektir} \iff \alpha_i \text{ çifttir} \quad (\forall i = 1, 2, \dots, k)$$

Tüm $\alpha_i$ üsleri çift sayı ise, $\alpha_i = 2\beta_i$ yazabiliriz:

$$n = p_1^{2\beta_1} p_2^{2\beta_2} \dots p_k^{2\beta_k} = \left(p_1^{\beta_1} p_2^{\beta_2} \dots p_k^{\beta_k}\right)^2$$

Bu da $n$'nin bir tam kare olduğunu gösterir.

**2. Yol (Bölüm 10'daki Eşleme Yöntemi):** $n$'nin herhangi bir $d$ pozitif böleni verildiğinde $\frac{n}{d}$ de bir pozitif bölendir. Bölenleri $(d, \frac{n}{d})$ çiftleri halinde gruplayalım. Eğer $n$ bir tam kare değilse hiçbir zaman $d = \frac{n}{d}$ (yani $d^2 = n$) olamaz. Dolayısıyla tüm bölenler ikişerli eşleşir ve bölen sayısı kesinlikle çift kalır. Yalnızca $n$ bir tam kare olduğunda $d = \sqrt{n}$ böleni kendisiyle eşleşerek tek başına kalır; bu da toplam bölen sayısını tek yapar. □

**Örnek 12.27.** $x^2 - y^2 = 2024$ *denklemini sağlayan kaç farklı* $(x, y)$ *pozitif tamsayı ikilisi vardır?*

*Çözüm.* İki kare farkı özdeşliğini uygulayarak $(x - y)(x + y) = 2024$ yazalım. Burada iki temel noktaya dikkat edilmelidir:

1. $x, y \in \mathbb{Z}^+$ olduğundan $x + y > x - y > 0$ olmalıdır.
2. Bölüm 17'de ele aldığımız parite kontrolü: $(x+y)+(x-y) = 2x$ (çift) olduğundan, $x + y$ ve $x - y$ çarpanlarının teklik-çiftlik durumları aynı olmalıdır (ya ikisi de tek ya ikisi de çift).

2024 sayısını asal çarpanlarına ayıralım:

$$2024 = 2^3 \cdot 11 \cdot 23$$

Çarpımları 2024 olan ve ikisi de çift olan $A = x - y$ ve $B = x + y$ sayılarını arıyoruz. Çarpanlar çift olduğundan $A = 2u$ ve $B = 2v$ yazabiliriz:

$$(2u)(2v) = 2024 \implies 4uv = 2024 \implies uv = 506$$

506 sayısının asal çarpanları:

$$506 = 2 \cdot 11 \cdot 23$$

Şimdi $uv = 506$ ve $u < v$ ($x - y < x + y$ olduğundan) koşulunu sağlayan $(u, v)$ çiftlerinin sayısını belirleyelim. 506'nın pozitif bölen sayısı:

$$d(506) = (1 + 1)(1 + 1)(1 + 1) = 8$$

506 bir tam kare olmadığından bölenler 4 çift oluşturur ($u < v$ koşulunu sağlayan tam 4 adet $(u, v)$ çifti vardır). Belirlenen her $(u, v)$ çifti için:

$$x - y = 2u, \quad x + y = 2v \implies x = u + v, \quad y = v - u$$

tek biçimde pozitif tam sayı çözümler üretir ($v > u \geq 1$ olduğundan $x, y > 0$ sağlanır). Dolayısıyla denklemin tam 4 farklı pozitif tam sayı çözümü vardır. □

**Örnek 12.28.** $12^n \cdot 18^{2n}$ *sayısının pozitif bölen sayısı* 2091 *olduğuna göre,* $n$ *pozitif tamsayısını bulunuz.*

*Çözüm.* Bölen sayısı formülünü uygulayabilmek için sayıyı önce asal tabanlarda, yani kanonik biçimde yazmalıyız:

Verilen sayıyı $A$ olarak adlandıralım:

$$A = 12^n \cdot 18^{2n}$$

12 ve 18'i asal çarpanlarına ayıralım:

$$12 = 2^2 \cdot 3, \quad 18 = 2 \cdot 3^2$$

Yerine yazıp üsleri toplayalım:

$$\begin{aligned} A &= (2^2 \cdot 3)^n \cdot (2 \cdot 3^2)^{2n} \\ &= (2^{2n} \cdot 3^n) \cdot (2^{2n} \cdot 3^{4n}) \\ &= 2^{2n+2n} \cdot 3^{n+4n} \\ &= 2^{4n} \cdot 3^{5n} \end{aligned}$$

Artık kanonik formdayız. Pozitif bölen sayısı formülünü uygulayabiliriz:

$$d(A) = (4n+1)(5n+1)$$

Soruda bu değerin 2091 olduğu verilmiştir:

$$(4n+1)(5n+1) = 2091$$

İfadeyi açıp ikinci dereceden denklemi çözelim:

$$(4n+1)(5n+1) = 2091 \implies 20n^2 + 9n + 1 = 2091 \implies 20n^2 + 9n - 2090 = 0$$

Bu ifade çarpanlarına ayrılırsa:

$$(n-10)(20n+209) = 0$$

elde edilir (kontrol: $20n^2 + 209n - 200n - 2090 = 20n^2 + 9n - 2090$). $n$ pozitif olduğundan $20n + 209 = 0$ kökü elenir ve $n = 10$ bulunur.

Sağlama: $n = 10$ için $(4n+1)(5n+1) = 41 \times 51 = 2091$. □

**Örnek 12.29.** *$A = \frac{1000!}{500! \cdot 500!} = \binom{1000}{500}$ binom katsayısının 7'ye bölünebilen en büyük kuvvetini bulunuz. Yani $v_7\left(\binom{1000}{500}\right)$ değerini hesaplayınız.*

*Çözüm.* Bölüm 5 ve 6'da incelediğimiz kombinasyon ifadelerinin çarpan yapısını belirlerken bölme kuralı gereği asal üslerin farkını alırız:

$$v_7\left(\frac{1000!}{(500!)^2}\right) = v_7(1000!) - 2 \cdot v_7(500!)$$

Legendre formülünü kullanarak $v_7(1000!)$ ve $v_7(500!)$ değerlerini ayrı ayrı hesaplayalım. 7'nin kuvvetleri: $7^1 = 7$, $7^2 = 49$, $7^3 = 343$, $7^4 = 2401 > 1000$.

Önce $v_7(1000!)$'i bulalım:

$$\begin{aligned} v_7(1000!) &= \left\lfloor \frac{1000}{7} \right\rfloor + \left\lfloor \frac{1000}{49} \right\rfloor + \left\lfloor \frac{1000}{343} \right\rfloor \\ &= 142 + 20 + 2 = 164 \end{aligned}$$

Şimdi $v_7(500!)$'i bulalım:

$$\begin{aligned} v_7(500!) &= \left\lfloor \frac{500}{7} \right\rfloor + \left\lfloor \frac{500}{49} \right\rfloor + \left\lfloor \frac{500}{343} \right\rfloor \\ &= 71 + 10 + 1 = 82 \end{aligned}$$

Şimdi farkı alalım:

$$v_7\left(\binom{1000}{500}\right) = 164 - 2 \cdot (82) = 164 - 164 = 0$$

Böylece $\binom{1000}{500}$ sayısı 7 ile bölünemez; bu asalın en büyük kuvveti $7^0 = 1$'dir. □

# Bölüm 13

# EBOB ve EKOK

Tamsayılar teorisinde ve algoritma tasarımında en temel yapı taşlarından biri, iki veya daha fazla sayının ortak çarpanlarını ve ortak katlarını incelemektir. İster bir kesri en sade haline getirmek, ister periyodik çalışan iki sistemin aynı anda ne zaman tetikleneceğini belirlemek, isterse kriptografik anahtarlar üretmek olsun; karşımıza daima iki temel kavram çıkar: **En Büyük Ortak Bölen** (EBOB) ve **En Küçük Ortak Kat** (EKOK).

Bölüm 11'de ele aldığımız bölünebilme bağıntısı ve Bölüm 12'de gördüğümüz Aritmetiğin Temel Teoremi (asal çarpanlara ayrılışın tekliği), sayıların iç yapısını inceleme imkânı vermişti. Bu konuda, sayıların bu iç yapılarını birbiriyle kıyaslayacak, aralarındaki ortak bölenleri ve katları belirleyen kuralları kuracağız. Ardından, antik çağlardan bu yana kullanılan en etkili ve hızlı yöntemlerden biri olan **Öklid algoritması** ile tanışıp hesaplama karmaşıklığını inceleyeceğiz.

## 13.1 Tanımlar ve Özellikler

Soyut cebirsel tanımlara geçmeden önce, somut bir problemle başlayalım.

Elinizde 24 cm $\times$ 36 cm boyutlarında dikdörtgen şeklinde bir karton olduğunu düşünün. Bu kartonu, hiç parça artmayacak şekilde birbiriyle özdeş kare parçalara ayırmak istiyorsunuz. Seçeceğiniz karenin kenar uzunluğu tamsayı cinsinden kaç santimetre olabilir?

Bir karenin kenar uzunluğu $d$ olsun. Kartonun tamamen kaplanabilmesi için $d$ uzunluğunun hem 24'ü hem de 36'yı tam bölmesi gerekir:

$$d \mid 24 \quad \text{ve} \quad d \mid 36$$

24'ün pozitif bölenleri kümesi:

$$\mathcal{B}_{24} = \{1, 2, 3, 4, 6, 8, 12, 24\}$$

36'nın pozitif bölenleri kümesi:

$$\mathcal{B}_{36} = \{1, 2, 3, 4, 6, 9, 12, 18, 36\}$$

Her iki sayıyı da aynı anda bölen sayılar, bu iki kümenin arakesitidir:

$$\mathcal{B}_{24} \cap \mathcal{B}_{36} = \{1, 2, 3, 4, 6, 12\}$$

Demek ki karemizin kenarı $1, 2, 3, 4, 6$ veya 12 cm olabilir. Eğer amacımız en az sayıda kare elde etmekse, karenin kenarını mümkün olan en büyük değer seçmeliyiz; bu değer de kümenin en büyük elemanı olan 12'dir.

Şimdi soruyu tersine çevirelim: Biri 4 dakikada bir, diğeri 6 dakikada bir çalan iki çalar saat aynı anda çaldıktan sonra, tekrar ilk kez kaçıncı dakikada birlikte çalarlar?

Birinci saat 4, 8, 12, 16, 20, 24, ... dakikalarda çalar. İkinci saat 6, 12, 18, 24, 30, ... dakikalarda çalar. Birlikte çaldıkları dakikalar, 4 ve 6'nın pozitif ortak katlarıdır: $\{12, 24, 36, 48, \dots\}$ Bu ortak katların en küçüğü ise 12'dir.

Bu iki somut gözlem bizi EBOB ve EKOK kavramlarının matematiksel tanımlarına götürür.

**Tanım 13.1** (En Büyük Ortak Bölen)**.** *$a$ ve $b$, en az biri sıfırdan farklı iki tamsayı olsun. Hem $a$'yı hem de $b$'yi bölen en büyük pozitif tamsayıya, $a$ ve $b$'nin* ***en büyük ortak böleni*** *denir ve* $\gcd(a, b)$ *ile gösterilir:*

$$\gcd(a, b) = \max\{d \in \mathbb{Z}^+ : d \mid a \text{ ve } d \mid b\}$$

Özel olarak:

- Her $a \neq 0$ tamsayısı için $\gcd(a, 0) = |a|$ kabul edilir; çünkü her $d$ tamsayısı 0'ı böler ($0 = d \cdot 0$), dolayısıyla ortak bölenler yalnızca $a$'nın bölenleridir.
- $\gcd(0, 0)$ tanımsızdır (veya bazı cebirsel bağlamlarda 0 kabul edilir), zira her tamsayı 0'ı böldüğünden sonlu bir maksimum bulunamaz.
- $\gcd(a, b) = \gcd(|a|, |b|)$ olduğundan, aksi belirtilmedikçe $a, b > 0$ pozitif tamsayılarını incelemek yeterlidir.

**Tanım 13.2** (Aralarında Asallık)**.** *Eğer* $\gcd(a, b) = 1$ *ise, $a$ ve $b$ sayılarına* ***aralarında asal*** *(coprime / relatively prime) sayılar denir. Bu durum, $a$ ve $b$'nin* 1*'den başka pozitif ortak böleni olmadığı anlamına gelir.*

**Tanım 13.3** (En Küçük Ortak Kat)**.** *$a$ ve $b$ sıfırdan farklı iki tamsayı olsun. Hem $a$'nın hem de $b$'nin katı olan en küçük pozitif tamsayıya, $a$ ve $b$'nin* ***en küçük ortak katı*** *denir ve* $\operatorname{lcm}(a, b)$ *ile gösterilir:*

$$\operatorname{lcm}(a, b) = \min\{m \in \mathbb{Z}^+ : a \mid m \text{ ve } b \mid m\}$$

### Asal Çarpanlar Perspektifi

Bölüm 12'de gördüğümüz Aritmetiğin Temel Teoremi uyarınca, 1'den büyük her tamsayı asal çarpanlarının çarpımı olarak tek bir biçimde yazılabilir. $a$ ve $b$ sayılarını ortak asal tabanlar kullanarak ifade edelim:

$$a = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}, \qquad b = p_1^{\beta_1} p_2^{\beta_2} \cdots p_k^{\beta_k}$$

Burada $\alpha_i, \beta_i \geq 0$ tamsayılardır (bir asal sayı yalnızca birinde bulunuyorsa, diğerinde üssü 0 alınır).

Bir $d$ sayısının hem $a$'yı hem de $b$'yi bölebilmesi için, $d$'nin içerdiği her $p_i$ asalının üssü hem $\alpha_i$'den hem de $\beta_i$'den büyük olamaz. Dolayısıyla $d$'nin alabileceği en büyük asal üssü $\min(\alpha_i, \beta_i)$ olur.

Benzer şekilde, bir $m$ sayısının hem $a$'nın hem de $b$'nin katı olabilmesi için, $m$'nin içerdiği her $p_i$ asalının üssü hem $\alpha_i$'den hem de $\beta_i$'den küçük olamaz. Bu nedenle $m$'nin alabileceği en küçük asal üssü $\max(\alpha_i, \beta_i)$ olur.

**Teorem 13.4** (Asal Çarpanlar ile EBOB ve EKOK)**.** *$a = \prod_{i=1}^{k} p_i^{\alpha_i}$ ve $b = \prod_{i=1}^{k} p_i^{\beta_i}$ olmak üzere:*

$$\gcd(a, b) = \prod_{i=1}^{k} p_i^{\min(\alpha_i, \beta_i)}$$

$$\operatorname{lcm}(a, b) = \prod_{i=1}^{k} p_i^{\max(\alpha_i, \beta_i)}$$

Bu teoremin doğrudan bir sonucu olarak, EBOB ve EKOK'un temel bir özelliğini elde ederiz: *En büyük ortak bölen, yalnızca büyüklük bakımından en büyük olan bölen değildir; diğer tüm ortak bölenlerin de bir katıdır.*

**Teorem 13.5** (Ortak Bölenlerin Bölünebilme Özelliği)**.** *$a$ ve $b$ tamsayıları için bir $d$ tamsayısı hem $a$'yı hem $b$'yi bölüyorsa, $d$ mutlaka $\gcd(a, b)$'yi de böler:*

$$d \mid a \text{ ve } d \mid b \implies d \mid \gcd(a, b)$$

*Benzer biçimde, $a$ ve $b$'nin her ortak katı $m$ için:*

$$a \mid m \text{ ve } b \mid m \implies \operatorname{lcm}(a, b) \mid m$$

*Kanıt.* $a$ ve $b$'nin asal çarpan ayrışımlarını Teorem 13.4'deki gibi yazalım. $d \mid a$ ve $d \mid b$ ise, $d$'nin asal çarpanlarındaki herhangi bir $p_i$ asalının üssü $\gamma_i$ olsun. $d \mid a \implies \gamma_i \leq \alpha_i$ ve $d \mid b \implies \gamma_i \leq \beta_i$ olduğundan $\gamma_i \leq \min(\alpha_i, \beta_i)$ elde edilir. Bu da doğrudan $d \mid \gcd(a, b)$ demektir. Ortak katlar için ispat benzer şekilde max fonksiyonu kullanılarak yapılır. □

**Teorem 13.6** (EBOB'un Temel Cebirsel Özellikleri)**.** *Her $a, b, c, m \in \mathbb{Z}^+$ için:*

1. *Değişme özelliği:* $\gcd(a,b) = \gcd(b,a)$
2. *Birleşme özelliği:* $\gcd(a,\gcd(b,c)) = \gcd(\gcd(a,b),c) = \gcd(a,b,c)$
3. *Dağılma (ölçekleme) özelliği:* $\gcd(m\cdot a, m\cdot b) = m\cdot\gcd(a,b)$
4. *Sadeleştirme: Eğer* $d = \gcd(a,b)$ *ise,* $\gcd\left(\frac{a}{d},\frac{b}{d}\right) = 1$.

*Kanıt.* (3) özelliğini gösterelim: $a$ ve $b$'nin asal çarpanlarındaki üsler $\alpha_i, \beta_i$ ve $m$'nin asal çarpan üssü $k_i$ olsun. $m\cdot a$'daki asal üs $k_i+\alpha_i$, $m\cdot b$'deki asal üs $k_i+\beta_i$'dir. Herhangi iki reel sayı için $\min(k+x,k+y) = k+\min(x,y)$ özdeşliği geçerli olduğundan:

$$\min(k_i+\alpha_i, k_i+\beta_i) = k_i + \min(\alpha_i,\beta_i)$$

Her asal çarpan için bu eşitlik sağlandığından $\gcd(m\cdot a, m\cdot b) = m\cdot\gcd(a,b)$ bulunur.

(4) özelliği ise (3)'te $m = d$ ve $a' = a/d, b' = b/d$ konularak hemen görülür: $d = \gcd(d\cdot a', d\cdot b') = d\cdot\gcd(a',b') \implies \gcd(a',b') = 1$. □

## Çözümlü Örnekler

**Örnek 13.7.** *$a = 720$ ve $b = 168$ sayılarının asal çarpanlarını kullanarak* $\gcd(a,b)$ *ve* $\operatorname{lcm}(a,b)$ *değerlerini hesaplayınız.*

*Çözüm.* İlk olarak iki sayıyı da asal çarpanlarına ayıralım:

$$720 = 72\cdot 10 = 2^3\cdot 3^2\cdot 2\cdot 5 = 2^4\cdot 3^2\cdot 5^1\cdot 7^0$$

$$168 = 8\cdot 21 = 2^3\cdot 3^1\cdot 5^0\cdot 7^1$$

Ortak asalların üslerinin minimumunu alarak EBOB'u bulalım:

$$\gcd(720,168) = 2^{\min(4,3)}\cdot 3^{\min(2,1)}\cdot 5^{\min(1,0)}\cdot 7^{\min(0,1)} = 2^3\cdot 3^1\cdot 5^0\cdot 7^0 = 8\cdot 3 = 24$$

Ortak asalların üslerinin maksimumunu alarak EKOK'u bulalım:

$$\operatorname{lcm}(720,168) = 2^{\max(4,3)}\cdot 3^{\max(2,1)}\cdot 5^{\max(1,0)}\cdot 7^{\max(0,1)} = 2^4\cdot 3^2\cdot 5^1\cdot 7^1 = 16\cdot 9\cdot 5\cdot 7 = 5040$$

□

**Örnek 13.8.** *$n$ bir pozitif tamsayı olmak üzere,* $\gcd(n,n+2)$ *ifadesinin alabileceği değerleri bulunuz.*

*Çözüm.* İki sayıyı bölen bir $d$ tamsayısı, bu sayıların tamsayı katsayılı her lineer kombinasyonunu (farkını, toplamını) da bölmek zorundadır. $n$'li terimi yok ederek geriye sabit bir değer bırakalım.

$d = \gcd(n, n+2)$ olsun. Tanım gereği:

$$d \mid n \quad \text{ve} \quad d \mid (n+2)$$

Bölünebilmenin lineerlik kuralından, $d$ bu iki sayının farkını da bölmelidir:

$$d \mid (n+2) - n \implies d \mid 2$$

2'nin pozitif bölenleri yalnızca 1 ve 2'dir. O halde $d \in \{1, 2\}$ olabilir.

- Eğer $n$ tek sayı ise, $n$ ve $n+2$ de tek sayıdır. İki tek sayının ortak böleni çift olamaz, dolayısıyla $\gcd(n, n+2) = 1$.
- Eğer $n$ çift sayı ise, $2 \mid n$ ve $2 \mid (n+2)$ olduğundan 2 bir ortak bölendir. $d \mid 2$ olduğundan $\gcd(n, n+2) = 2$.

Sonuç olarak ifade, $n$'in tek veya çift olmasına göre 1 veya 2 değerini alır. □

**Örnek 13.9.** $\gcd(a, b) = 1$ *ise,* $\gcd(a+b, a-b)$ *ifadesinin yalnızca* 1 *veya* 2 *olabileceğini kanıtlayınız.*

*Çözüm.* $a+b$ ve $a-b$ ifadelerini taraf tarafa toplayıp çıkararak $a$ ve $b$'yi yalnız bırakmaya çalışalım.

$g = \gcd(a+b, a-b)$ olsun. O halde:

$$g \mid (a+b) \quad \text{ve} \quad g \mid (a-b)$$

Taraf tarafa toplarsak:

$$g \mid (a+b) + (a-b) \implies g \mid 2a$$

Taraf tarafa çıkarırsak:

$$g \mid (a+b) - (a-b) \implies g \mid 2b$$

$g$ sayısı hem $2a$'yı hem de $2b$'yi böldüğüne göre, Teorem 13.5 gereğince bu iki sayının en büyük ortak bölenini de bölmek zorundadır:

$$g \mid \gcd(2a, 2b)$$

Ölçekleme özelliğini kullanırsak:

$$\gcd(2a, 2b) = 2 \cdot \gcd(a, b) = 2 \cdot 1 = 2$$

Böylece $g \mid 2$ elde edilir. Pozitif bölenler 1 ve 2 olduğundan, $g$ yalnızca 1 veya 2 olabilir. □

**Örnek 13.10.** $\gcd(x,y) = 12$ *ve* $x + y = 180$ *şartlarını sağlayan kaç farklı* $(x,y)$ *pozitif tamsayı ikilisi vardır?*

*Çözüm.* $\gcd(x,y) = 12$ koşulu, $x$ ve $y$'nin 12'nin birer katı olduğunu belirtir. Bu çarpanı dışarı alırsak kalan bileşenler aralarında asal olmalıdır.

$x = 12a$ ve $y = 12b$ diyelim. Burada EBOB'un tanımı gereği $\gcd(a,b) = 1$ olmalıdır. Verilen toplam denkleminde yerine yazalım:

$$12a + 12b = 180 \implies 12(a+b) = 180 \implies a + b = 15$$

Şimdi toplamları 15 olan ve aralarında asal pozitif $(a,b)$ tamsayı çiftlerini listeleyelim:

$$\begin{aligned}
(a,b) = (1,14) &\implies \gcd(1,14) = 1 \quad \text{(uygun)} \\
(a,b) = (2,13) &\implies \gcd(2,13) = 1 \quad \text{(uygun)} \\
(a,b) = (3,12) &\implies \gcd(3,12) = 3 \neq 1 \quad \text{(uygun değil)} \\
(a,b) = (4,11) &\implies \gcd(4,11) = 1 \quad \text{(uygun)} \\
(a,b) = (5,10) &\implies \gcd(5,10) = 5 \neq 1 \quad \text{(uygun değil)} \\
(a,b) = (6,9) &\implies \gcd(6,9) = 3 \neq 1 \quad \text{(uygun değil)} \\
(a,b) = (7,8) &\implies \gcd(7,8) = 1 \quad \text{(uygun)}
\end{aligned}$$

$(a,b)$ sırasını göz önüne alırsak, simetrik çiftler de geçerlidir: $(8,7)$, $(11,4)$, $(13,2)$ ve $(14,1)$. Toplamda $4 \times 2 = 8$ farklı $(a,b)$ çifti, dolayısıyla 8 farklı $(x,y)$ ikilisi vardır. □

**En Sık Yapılan Hata:** Genellikle $x = 12a$ ve $y = 12b$ dönüşümü yapıldıktan sonra $a+b = 15$ denklemini sağlayan tüm pozitif $(a,b)$ çiftleri sayılır (14 adet). Ancak eğer $\gcd(a,b) > 1$ olursa, $\gcd(x,y) = 12 \cdot \gcd(a,b) > 12$ çıkacağından başlangıçtaki koşul bozulur. $\gcd(a,b) = 1$ kısıtlaması mutlaka denetlenmelidir.

## 13.2 Öklid Algoritması

Asal çarpanlara ayırma yöntemi teorik açıdan açık olsa da büyük sayılarda pratik bir engele çarpar: **Büyük sayıları asal çarpanlarına ayırmak son derece zordur.**

Örneğin yüz basamaklı iki sayının asal çarpanlarını bulmak günümüz süper bilgisayarlarıyla dahi pratik zaman sınırları içinde mümkün değildir (modern RSA şifreleme sistemleri de bu zorluğa dayanır). Peki asal çarpanlarını bilmediğimiz büyük sayıların EBOB'unu hızlıca bulabilir miyiz?

Bu sorunun yanıtı, M.Ö. 300 civarında İskenderiyeli Öklid tarafından verilmiştir.

### Çıkarma Yoluyla EBOB ve Bölme Adımı

EBOB'un temel bir özelliği şudur: Eğer bir $d$ sayısı $a$'yı ve $b$'yi $(a > b)$ bölüyorsa, farkları olan $a - b$'yi de bölmek zorundadır. Dahası, $b$ ile $a - b$'nin ortak bölenleri kümesi ile $a$ ile $b$'nin ortak bölenleri kümesi tamamen aynıdır.

**Teorem 13.11.** *Her $a, b \in \mathbb{Z}$ için:*

$$\gcd(a, b) = \gcd(b, a - b)$$

*Kanıt.* $d \mid a$ ve $d \mid b$ olsun. O halde $d \mid (a - b)$ olur. Dolayısıyla $a$ ve $b$'nin her ortak böleni, $b$ ve $a - b$'nin de bir ortak bölenidir. Tersine, $c \mid b$ ve $c \mid (a - b)$ olsun. O halde $c \mid b + (a - b) = a$ olur. Dolayısıyla $b$ ve $a - b$'nin her ortak böleni, $a$ ve $b$'nin de bir ortak bölenidir. Her iki çiftin ortak bölenler kümesi özdeş olduğundan, bu kümelerin en büyük elemanları da aynı olmak zorundadır: $\gcd(a, b) = \gcd(b, a - b)$. □

Bu adımı tekrarlayarak $a$'dan $b$'yi art arda çıkarabiliriz:

$$\gcd(a, b) = \gcd(b, a - b) = \gcd(b, a - 2b) = \ldots = \gcd(b, a - qb)$$

Bölme algoritmasından bildiğimiz üzere $a$'nın $b$'ye bölümünden kalan $r = a - qb$ $(0 \leq r < b)$ olduğuna göre, bu işlemi tek bir bölme adımıyla gerçekleştirebiliriz:

**Teorem 13.12** (Öklid Algoritmasının Temel Adımı)**.** *$a, b \in \mathbb{Z}^+$ ve $a = q \cdot b + r$ $(0 \leq r < b)$ olsun. O halde:*

$$\gcd(a, b) = \gcd(b, r)$$

*Kanıt.* $g_1 = \gcd(a, b)$ ve $g_2 = \gcd(b, r)$ olsun.

1. $g_1 \mid a$ ve $g_1 \mid b$ olduğundan, $r = a - qb$ ifadesi $a$ ve $b$'nin lineer kombinasyonudur; dolayısıyla $g_1 \mid r$. Hem $b$'yi hem $r$'yi böldüğü için, Teorem 13.5 uyarınca $g_1 \mid \gcd(b, r)$, yani $g_1 \mid g_2$. Pozitif olduklarından $g_1 \leq g_2$.

2. $g_2 \mid b$ ve $g_2 \mid r$ olduğundan, $a = qb + r$ ifadesinden ötürü $g_2 \mid a$. Hem $a$'yı hem $b$'yi böldüğü için $g_2 \mid \gcd(a, b)$, yani $g_2 \mid g_1$. Buradan $g_2 \leq g_1$.

$g_1 \leq g_2$ ve $g_2 \leq g_1$ eşitsizlikleri ancak $g_1 = g_2$ durumunda sağlanır. □

Algoritma şu şekilde işler: Kalan $r = 0$ olana kadar $(a, b) \leftarrow (b, a \bmod b)$ dönüşümü yinelenir. Kalan 0 olduğunda, son sıfırdan farklı kalan aradığımız $\gcd(a, b)$ değeridir; çünkü $\gcd(g, 0) = g$'dir.

**Somut Bir Hesaplama:** $\gcd(1071, 462)$

Öklid algoritmasını adım adım uygulayalım:

$$\begin{aligned} 1071 &= 2 \cdot 462 + 147 && \Longrightarrow \gcd(1071, 462) = \gcd(462, 147) \\ 462 &= 3 \cdot 147 + 21 && \Longrightarrow \gcd(462, 147) = \gcd(147, 21) \\ 147 &= 7 \cdot 21 + 0 && \Longrightarrow \gcd(147, 21) = \gcd(21, 0) = 21 \end{aligned}$$

Yalnızca üç bölme işleminde $\gcd(1071, 462) = 21$ sonucuna ulaştık; hiçbir asal çarpan hesabı yapmamız gerekmedi.

### Algoritmanın Karmaşıklığı ve Çalışma Zamanı

Bir algoritmanın kaç adımda sonlandığını bilmek, özellikle olimpiyat ve yarışma programlamasında belirleyicidir.

**Teorem 13.13** (Kalanın Küçülme Hızı)**.** *Eğer $a \geq b > 0$ ve $r = a \bmod b$ ise, $r < \frac{a}{2}$ olur.*

*Kanıt.* İki durumu inceleyelim:

- Durum 1: $b \leq \frac{a}{2}$. Bölme algoritması gereği $r < b$ olduğundan, doğrudan $r < \frac{a}{2}$ elde edilir.
- Durum 2: $b > \frac{a}{2}$. Bu durumda $b$, $a$'nın içinde en fazla 1 defa bulunabilir ($q = 1$). O halde kalan $r = a - b < a - \frac{a}{2} = \frac{a}{2}$ olur.

Her iki durumda da $r < \frac{a}{2}$ eşitsizliği kesinlikle sağlanır. □

Bu teorem, her iki adımda bir sayıların en az yarı yarıya küçüldüğünü gösterir. Dolayısıyla $N = \max(a, b)$ olmak üzere algoritma en fazla $2\log_2(N)$ adımda biter. Yani Öklid algoritmasının zaman karmaşıklığı:

$$O(\log(\min(a, b)))$$

mertebesindedir. 64-bitlik büyük bir tamsayı ($a \approx 10^{18}$) için bile Öklid algoritması en fazla 90–100 bölme işleminde tamamlanır.

**En Kötü Durum (Lamé Teoremi):** Öklid algoritmasında bölümlerin sürekli $q = 1$ çıktığı en yavaş durum, Bölüm 9'da incelediğimiz ardışık Fibonacci sayıları ile ($F_{n+1}$ ve $F_n$) karşılaşınca oluşur. Fransız matematikçi Gabriel Lamé, basamak sayısı $k$ olan iki sayının EBOB'unun en fazla $5k$ adımda bulunabileceğini ispatlamıştır.

### Kod Gerçekleştirmeleri

Öklid algoritması hem rekürsif (rekürsif) hem de döngüsel (iteratif) olarak sade bir biçimde kodlanabilir.

```
FUNCTION gcd(a, b)
    WHILE b != 0 DO
        remainder ← a MOD b
        a ← b
        b ← remainder
    END WHILE
    RETURN ABSOLUTE_VALUE(a)
END FUNCTION
```

Aynı algoritmanın C dilindeki rekürsif ve döngüsel karşılıkları:

```c
/* Rekursif yontem */
long long gcd_rec(long long a, long long b) {
    return (b == 0) ? a : gcd_rec(b, a % b);
}

/* Döngüsel (iteratif) yöntem */
long long gcd_iter(long long a, long long b) {
    while (b != 0) {
        long long r = a % b;
        a = b;
        b = r;
    }
    return a;
}
```

### Çözümlü Örnekler

**Örnek 13.14.** *Her $n$ pozitif tamsayısı için* $\gcd(5n+3, 3n+2)$ *ifadesinin değerini bulunuz.*

*Çözüm.* Öklid algoritmasının $\gcd(A, B) = \gcd(B, A - B)$ kuralını kullanarak $n$'li terimlerin katsayılarını sıfırlayana kadar çıkarma adımlarını yürütelim:

$$\begin{aligned}\gcd(5n+3, 3n+2) &= \gcd(3n+2, (5n+3) - (3n+2)) \\ &= \gcd(3n+2, 2n+1)\end{aligned}$$

Bir adım daha çıkarma uygulayalım:

$$\begin{aligned}\gcd(3n+2, 2n+1) &= \gcd(2n+1, (3n+2) - (2n+1)) \\ &= \gcd(2n+1, n+1)\end{aligned}$$

Bir kez daha uygulayalım:

$$\begin{aligned}\gcd(2n+1, n+1) &= \gcd(n+1, (2n+1)-(n+1)) \\ &= \gcd(n+1, n)\end{aligned}$$

Son olarak:

$$\gcd(n+1, n) = \gcd(n, (n+1)-n) = \gcd(n, 1) = 1$$

Demek ki $n$ hangi pozitif tamsayı olursa olsun $5n+3$ ve $3n+2$ daima aralarında asaldır; EBOB her zaman 1'dir. □

**Örnek 13.15** (Uluslararası Matematik Olimpiyatı 1959, Problem 1)**.** *Her $n$ pozitif tamsayısı için $\frac{21n+4}{14n+3}$ kesrinin sadeleştirilemez (indirgenemez) olduğunu kanıtlayınız.*

*Çözüm.* Bir kesrin sadeleştirilemez olması, pay ve paydasının aralarında asal olması, yani en büyük ortak bölenlerinin 1 olması anlamına gelir. $\gcd(21n+4, 14n+3) = 1$ olduğunu Öklid algoritması ile gösterelim:

$$21n+4 = 1 \cdot (14n+3) + (7n+1)$$

Buradan:

$$\gcd(21n+4, 14n+3) = \gcd(14n+3, 7n+1)$$

Şimdi $14n+3$ ifadesini $7n+1$'e bölelim:

$$14n+3 = 2 \cdot (7n+1) + 1$$

Kalan 1'dir. Teorem 13.12 uyarınca:

$$\gcd(14n+3, 7n+1) = \gcd(7n+1, 1) = 1$$

Pay ve paydanın EBOB'u 1 olduğundan kesir hiçbir $n$ değeri için sadeleştirilemez. □

**Örnek 13.16.** *$a$ ve $b$ pozitif tamsayılar olmak üzere,*

$$\gcd(2^a-1, 2^b-1) = 2^{\gcd(a,b)} - 1$$

*olduğunu kanıtlayınız.*

*Çözüm.* Genelliği bozmadan $a \geq b$ kabul edelim. Bölme algoritması gereği $a = qb + r$ yazabiliriz ($0 \leq r < b$). Şu cebirsel açılımı inceleyelim:

$$2^a - 1 = 2^{qb+r} - 1 = 2^r(2^{qb} - 1) + (2^r - 1)$$

Her pozitif $k$ için $(x-1) \mid (x^k - 1)$ olduğundan, $x = 2^b$ seçilirse:

$$(2^b - 1) \mid ((2^b)^q - 1) \implies (2^b - 1) \mid (2^{qb} - 1)$$

Dolayısıyla bir $M$ tamsayısı için $2^{qb} - 1 = M \cdot (2^b - 1)$ olur. Yerine yazarsak:

$$2^a - 1 = 2^r \cdot M \cdot (2^b - 1) + (2^r - 1)$$

Bu eşitlik, $2^a - 1$ sayısının $2^b - 1$'e bölümünden kalanın $2^r - 1$ olduğunu gösterir. Ortak bölenler bakımından:

$$\gcd(2^a - 1, 2^b - 1) = \gcd(2^b - 1, 2^r - 1)$$

Bu bağıntı, üslerde yürütülen Öklid adımının $(a, b) \to (b, r = a \bmod b)$ birebir tabana yansımasıdır. Öklid algoritması üslerde ilerleyip sıfırdan farklı son kalan olan $\gcd(a, b)$'ye ulaştığında, tabandaki ifade de $2^{\gcd(a,b)} - 1$ değerine ulaşır. □

## 13.3 $\gcd \cdot \operatorname{lcm} = ab$

EBOB hesaplamasını $O(\log n)$ karmaşıklığında yapmayı gördük. Peki iki büyük sayının EKOK'unu bulmak için tekrar asal çarpanlara dönmek gerekir mi?

İki sayının EBOB'u ile EKOK'u arasında doğrudan bir bağıntı bulunmaktadır; bu sayede EKOK hesabı da tek bir EBOB hesabına indirgenir.

### Temel Özdeşlik ve İspatı

**Teorem 13.17.** *Her $a, b \in \mathbb{Z}^+$ için:*

$$\gcd(a, b) \cdot \operatorname{lcm}(a, b) = a \cdot b$$

*Kanıt.* Herhangi iki gerçel $x$ ve $y$ sayısı için şu eşitlik daima geçerlidir:

$$\min(x, y) + \max(x, y) = x + y$$

(Sayıların hangisinin küçük ya da büyük olduğuna bakılmaksızın biri min'e, diğeri max'a karşılık gelir ve toplamları $x + y$ olur.)

Şimdi $a$ ve $b$'yi asal çarpanlarına ayıralım:

$$a = \prod_{i=1}^{k} p_i^{\alpha_i}, \qquad b = \prod_{i=1}^{k} p_i^{\beta_i}$$

Teorem 13.4 uyarınca:

$$\gcd(a, b) = \prod_{i=1}^{k} p_i^{\min(\alpha_i, \beta_i)}, \qquad \operatorname{lcm}(a, b) = \prod_{i=1}^{k} p_i^{\max(\alpha_i, \beta_i)}$$

Bu iki ifadeyi çarpalım:

$$\begin{aligned}\gcd(a,b)\cdot\operatorname{lcm}(a,b) &= \left(\prod_{i=1}^{k} p_i^{\min(\alpha_i,\beta_i)}\right)\cdot\left(\prod_{i=1}^{k} p_i^{\max(\alpha_i,\beta_i)}\right)\\ &= \prod_{i=1}^{k} p_i^{\min(\alpha_i,\beta_i)+\max(\alpha_i,\beta_i)}\\ &= \prod_{i=1}^{k} p_i^{\alpha_i+\beta_i}\\ &= \left(\prod_{i=1}^{k} p_i^{\alpha_i}\right)\cdot\left(\prod_{i=1}^{k} p_i^{\beta_i}\right) = a\cdot b\end{aligned}$$

Böylece ispat tamamlanır. □

Bu eşitlik sayesinde EKOK bulmak için ayrı bir algoritma yazmaya gerek kalmaz; Öklid algoritması ile $\gcd(a,b)$ hesaplandıktan sonra formül işletilir:

$$\operatorname{lcm}(a,b) = \frac{a\cdot b}{\gcd(a,b)}$$

### Programlamada Taşma (Overflow) Tuzağı

Formülü doğrudan `(a * b) / gcd(a, b)` biçiminde kodlamak yaygın bir hatadır.

$a$ ve $b$ sayıları $10^9$ mertebesinde olduğunda her ikisi de standart 32-bit işaretli tamsayı türüne (`int`) sığar. Ancak bu iki sayı önce çarpıldığında:

$$a\cdot b \approx 10^{18}$$

değeri 32-bit sınırını ($2^{31}-1 \approx 2\times10^9$) aşar ve aritmetik taşma (integer overflow) meydana gelir. Sonuç anlamsız veya negatif çıkabilir.

Hatta 64-bitlik `long long` türü kullanılsa dahi, $a,b \approx 10^{14}$ olduğunda çarpım $10^{28}$ değerine ulaşarak 64-bit sınırını ($\approx 9\times10^{18}$) da aşar.

**Çözüm:** $\gcd(a,b)$ sayısı tanım gereği $a$'yı kalansız böler. Bu nedenle çarpma işleminden önce bölme işlemini gerçekleştirmek taşmayı engeller:

$$\operatorname{lcm}(a,b) = \left(\frac{a}{\gcd(a,b)}\right)\cdot b$$

$\frac{a}{\gcd(a,b)}$ ifadesi daima bir tamsayıdır ve sayıyı küçülterek ara taşmaların önüne geçer.

```
long long lcm(long long a, long long b) {
    if (a == 0 || b == 0) return 0;
    /* Once bol, sonra carp: tasmayi onler! */
    return (a / gcd_iter(a, b)) * b;
}
```

## Üç veya Daha Fazla Sayıda Durum

**En Sık Yapılan Hata:** $\gcd(a,b)\cdot\text{lcm}(a,b) = a\cdot b$ bağıntısına dayanarak benzer eşitliğin üç sayı için de geçerli olacağı yanılgısına sıkça düşülür:

$$\gcd(a,b,c)\cdot\text{lcm}(a,b,c) \stackrel{?}{=} a\cdot b\cdot c \quad \textbf{(YANLIŞ!)}$$

Bu eşitlik genel olarak **geçersizdir. Karşı Örnek:** $a=2, b=2, c=2$ olsun. $\gcd(2,2,2)=2$ ve $\text{lcm}(2,2,2)=2$'dir. $\gcd\cdot\text{lcm} = 2\cdot 2 = 4$ iken $a\cdot b\cdot c = 2\cdot 2\cdot 2 = 8$'dir $(4\neq 8)$.
Benzer şekilde $a=2, b=3, c=6$ için $\gcd(2,3,6)=1$ ve $\text{lcm}(2,3,6)=6$'dır. $\gcd\cdot\text{lcm}=6$ iken $a\cdot b\cdot c = 36$'dır.

Bunun nedeni, üç gerçel sayı için genel olarak

$$\min(x,y,z)+\max(x,y,z) \neq x+y+z$$

olmasıdır; ortadaki değer (medyan) bu toplamda yer almaz.

Üç sayı için geçerli bağıntı şöyledir:

**Teorem 13.18** (Üç Sayı için EKOK Formülü)**.** *Her $a,b,c\in\mathbb{Z}^+$ için:*

$$\text{lcm}(a,b,c) = \frac{a\cdot b\cdot c\cdot\gcd(a,b,c)}{\gcd(a,b)\cdot\gcd(b,c)\cdot\gcd(c,a)}$$

*Kanıt.* Herhangi bir asal $p$ için $a,b,c$'deki üsler sırasıyla $x,y,z$ olsun. Genelliği bozmadan $x\leq y\leq z$ varsayabiliriz. Payın üssü:

$$x+y+z+\min(x,y,z) = x+y+z+x = 2x+y+z$$

Paydanın üssü:

$$\min(x,y)+\min(y,z)+\min(x,z) = x+y+x = 2x+y$$

Payın üssünden paydanın üssünü çıkarırsak:

$$(2x+y+z)-(2x+y) = z = \max(x,y,z)$$

Bu değer tam olarak $\text{lcm}(a,b,c)$ içindeki $p$ asalının üssüdür. Her asal için bu eşitlik sağlandığından teorem ispatlanmış olur. □

## Çözümlü Örnekler

**Örnek 13.19.** *$a+b=56$ ve $\text{lcm}(a,b)=105$ eşitliklerini sağlayan tüm $(a,b)$ pozitif tamsayı ikililerini bulunuz.*

*Çözüm.* $d = \gcd(a,b)$ tanımlarsak, $d$ sayısı hem $a$'yı hem $b$'yi böldüğü için toplamları olan $a+b = 56$'yı da bölmelidir. Aynı zamanda $d$, $\operatorname{lcm}(a,b) = 105$'in de bölenidir.

Böylece:

$$d \mid (a+b) \implies d \mid 56$$
$$d \mid \operatorname{lcm}(a,b) \implies d \mid 105$$

O halde $d$, 56 ve 105'in bir ortak bölenidir:

$$d \mid \gcd(56, 105)$$

Öklid algoritması ile $\gcd(56,105)$'i bulalım:

$$105 = 1 \cdot 56 + 49 \implies \gcd(105,56) = \gcd(56,49) = \gcd(49,7) = 7$$

Dolayısıyla $d \mid 7$'dir. Pozitif bölenler olarak $d = 1$ veya $d = 7$ olabilir.

*Durum 1:* $d = 1$ ise. $\gcd(a,b) = 1 \implies a \cdot b = \operatorname{lcm}(a,b) = 105$. Toplamları 56, çarpımları 105 olan iki sayı arayalım: $t^2 - 56t + 105 = 0$ denkleminin diskriminantı $\Delta = 56^2 - 4 \cdot 105 = 3136 - 420 = 2716$ tamkare olmadığından tamsayı çözümü yoktur.

*Durum 2:* $d = 7$ ise. $a = 7x$ ve $b = 7y$ diyelim; burada $\gcd(x,y) = 1$ olmalıdır. $a+b = 56 \implies 7(x+y) = 56 \implies x+y = 8$. $\operatorname{lcm}(a,b) = 7xy = 105 \implies xy = 15$.

Toplamları 8, çarpımları 15 olan sayılar 3 ve 5'tir: $\{x,y\} = \{3,5\}$. Aralarında asal olduklarını kontrol edelim: $\gcd(3,5) = 1$ koşulu sağlanır. Buradan $a$ ve $b$ değerleri:

$$\{a,b\} = \{7 \cdot 3, 7 \cdot 5\} = \{21, 35\}$$

Sıralı ikililer olarak çözümler $(21,35)$ ve $(35,21)$'dir. □

**Örnek 13.20.** $\operatorname{lcm}(a,b) - \gcd(a,b) = 103$ *eşitliğini sağlayan tüm* $(a,b)$ *pozitif tamsayı ikililerini bulunuz.*

*Çözüm.* $g = \gcd(a,b)$ diyelim. $a = gx$ ve $b = gy$ olsun ($\gcd(x,y) = 1$). Bu durumda $\operatorname{lcm}(a,b) = gxy$ olur. İfadeyi $g$ parantezine alalım:

$$gxy - g = 103 \implies g(xy-1) = 103$$

103 asal bir sayıdır; bu nedenle pozitif çarpanları yalnızca 1 ve 103'tür. Buradan iki olasılık çıkar:

*Olasılık 1:* $g = 103$ ve $xy - 1 = 1$. $xy - 1 = 1 \implies xy = 2$. $x$ ve $y$ pozitif tamsayılar olduğundan $\{x,y\} = \{1,2\}$ olmalıdır. Buradan:

$$\{a,b\} = \{103 \cdot 1, 103 \cdot 2\} = \{103, 206\}$$

Kontrol edelim: $\gcd(103,206) = 103$, $\operatorname{lcm}(103,206) = 206$. $206 - 103 = 103$ eşitliği sağlanır.

*Olasılık 2:* $g = 1$ ve $xy - 1 = 103$. $xy - 1 = 103 \implies xy = 104$. Ayrıca $g = 1$ olduğundan $\gcd(x, y) = 1$ olmalıdır. $104 = 2^3 \cdot 13 = 8 \cdot 13$. Çarpımları 104 olan ve aralarında asal $\{x, y\}$ çiftleri:

- $\{x, y\} = \{1, 104\} \implies \{a, b\} = \{1, 104\}$
- $\{x, y\} = \{8, 13\} \implies \{a, b\} = \{8, 13\}$

($\{2, 52\}$ ve $\{4, 26\}$ çiftlerinde $\gcd > 1$ olduğundan elenir.)

Sonuç olarak tüm $(a, b)$ ikilileri: $(103, 206)$, $(206, 103)$, $(1, 104)$, $(104, 1)$, $(8, 13)$ ve $(13, 8)$'dir. □

**Örnek 13.21.** $\gcd(x, y) = 24$ *ve* $\operatorname{lcm}(x, y) = 2496$ *koşulunu sağlayan kaç farklı* $(x, y)$ *pozitif tamsayı çifti vardır?*

*Çözüm.* $x = 24a$ ve $y = 24b$ dönüşümü yapalım ($\gcd(a, b) = 1$).

$$\operatorname{lcm}(x, y) = 24ab = 2496 \implies ab = \frac{2496}{24} = 104$$

Problem, çarpımları 104 olan ve $\gcd(a, b) = 1$ koşulunu sağlayan $(a, b)$ ikililerini saymaya indirgenir.

104'ü asal çarpanlarına ayıralım:

$$104 = 2^3 \cdot 13^1$$

$\gcd(a, b) = 1$ olması için, 104'ün asal çarpan blokları ($2^3$ ve $13^1$) ya bütünüyle $a$'ya ya da bütünüyle $b$'ye ait olmalıdır; bölünemezler. Burada 2 farklı asal blok vardır: $2^3$ ve $13^1$. Her blok için 2 seçenek mevcuttur (ya $a$'da ya da $b$'de yer alır). Dolayısıyla aralarında asal $(a, b)$ ikililerinin sayısı:

$$2^2 = 4$$

Bu çiftler $(a, b) \in \{(1, 104), (8, 13), (13, 8), (104, 1)\}$ olup her biri geçerli bir $(x, y)$ ikilisi üretir. Yanıt 4'tür. □

**Örnek 13.22.** $a, b, c$ *pozitif tamsayılar olmak üzere,* $\operatorname{lcm}(a, b) = \operatorname{lcm}(a + c, b - c)$ *ise* $c$*'nin alabileceği değerler hakkında ne söylenebilir?*

*Çözüm.* $L = \operatorname{lcm}(a, b) = \operatorname{lcm}(a + c, b - c)$ diyelim. EKOK tanımı gereği, her iki ikilideki sayılar $L$'yi böler:

$$a \mid L, \quad b \mid L, \quad (a + c) \mid L, \quad (b - c) \mid L$$

Uç durumları inceleyelim: Eğer $c = b$ olursa $b - c = 0$ olur ki bu pozitif tamsayılar kümesinde tanımsızdır. $a = b$ durumunda $L = a$ olur; o halde $(a + c) \mid a$ olmalıdır. Fakat pozitif $c$ için $a + c > a$ olduğundan bu imkânsızdır; dolayısıyla $a \neq b$ olmalıdır.

Böyle bir eşitliğin sağlanabileceğini somut bir değerle görelim: $a = 2, b = 3$ seçelim. $\text{lcm}(2,3) = 6$. $c = 1$ alırsak: $a + c = 2 + 1 = 3$, $b - c = 3 - 1 = 2$. $\text{lcm}(3,2) = 6$. Görüldüğü gibi $a = 2, b = 3, c = 1$ bir çözümdür.

Genel olarak, $a+c = b$ ve $b-c = a$ denklemleri aslında aynı denklemdir; ikincisi birincinin yalnızca yeniden düzenlenmiş halidir ve her ikisi de $c = b - a$ sonucunu verir. Bu seçimle $(a + c, b - c) = (b, a)$ olur; yani ikili yalnızca yer değiştirir ve $\text{lcm}(a,b) = \text{lcm}(b,a)$ olduğundan EKOK eşitliği apaçık sağlanır. Somut bir örnekle doğrularsak: $a = 2, b = 3$ için $c = b - a = 1$'dir ve gerçekten $(a + c, b - c) = (3, 2)$ olup ikili yer değiştirmiştir. Bu inceleme bize, buradaki çözümün ikilinin yalnızca yer değiştirmesi durumuna dayandığını gösterir. □

Bu bölümde kurduğumuz EBOB, EKOK ve Öklid algoritması temelleri, Bölüm 14'te ele alacağımız modüler aritmetik konusunun çekirdeğini oluşturacaktır.

# Bölüm 14

# Modüler Aritmetik

Gündelik yaşamda zamanı takip ederken farkında olmadan matematiğin en kullanışlı ve temel yapılarından birine başvururuz. Saat şu an 09 : 00'u gösteriyorsa, 5 saat sonra saatin 14 : 00 değil de 02 : 00 olduğunu söylemek son derece doğaldır. Benzer şekilde, günlerden salı ise 14 gün sonra yine salı olacağını hemen hesaplarız; çünkü haftanın günleri yedişer yedişer tekrarlanan bir döngüden ibarettir.

Olimpiyat matematiğinde ve bilgisayar biliminde büyük sayılarla işlem yaparken, sayıların kendisinden ziyade belirli bir periyoda göre bıraktıkları *kalanlar* önem kazanır. Örneğin $2^{1000}$ sayısının tüm basamaklarını hesaplamak ne kâğıt üzerinde ne de standart veri tipleriyle pratik bir iştir; oysa bu sayının 7 ile bölümünden kalanı bulmak yalnızca birkaç adımlık kısa bir işlem gerektirir.

Bölüm 11'de ele aldığımız bölünebilme kuralları ve bölen kavramı, sayıları tam bölen çarpanları inceliyordu. Burada ise bölünememenin geride bıraktığı fazlalıkları, yani kalanları sistematik bir cebirsel yapıya kavuşturacağız. Carl Friedrich Gauss'un temellerini attığı bu dile **modüler aritmetik** diyoruz.

## 14.1 Kongruans

### 14.1.1 Motivasyon ve Sezgi

Kavramın çıkış noktasını anlamak için önce sayılar arasındaki farklara odaklanalım. 17 ve 38 sayılarını göz önüne alalım. Bu iki sayıyı ayrı ayrı 7'ye böldüğümüzde:

$$17 = 2 \times 7 + 3$$

$$38 = 5 \times 7 + 3$$

Görüldüğü üzere her iki sayı da 7'ye bölündüğünde aynı kalanı (3) verir. Şimdi bu iki sayının farkına bakalım:

$$38 - 17 = 21 = 3 \times 7$$

Fark, 7'nin tam katıdır. Bu beklenen bir durumdur: İki sayı aynı kalana sahipse, bu sayılardan kalan kısımları çıkardığımızda geriye yalnızca bölene ait tam katlar kalır. Dolayısıyla farkları bölene tam bölünmek zorundadır.

Modüler aritmetiğin temel sezgisi şudur: Belli bir $m$ tam sayısı sabit tutulduğunda, $m$'ye bölündüklerinde aynı kalanı veren tüm tam sayılar "eşdeğer" kabul edilir.

**Tanım 14.1** (Kongruans (Denklik))**.** *$m \geq 1$ bir pozitif tam sayı ve $a, b \in \mathbb{Z}$ olsun. Eğer $m$ sayısı $a - b$ farkını tam bölüyorsa, yani $m \mid (a - b)$ ise, $a$ sayısı $b$ sayısına $m$* ***modülüne göre denktir (kongruanstır)*** *denir ve bu durum*

$$a \equiv b \pmod{m}$$

*biçiminde gösterilir. Buradaki $m$ değerine* ***modül*** *denir. Eğer $m \nmid (a-b)$ ise $a \not\equiv b \pmod{m}$ yazılır.*

Bu tanım, Bölüm 11'deki bölme algoritması gereği $a$ ile $b$'nin $m$'ye bölündüklerinde aynı kalanı vermesiyle tamamen eşdeğerdir. Gerçekten de $a = q_1 m + r_1$ ve $b = q_2 m + r_2$ $(0 \leq r_1, r_2 < m)$ yazılırsa:

$$a - b = (q_1 - q_2)m + (r_1 - r_2)$$

olur. $m \mid (a - b)$ olabilmesi için $m \mid (r_1 - r_2)$ olmalıdır. Ancak $-m < r_1 - r_2 < m$ olduğundan bu ancak ve ancak $r_1 - r_2 = 0$, yani $r_1 = r_2$ durumunda mümkündür.

### 14.1.2 Denklik Bağıntısı Olarak Kongruans

Kongruans simgesi ($\equiv$) sıradan eşitlik simgesine ($=$) kasıtlı olarak benzetilmiştir; zira kongruans, tam sayılar kümesi üzerinde bir **denklik bağıntısı** oluşturur.

**Teorem 14.2** (Kongruansın Temel Özellikleri)**.** *Her $a, b, c \in \mathbb{Z}$ ve $m \in \mathbb{Z}^+$ için aşağıdaki özellikler sağlanır:*

1. ***Yansıma (Reflexivity):*** $a \equiv a \pmod{m}$.
2. ***Simetri (Symmetry):*** $a \equiv b \pmod{m} \implies b \equiv a \pmod{m}$.
3. ***Geçişme (Transitivity):*** $a \equiv b \pmod{m}$ *ve* $b \equiv c \pmod{m} \implies a \equiv c \pmod{m}$.

*Kanıt.* Özellikleri sırasıyla kanıtlayalım:

1. $a - a = 0 = 0 \cdot m$ olduğundan $m \mid (a - a)$ sağlanır, yani $a \equiv a \pmod{m}$.
2. $a \equiv b \pmod{m} \implies m \mid (a - b) \implies a - b = k \cdot m$ $(k \in \mathbb{Z})$. Buradan $b - a = (-k) \cdot m$ olur. $-k \in \mathbb{Z}$ olduğundan $m \mid (b - a)$ elde edilir, yani $b \equiv a \pmod{m}$.

3. $a \equiv b \pmod{m}$ ve $b \equiv c \pmod{m}$ ise $a - b = k_1 m$ ve $b - c = k_2 m$ olacak şekilde $k_1, k_2 \in \mathbb{Z}$ vardır. Taraf tarafa toplarsak:

$$(a - b) + (b - c) = a - c = (k_1 + k_2)m$$

$k_1 + k_2 \in \mathbb{Z}$ olduğundan $m \mid (a - c)$ çıkar, yani $a \equiv c \pmod{m}$.

□

Bu üç özellik sayesinde tam sayılar kümesi $\mathbb{Z}$, $m$ modülüne göre $m$ tane ayrık kümeye parçalanır. Bu kümelere **kalan sınıfları** (denklik sınıfları) denir. Genellikle her sınıfı temsil etmek üzere $\{0, 1, 2, \ldots, m-1\}$ kümesindeki en küçük negatif olmayan sayılar seçilir.

### 14.1.3 Negatif Sayılar ve Programlamada Mod İşlemi

Matematiksel tanıma göre bir sayının mod $m$'deki dengi her zaman pozitif olmak zorunda değildir; örneğin:

$$-3 \equiv 4 \pmod{7}$$

çünkü $7 \mid (-3 - 4) \implies 7 \mid (-7)$ doğrudur. Ancak standart temsil istendiğinde, sonuca $m$'nin katları eklenerek $0 \leq r < m$ aralığına düşürülür:

$$-17 \pmod{5} \implies -17 + 4 \times 5 = 3 \implies -17 \equiv 3 \pmod{5}$$

**Örnek 14.3.** *$-44$ sayısının 9 modülündeki en küçük negatif olmayan denkliğini bulunuz.*

*Çözüm.* $-44$'e 9'un katlarını ekleyelim: $-44 + 9 \times 5 = -44 + 45 = 1$. $0 \leq 1 < 9$ olduğundan, $-44 \equiv 1 \pmod{9}$ elde edilir. □

#### Programlama Dillerinde Kalan İşlemi

Programlama yarışmalarında sık yapılan hatalardan biri, dillerdeki kalan operatörünün (`%`) matematiksel modüler aritmetikle birebir örtüşmemesidir.

C/C++ dillerinde `%` operatörü sıfıra doğru yuvarlama mantığıyla çalışır. Bu nedenle negatif sayılarda negatif sonuç üretir:

```
// C / C++ dillerinde:
int r = -17 % 5; // r degeri -2 cikar! (Cunku -17 = (-3)*5 + (-2))
```

Oysa matematikte $r \in \{0, 1, 2, 3, 4\}$ olmalıdır (yani $-17 \equiv 3 \pmod{5}$). Python'da ise `%` operatörü tabanlama mantığıyla tanımlandığından matematiksel kongruansla doğrudan uyumludur:

```
r = -17 MOD 5
```

C/C++ kullanırken negatif sayıların modunu güvenli bir biçimde almak için şu kalıp tercih edilir:

```
long long guvenli_mod(long long a, long long m) {
    long long r = a % m;
    if (r < 0) r += m;
    return r;
}
```

**Örnek 14.4.** *$x \equiv 5 \pmod{11}$ olduğuna göre, $2x + 7$ ifadesinin 11 modülündeki en küçük negatif olmayan temsilcisini bulunuz.*

*Çözüm.* $x \equiv 5 \pmod{11}$ ifadesi, $x = 11k + 5$ $(k \in \mathbb{Z})$ anlamına gelir. İfadede yerine yazarsak:

$$2x + 7 = 2(11k + 5) + 7 = 22k + 10 + 7 = 11(2k + 1) + 6$$

11'in tam katı olan terim atılırsa geriye kalan 6 olur. Dolayısıyla $2x + 7 \equiv 6 \pmod{11}$ bulunur. □

## 14.2 Modüler Aritmetik ve Üs

### 14.2.1 Aritmetik İşlemlerin Korunumu

Modüler aritmetiğin asıl kolaylığı, büyük sayılarla yapılan cebirsel işlemlerde her ara adımda mod alabilme esnekliğinden gelir. Bir sayıyı önce büyütüp sonra modunu almak ile, sayıların modunu alıp küçük sayılarla işlem yapmak tamamen aynı sonucu verir.

**Teorem 14.5** (İşlem Korunumu)**.** *$a \equiv a' \pmod{m}$ ve $b \equiv b' \pmod{m}$ olsun. Bu takdirde:*

1. $a + b \equiv a' + b' \pmod{m}$
2. $a - b \equiv a' - b' \pmod{m}$
3. $a \cdot b \equiv a' \cdot b' \pmod{m}$
4. *Her $k \in \mathbb{Z}^+$ için* $a^k \equiv (a')^k \pmod{m}$

*Kanıt.* Varsayımlar gereği $a - a' = k_1 m$ ve $b - b' = k_2 m$ olacak şekilde $k_1, k_2 \in \mathbb{Z}$ sayıları vardır.

1. $(a + b) - (a' + b') = (a - a') + (b - b') = k_1 m + k_2 m = (k_1 + k_2)m$. Buradan $m \mid ((a + b) - (a' + b'))$, yani $a + b \equiv a' + b' \pmod{m}$.
2. $(a - b) - (a' - b') = (a - a') - (b - b') = (k_1 - k_2)m$. Dolayısıyla $a - b \equiv a' - b' \pmod{m}$.

3. Çarpım için şu cebirsel özdeşliği kuralım:

$$ab - a'b' = ab - a'b + a'b - a'b' = b(a - a') + a'(b - b')$$

Burada $a - a' = k_1 m$ ve $b - b' = k_2 m$ yazarsak:

$$ab - a'b' = b(k_1 m) + a'(k_2 m) = m(bk_1 + a'k_2)$$

Parantez içi bir tam sayı olduğundan $m \mid (ab - a'b')$, yani $ab \equiv a'b' \pmod{m}$ elde edilir.

4. Üçüncü özelliğin $k$ defa art arda uygulanmasıyla (veya matematiksel tümevarımla) $a^k \equiv (a')^k \pmod{m}$ doğrudan kanıtlanır.

□

### 14.2.2 Bölme İşlemi Neden Yapılamaz? (Sadeleştirme Kuralı)

Toplama, çıkarma ve çarpmada geçerli olan bu esneklik, **bölme işleminde doğrudan uygulanamaz**.

**Örnek 14.6.** *Aşağıdaki kongruansı inceleyelim:*

$$6 \equiv 2 \pmod{4}$$

*Her iki tarafı* 2*'ye bölersek* $3 \equiv 1 \pmod{4}$ *elde ederiz. Fakat* $3 - 1 = 2$ *olup* $4 \nmid 2$*'dir; dolayısıyla denklik bozulur:* $3 \not\equiv 1 \pmod{4}$.

Bölme yaparken modülün de değişmesi gerekebilir:

**Teorem 14.7** (Sadeleştirme Kuralı)**.** *$c \neq 0$ bir tam sayı olsun.*

$$a \cdot c \equiv b \cdot c \pmod{m} \iff a \equiv b \pmod{\frac{m}{\gcd(c, m)}}$$

*Özel olarak, eğer* $\gcd(c, m) = 1$ *ise (yani $c$ ile $m$ aralarında asal ise), her iki taraf $c$'ye doğrudan bölünebilir:*

$$a \cdot c \equiv b \cdot c \pmod{m} \iff a \equiv b \pmod{m}$$

*Kanıt.* $ac \equiv bc \pmod{m} \implies m \mid (ac - bc) \implies m \mid c(a - b)$ demektir. Her iki tarafı $d = \gcd(c, m)$ değerine bölelim. Buradan:

$$\frac{m}{d} \;\Big|\; \frac{c}{d}(a - b)$$

elde edilir. EBOB tanımı gereği $\gcd\left(\frac{m}{d}, \frac{c}{d}\right) = 1$'dir. Bölüm 11'de gördüğümüz Öklid lemmasına göre, aralarında asal iki sayıdan biri diğerinin çarpanını bölmek zorundadır. Dolayısıyla:

$$\frac{m}{d} \;\Big|\; (a - b) \iff a \equiv b \pmod{\frac{m}{d}}$$

olup ispat tamamlanır. □

**Örnek 14.8.** $15x \equiv 35 \pmod{20}$ *kongruansını sağlayan tüm* $x$ *tam sayılarını bulunuz.*

*Çözüm.* Önce denklemi 5 ile sadeleştirelim. Burada $c = 5$ ve modül $m = 20$'dir. $\gcd(5, 20) = 5$ olduğundan yeni modülümüz $20/5 = 4$ olur:

$$3x \equiv 7 \pmod{4}$$

$7 \equiv 3 \pmod{4}$ olduğundan denklem $3x \equiv 3 \pmod{4}$ halini alır. Şimdi her iki tarafı 3'e bölelim. Bölüm 13'ten bildiğimiz üzere $\gcd(3, 4) = 1$ olduğundan modül değişmez:

$$x \equiv 1 \pmod{4}$$

Demek ki $x$, 4'e bölündüğünde 1 kalanını veren bir sayıdır. Ancak orijinal soruda modül 20 idi. $x = 4k + 1$ biçimindeki sayıların mod 20'deki karşılıklarını bulmak için $k \in \{0, 1, 2, 3, 4\}$ değerlerini veririz:

$$x \in \{1, 5, 9, 13, 17\} \pmod{20}$$

Orijinal modülde tam 5 farklı çözüm vardır; bu sayı doğrudan $\gcd(15, 20) = 5$ değerine eşittir. □

### 14.2.3 Hızlı Üs Alma (Binary Exponentiation)

Algoritmik problemlerde sıkça $a^b \pmod{m}$ değerini hesaplamamız istenir; burada $b$, $10^{18}$ gibi oldukça büyük bir sayı olabilir. $a$'yı $b$ defa kendisiyle çarpmak $O(b)$ adım sürer ve zaman sınırını aşar.

Bunun yerine ikili (binary) üs alma tekniği kullanılır. Buradaki temel fikir, $b$'nin paritesine (tek ya da çift oluşuna) göre üssü yarıya indirmektir:

$$a^b = \begin{cases} 1, & b = 0 \\ (a^{b/2})^2, & b \text{ çift ise} \\ a \cdot (a^{(b-1)/2})^2, & b \text{ tek ise} \end{cases}$$

Böylece her adımda üs yarıya iner ve hesaplama karmaşıklığı $O(\log b)$ seviyesine düşer.

```
// O(log b) zamaninda a^b mod m hesabi
long long hizli_us(long long a, long long b, long long m) {
    long long sonuc = 1;
    a %= m;
    while (b > 0) {
        if (b & 1) { // b tek sayi ise
            sonuc = (sonuc * a) % m;
        }
        a = (a * a) % m;
        b >>= 1; // b'yi 2'ye bol
```

```
    }
    return sonuc;
}
```

**Örnek 14.9.** $3^{25} \pmod{13}$ *değerini hızlı üs alma mantığıyla elle hesaplayınız.*

*Çözüm.* 25'i ikili tabanda yazalım: $25 = 16+8+1 = (11001)_2$. 3'ün 2'nin kuvveti olan üslerini mod 13'te adım adım karesini alarak belirleyelim:

$$\begin{aligned} 3^1 &\equiv 3 \pmod{13} \\ 3^2 &\equiv 9 \equiv -4 \pmod{13} \\ 3^4 &\equiv (-4)^2 = 16 \equiv 3 \pmod{13} \\ 3^8 &\equiv 3^2 \equiv 9 \equiv -4 \pmod{13} \\ 3^{16} &\equiv (-4)^2 = 16 \equiv 3 \pmod{13} \end{aligned}$$

$25 = 16 + 8 + 1$ olduğundan bu bileşenleri çarparız:

$$3^{25} = 3^{16} \cdot 3^8 \cdot 3^1 \equiv 3 \cdot (-4) \cdot 3 \pmod{13}$$

$$3 \cdot (-4) \cdot 3 = -36$$

$-36$'yı 13 modülüne indirgersek: $-36 + 3 \times 13 = -36 + 39 = 3$. Dolayısıyla $3^{25} \equiv 3 \pmod{13}$ bulunur. □

## 14.3 Son Basamak ve Kalan Problemleri

### 14.3.1 Motivasyon ve Sayı Tabanlarıyla İlişki

10 tabanında yazılmış bir $N$ tam sayısını ele alalım:

$$N = a_k 10^k + a_{k-1} 10^{k-1} + \cdots + a_1 10 + a_0$$

Burada 10'un katı olan terimlerin tümü mod 10'da sıfırlanır:

$$N \equiv a_0 \pmod{10}$$

Buna göre bir sayının:

- Son bir basamağı (birler basamağı) $\iff N \pmod{10}$,
- Son iki basamağı $\iff N \pmod{100}$,
- Son $k$ basamağı $\iff N \pmod{10^k}$

değerine karşılık gelir. Dolayısıyla son basamak soruları, doğrudan mod $10^k$ aritmetiğidir.

### 14.3.2 Periyodiklik (Döngü İlkesi)

$a \pmod m$ dizisinin kuvvetlerini sıralayalım:

$$a^1, a^2, a^3, a^4, \ldots \pmod m$$

Mod $m$'de yalnızca $m$ farklı kalan sınıfı $\{0, 1, \ldots, m-1\}$ bulunabilir. Bölüm 7'de incelediğimiz güvercin yuvası ilkesi gereği, ilk $m+1$ terim içinde en az iki terim birbirine denk olmak zorundadır. Bir kalan tekrar ettiği anda, sonraki tüm adımlar aynı döngüyü izler.

**Örnek 14.10.** $7^{2026}$ *sayısının birler basamağını bulunuz.*

*Çözüm.* Birler basamağını bulmak, ifadeyi mod 10'da incelemek demektir. 7'nin ardışık kuvvetlerine bakarak periyodu belirleyelim:

$$\begin{aligned}
7^1 &\equiv 7 \pmod{10} \\
7^2 &\equiv 49 \equiv 9 \pmod{10} \\
7^3 &\equiv 9 \times 7 = 63 \equiv 3 \pmod{10} \\
7^4 &\equiv 3 \times 7 = 21 \equiv 1 \pmod{10} \\
7^5 &\equiv 1 \times 7 = 7 \pmod{10} \quad \text{(Döngü başa döndü)}
\end{aligned}$$

Kalanlar $7, 9, 3, 1$ biçiminde 4 adımda bir tekrarlanır. Bu nedenle üssün 4 ile bölümünden kalanına bakarız:

$$2026 = 4 \times 506 + 2 \implies 2026 \equiv 2 \pmod 4$$

O halde:

$$7^{2026} \equiv 7^2 \equiv 9 \pmod{10}$$

Sayının birler basamağı 9'dur. □

### 14.3.3 Fermat'nın Küçük Teoremi

Bir önceki alt bölümde kuvvetlerin periyodunu incelemiştik. Modül asal olduğunda bu periyot için evrensel bir üst sınır vardır: periyot her zaman $p-1$ sayısını böler. Bu gözlem, büyük üsleri küçültmenin en pratik yolunu verir.

**Teorem 14.11** (Fermat'nın Küçük Teoremi)**.** *$p$ bir asal sayı ve $a$ bir tam sayı olsun. Eğer $p \nmid a$ ise,*

$$a^{p-1} \equiv 1 \pmod p$$

*olur.*

*Kanıt.* $a, 2a, 3a, \ldots, (p-1)a$ sayılarını mod $p$'de inceleyelim.

(i) Hiçbiri 0'a denk değildir: $p$ asal, $p \nmid a$ ve $1 \leq k \leq p-1$ iken $p \mid ka$ olamaz.

(ii) İkişer ikişer farklıdırlar: $ia \equiv ja \pmod{p}$ olsaydı $p \mid (i-j)a$ olurdu; $p$ asal ve $p \nmid a$ olduğundan $p \mid (i-j)$ çıkardı, ancak $|i-j| \leq p-1$ olduğundan bu ancak $i = j$ ile mümkündür.

(iii) Böylece bu $p-1$ sayının mod $p$'deki kalanları $\{1, 2, \ldots, p-1\}$ kümesinin bir permütasyonudur. İki tarafın çarpımını eşitleyelim:

$$a \cdot 2a \cdots (p-1)a \equiv 1 \cdot 2 \cdots (p-1) \pmod{p}$$

yani

$$a^{p-1}(p-1)! \equiv (p-1)! \pmod{p}$$

elde edilir.

(iv) $\gcd((p-1)!, p) = 1$ olduğundan (Sadeleştirme Kuralı) iki tarafı $(p-1)!$ ile sadeleştirebiliriz:

$$a^{p-1} \equiv 1 \pmod{p}.$$

□

**Sonuç 14.12.** *Her $a$ tam sayısı ve her $p$ asal sayısı için*

$$a^p \equiv a \pmod{p}$$

*sağlanır. Gerçekten de $p \mid a$ ise iki taraf da 0'a denktir; aksi halde Fermat'nın Küçük Teoremi'ni $a$ ile çarpmak yeterlidir.*

**Örnek 14.13.** ***a)*** $3^{100}$ *sayısının 7'ye bölümünden kalanı bulunuz.*

***b)*** $2^{1000}$ *sayısının 13'e bölümünden kalanı bulunuz.*

*Çözüm.* **a)** 7 asal ve 3 ile aralarında asal olduğundan $3^6 \equiv 1 \pmod{7}$'dir. $100 = 6 \cdot 16 + 4$ olduğundan

$$3^{100} \equiv 3^4 = 81 \equiv 4 \pmod{7}$$

bulunur.

**b)** 13 asal ve $\gcd(2, 13) = 1$ olduğundan $2^{12} \equiv 1 \pmod{13}$'tür. $1000 = 12 \cdot 83 + 4$ olduğundan

$$2^{1000} \equiv 2^4 = 16 \equiv 3 \pmod{13}$$

elde edilir. □

**Uyarı 1.** *Fermat'nın Küçük Teoremi bu biçimiyle yalnızca modül asal olduğunda kullanılabilir; bileşik modüllerde periyot yöntemine dönülür. Ayrıca teoremin uygulanabilmesi için $p \nmid a$ şartı gereklidir.*

### 14.3.4 Kritik Bir Tuzak: Üste Mod Uygulanmaz

**Teorem 14.14** (Önemli Uyarı)**.** *Genel olarak* $a^b \pmod m \neq a^{(b \pmod m)} \pmod m$*. Üs, tabanın tabi olduğu mod ile doğrudan sadeleştirilemez.*

Örneğin $2^5 \pmod 3$ ifadesini ele alalım:

- Doğru hesap: $2^5 = 32$ ve $32 = 3 \times 10 + 2 \implies 2^5 \equiv 2 \pmod 3$.
- Hatalı işlem: Üssün modunu alıp $5 \equiv 2 \pmod 3$ diyerek $2^2 \equiv 4 \equiv 1 \pmod 3$ yazmak yanlıştır.

Görüldüğü gibi $2 \neq 1$'dir. Üslerin periyodu modülün kendisiyle değil, kuvvetlerin tekrar etme düzeniyle ilgilidir.

**Örnek 14.15.** $2^{103}$ *sayısının* 13 *ile bölümünden kalanı bulunuz.*

*Çözüm.* İlk hamle olarak mod 13'te 1 ya da $-1$ değerini veren bir kuvvet arayalım:

$$\begin{aligned} 2^1 &= 2 \\ 2^2 &= 4 \\ 2^3 &= 8 \\ 2^4 &= 16 \equiv 3 \pmod{13} \\ 2^5 &= 3 \times 2 = 6 \\ 2^6 &= 6 \times 2 = 12 \equiv -1 \pmod{13} \end{aligned}$$

$2^6 \equiv -1 \pmod{13}$ denkliği hesabı oldukça kolaylaştırır. 103'ü 6'ya bölelim:

$$103 = 6 \times 17 + 1$$

Üslü ifadeyi bu çarpanlara göre ayıralım:

$$2^{103} = (2^6)^{17} \cdot 2^1 \equiv (-1)^{17} \cdot 2 \pmod{13}$$

$(-1)^{17} = -1$ olduğundan:

$$2^{103} \equiv -1 \cdot 2 = -2 \pmod{13}$$

En küçük pozitif temsilci için 13 ekleriz:

$$-2 + 13 = 11$$

Kalan 11'dir. □

**Örnek 14.16.** $3^{2025}$ *sayısının son iki basamağını bulunuz.*

*Çözüm.* Son iki basamağı bulmak için sayıyı mod 100'de incelememiz gerekir. 3'ün kuvvetlerini yazarak başlayalım:

$$\begin{aligned} 3^1 &= 3 \\ 3^2 &= 9 \\ 3^3 &= 27 \\ 3^4 &= 81 \equiv -19 \pmod{100} \\ 3^5 &= 243 \equiv 43 \pmod{100} \end{aligned}$$

Bu kuvvetler yerine $3^4 = 81$ değerini $80 + 1$ olarak yazıp açmak daha pratik bir yoldur:

$$3^{2025} = 3 \cdot 3^{2024} = 3 \cdot (3^4)^{506} = 3 \cdot 81^{506} = 3 \cdot (1 + 80)^{506}$$

Bölüm 6'da gördüğümüz binom açılımını mod 100'de uygulayalım:

$$(1 + 80)^{506} = 1 + \binom{506}{1} \cdot 80 + \binom{506}{2} \cdot 80^2 + \dots$$

Burada $80^2 = 6400$ sayısı 100'ün tam katıdır; dolayısıyla ikinci ve sonraki tüm terimler mod 100'de sıfır olur:

$$(1 + 80)^{506} \equiv 1 + 506 \times 80 \pmod{100}$$

Şimdi $506 \times 80 \pmod{100}$ çarpımına bakalım: $506 \equiv 6 \pmod{10}$ olduğundan, $506 \times 80 = (500 + 6) \times 80 = 40000 + 480 \equiv 80 \pmod{100}$. Buradan:

$$(1 + 80)^{506} \equiv 1 + 80 = 81 \pmod{100}$$

elde edilir. Bunu baştaki 3 katsayısıyla çarptığımızda:

$$3^{2025} = 3 \cdot 81^{506} \equiv 3 \cdot 81 = 243 \equiv 43 \pmod{100}$$

bulunur. Sayının son iki basamağı 43'tür. □

**Örnek 14.17.** $S = 1! + 2! + 3! + 4! + \cdots + 2024!$ *toplamının* 15 *ile bölümünden kalanı bulunuz.*

*Çözüm.* Faktöriyel terimleri büyüdükçe bölenin asal çarpanlarını içermeye başlar. $15 = 3 \times 5$ olduğundan, çarpanlarında hem 3 hem de 5 bulunan ilk terim 5!'dir ($5! = 120 = 15 \times 8$). Dolayısıyla her $n \geq 5$ için $15 \mid n!$, yani:

$$n! \equiv 0 \pmod{15} \quad (\forall n \geq 5)$$

Bu durum toplamı önemli ölçüde sadeleştirir; 5!'den sonraki tüm terimler sıfıra denktir:

$$S \equiv 1! + 2! + 3! + 4! \pmod{15}$$

İlk dört terimi doğrudan hesaplayalım:

$$1! = 1$$
$$2! = 2$$
$$3! = 6$$
$$4! = 24 \equiv 9 \pmod{15}$$

Değerleri toplarsak:

$$S \equiv 1 + 2 + 6 + 9 = 18 \pmod{15}$$

$18 \equiv 3 \pmod{15}$ olduğundan kalan 3 çıkar. □

**Örnek 14.18** (Kule Üs Problemi)**.** $2^{3^4}$ *sayısının* 7 *ile bölümünden kalanı bulunuz.*

*Çözüm.* Kule üslerde işlem sırası yukarıdan aşağıyadır: $2^{3^4} = 2^{(3^4)} = 2^{81}$. Taban 2, modül ise 7'dir. 2'nin 7 modundaki kuvvetlerini inceleyelim:

$$2^1 \equiv 2 \pmod{7}$$
$$2^2 \equiv 4 \pmod{7}$$
$$2^3 = 8 \equiv 1 \pmod{7}$$

Döngü uzunluğu 3'tür. Yani 2'nin üssü 3'ün katı olduğunda mod 7'deki değer 1 olur. Bu nedenle $2^k \pmod{7}$ ifadesini belirleyen, $k$ üssünün 3 modundaki değeridir:

$$k = 3^4 = 81$$

Üssü incelersek:

$$81 = 3 \times 27 \implies 81 \equiv 0 \pmod{3}$$

Üs 3'ün tam katı olduğuna göre:

$$2^{81} = (2^3)^{27} \equiv 1^{27} = 1 \pmod{7}$$

Kalan 1 bulunur. □

**Örnek 14.19.** *$n$ bir pozitif tam sayı olmak üzere, $n^5 - n$ ifadesinin her zaman* 30 *ile bölündüğünü modüler aritmetik kullanarak kanıtlayınız.*

*Çözüm.* $30 = 2 \times 3 \times 5$ biçiminde asal çarpanlarına ayrılır. Çarpanlar aralarında asal olduğundan, ifadenin 30 ile bölünmesi için 2, 3 ve 5 ile ayrı ayrı bölünmesi yeterlidir. İfadeyi çarpanlarına ayıralım:

$$n^5 - n = n(n^4 - 1) = n(n^2 - 1)(n^2 + 1) = n(n - 1)(n + 1)(n^2 + 1)$$

- **Mod 2 incelemesi:** $(n - 1)n$ ardışık iki tam sayıdır, dolayısıyla biri çifttir. Böylece $2 \mid (n^5 - n)$ sağlanır.

- **Mod 3 incelemesi:** $(n-1)n(n+1)$ ardışık üç tam sayıdır. Bölüm 11'den bildiğimiz üzere ardışık üç sayıdan en az biri 3'ün katıdır. Buradan $3 \mid (n^5 - n)$ elde edilir.
- **Mod 5 incelemesi:** $n \pmod 5$ için kalanlar $\{0, 1, 2, 3, 4\}$ kümesindedir:
  - $n \equiv 0 \pmod 5 \implies n$ çarpanından ötürü ifade $\equiv 0$.
  - $n \equiv 1 \pmod 5 \implies (n-1)$ çarpanından ötürü $\equiv 0$.
  - $n \equiv 4 \equiv -1 \pmod 5 \implies (n+1)$ çarpanından ötürü $\equiv 0$.
  - $n \equiv 2 \pmod 5 \implies n^2 + 1 \equiv 2^2 + 1 = 5 \equiv 0 \pmod 5$.
  - $n \equiv 3 \equiv -2 \pmod 5 \implies n^2 + 1 \equiv (-2)^2 + 1 = 5 \equiv 0 \pmod 5$.

  Her durumda ifade 5'e tam bölünür.

İfade 2, 3 ve 5 ile tam bölündüğünden, $\operatorname{lcm}(2, 3, 5) = 30$ ile de tam bölünür. □

## Bölüm Özeti

- **Kongruans:** $a \equiv b \pmod m \iff m \mid (a - b)$. Tam sayılar $m$ modülüne göre $m$ tane kalan sınıfına ayrılır.
- **Aritmetik Kurallar:** Toplama, çıkarma, çarpma ve pozitif tam sayı üsleri mod altında korunur:

$$(a \pm b) \pmod m = ((a \pmod m) \pm (b \pmod m)) \pmod m$$

$$(a \cdot b) \pmod m = ((a \pmod m) \cdot (b \pmod m)) \pmod m$$

- **Sadeleştirme:** $ac \equiv bc \pmod m$ denkliğinde $c$ sadeleştirilirse modül $m/\gcd(c, m)$ olur. Eğer $\gcd(c, m) = 1$ ise modül değişmez.
- **Hızlı Üs Alma:** $a^b \pmod m$ hesabı, $b$'nin ikili açılımı kullanılarak $O(\log b)$ adımda yapılabilir.
- **Döngüsellik:** $a^k \pmod m$ dizisi Bölüm 7'deki güvercin yuvası ilkesi gereği periyodiktir. Son basamaklar mod $10^k$ ile belirlenir.
- **Fermat'nın Küçük Teoremi:** $p$ asal ve $p \nmid a$ ise $a^{p-1} \equiv 1 \pmod p$; eşdeğer olarak $a^p \equiv a \pmod p$. Böylece asal modülde büyük üsler $p - 1$'e göre indirgenebilir.
- **Tuzak:** Üssün modu doğrudan alınamaz: $a^b \pmod m \neq a^{b \pmod m} \pmod m$. Üs, kuvvetlerin döngü periyoduna göre indirgenmelidir.

# Bölüm 15

# Taban Aritmetiği

Günlük hayatta sayıları neredeyse istisnasız onluk sistemde yazar ve düşünürüz: 354 sayısı bizim için üç yüz elli dörttür. Ancak doğanın veya matematiğin kendisinde 10 sayısına tanınmış nesnel bir ayrıcalık yoktur. İnsanlığın onluk sistemi benimsemesindeki temel etken, iki elimizde toplam on parmağımızın bulunmasıdır. Şayet sekiz parmağımız olsaydı, bugün muhtemelen sekizlik sistemi kullanıyor olacaktık.

Bilgisayar biliminde ise fiziksel gerçeklik farklıdır: Sayısal devreler transistörlerden oluşur ve bir transistör temel olarak iki kararlı duruma sahiptir: "elektrik akımı var" (yüksek voltaj, 1) veya "elektrik akımı yok" (düşük voltaj, 0). Bu nedenle modern bilgisayarların ana dili ikili (binary) tabandır. Dahası, uzun ikili dizileri daha okunabilir kılmak adına sekizli (octal) ve onaltılı (hexadecimal) tabanlar tercih edilir.

Bu bölümde pozisyonel sayı sistemlerinin matematiksel temellerini, tabanlar arası dönüşüm yöntemlerini, farklı tabanlarda aritmetik işlemleri ve ikili sistemin algoritmik problem çözmedeki yerini inceleyeceğiz.

## 15.1 Taban ve Dönüşüm

### 15.1.1 Pozisyonel Sayı Sistemi ve Sezgi

Bir sayıyı yazarken her bir rakamın değeri, bulunduğu basamağa (pozisyona) bağlıdır. Örneğin 10 tabanında yazılmış 354 sayısını basamak çözümlemesiyle ele alalım:

$$354 = 3 \cdot 100 + 5 \cdot 10 + 4 \cdot 1 = 3 \cdot 10^2 + 5 \cdot 10^1 + 4 \cdot 10^0$$

Buradaki temel ilke şudur: Her basamak, tabanın (10'un) artan bir kuvvetini temsil eder ve kullanılan rakamlar daima $\{0, 1, 2, \ldots, 9\}$ kümesindendir; yani tabandan kesinlikle küçüktür.

Tabanımız 10 yerine 2 olsaydı basamak değerleri 2'nin kuvvetleri ($2^0 = 1, 2^1 =$

$2, 2^2 = 4, 2^3 = 8, \dots$) olur ve kullanabileceğimiz rakamlar yalnızca $\{0, 1\}$ kümesinden seçilirdi. Örneğin $(101)_2$ sayısını açarsak:

$$(101)_2 = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 4 + 0 + 1 = 5$$

elde ederiz. Benzer biçimde taban 7 seçilseydi basamak ağırlıkları $7^0, 7^1, 7^2, \dots$ olacak ve izin verilen rakamlar $\{0, 1, 2, 3, 4, 5, 6\}$ kümesini oluşturacaktı.

Bu sezgiyi matematiksel bir tanıma dönüştürelim.

**Tanım 15.1** (Sayı Tabanı ve Pozisyonel Gösterim)**.** *$b \geq 2$ bir tamsayı olsun. Bir $N$ doğal sayısının $b$ tabanındaki (radix-b) gösterimi, $a_k \neq 0$ ve her $0 \leq i \leq k$ için $a_i \in \{0, 1, \dots, b-1\}$ olmak üzere,*

$$(a_k a_{k-1} \dots a_1 a_0)_b$$

*biçiminde ifade edilen dizidir. Bu gösterimin sayısal değeri*

$$N = \sum_{i=0}^{k} a_i b^i = a_k b^k + a_{k-1} b^{k-1} + \cdots + a_1 b + a_0$$

*olarak tanımlanır. Taban belirtilmemişse varsayılan taban 10'dur.*

Taban 10'dan büyük olduğunda ($b > 10$), 9'dan büyük rakamları tek bir sembolle göstermek için harfler kullanılır: $A = 10, B = 11, C = 12, D = 13, E = 14, F = 15, \dots$ gibi. Örneğin onaltılı tabanda $(2F)_{16} = 2 \cdot 16^1 + 15 \cdot 16^0 = 32 + 15 = 47$ olur.

Her pozitif tamsayının, verilen herhangi bir $b \geq 2$ tabanında tek bir şekilde yazılabileceğini aşağıdaki teorem garanti eder.

**Teorem 15.2** (Taban Gösteriminin Varlığı ve Tekliği)**.** *$b \geq 2$ sabit bir tamsayı olsun. Her pozitif $N$ tamsayısı için,*

$$N = a_k b^k + a_{k-1} b^{k-1} + \cdots + a_1 b + a_0$$

*eşitliğini sağlayan, $a_k \neq 0$ ve her $i \in \{0, 1, \dots, k\}$ için $0 \leq a_i < b$ koşuluna uyan bir $k \geq 0$ tamsayısı ve katsayılar dizisi $(a_k, a_{k-1}, \dots, a_0)$ tek bir biçimde mevcuttur.*

*Kanıt.* İspatı iki aşamada kuralım: önce varlık (algoritmik inşa), ardından teklik.

**1. Varlık:** Bölüm 11'de gördüğümüz Bölme Algoritmasını ($N = q \cdot b + r$, $0 \leq r < b$) ardışık olarak uygulayalım:

$$\begin{aligned}
N &= q_0 b + a_0, \quad 0 \leq a_0 < b \\
q_0 &= q_1 b + a_1, \quad 0 \leq a_1 < b \\
q_1 &= q_2 b + a_2, \quad 0 \leq a_2 < b \\
&\vdots \\
q_{k-1} &= q_k b + a_k, \quad 0 \leq a_k < b \quad (q_k = 0 \text{ olduğunda durulur})
\end{aligned}$$

$b \geq 2$ olduğundan, $N > q_0 > q_1 > q_2 > \cdots \geq 0$ kesin azalan bir negatif olmayan tamsayılar dizisidir. İyisıralama ilkesi gereği bu dizi sonsuza kadar devam edemez; sonlu bir adımda bir $q_k = 0$ değerine ulaşmak zorundadır ($q_{k-1} \neq 0$ olduğundan $a_k \neq 0$'dır).

Elde edilen kalanları geriye doğru yerine yazarsak:

$$N = q_0 b + a_0 = (q_1 b + a_1)b + a_0 = q_1 b^2 + a_1 b + a_0$$
$$= \ldots = a_k b^k + a_{k-1} b^{k-1} + \cdots + a_1 b + a_0$$

ifadesine ulaşırız. Bu, aranan gösterimin var olduğunu gösterir.

**2. Teklik:** Gösterimin tek olmadığını varsayalım. Bu durumda $N$ sayısının $b$ tabanında iki farklı gösterimi bulunsun. Gerekirse katsayıların başına sıfır ekleyerek basamak sayılarını eşitleyebiliriz:

$$N = \sum_{i=0}^{m} a_i b^i = \sum_{i=0}^{m} c_i b^i, \quad 0 \leq a_i, c_i < b$$

Buradan:

$$\sum_{i=0}^{m} (a_i - c_i) b^i = 0$$

yazılır. Gösterimler farklı olduğundan, $a_i \neq c_i$ şartını sağlayan en az bir indeks vardır. Bu indekslerin en küçüğüne $j$ diyelim ($0 \leq j \leq m$). O halde her $i < j$ için $a_i = c_i$'dir. Eşitlik şu biçimi alır:

$$(a_j - c_j)b^j + \sum_{i=j+1}^{m} (a_i - c_i)b^i = 0 \implies (c_j - a_j)b^j = \sum_{i=j+1}^{m} (a_i - c_i)b^i$$

Her iki tarafı $b^{j+1}$ modülünde incelersek, sağ taraf $b^{j+1}$'in tam katı olduğundan:

$$(c_j - a_j)b^j \equiv 0 \pmod{b^{j+1}} \implies c_j - a_j \equiv 0 \pmod{b}$$

elde edilir. Fakat $0 \leq a_j, c_j < b$ olduğundan aralarındaki fark $-b < c_j - a_j < b$ aralığındadır. Bu aralıkta $b$'ye bölünebilen tek tamsayı 0'dır. Demek ki $c_j - a_j = 0$, yani $a_j = c_j$ olmalıdır. Bu ise $j$'nin $a_j \neq c_j$ şartını sağlayan en küçük indeks olmasıyla çelişir. Dolayısıyla gösterim tek olmak zorundadır. □

### 15.1.2 Tabanlar Arası Dönüşüm Yöntemleri

#### Keyfi bir tabandan 10 tabanına geçiş

$(a_k a_{k-1} \ldots a_1 a_0)_b$ sayısını 10 tabanına dönüştürmek için tanım gereği polinom açılımı hesaplanır:

$$N = a_k b^k + a_{k-1} b^{k-1} + \cdots + a_1 b + a_0$$

Bu toplamı doğrudan kuvvet alarak hesaplamak yerine, algoritmalarda sıkça başvurulan **Horner Yöntemi** ile paranteze alarak iç içe hesaplamak hem elle işlem yaparken hem de kod yazarken işlem sayısını ve hata payını azaltır:

$$N = \Big( \ldots \big( (a_k \cdot b + a_{k-1}) \cdot b + a_{k-2} \big) \cdot b + \cdots + a_1 \Big) \cdot b + a_0$$

**Örnek 15.3.** $(110101)_2$ *sayısını* 10 *tabanına çeviriniz.*

*Çözüm.* **1. Yol (Kuvvetler toplamı):** Basamak değerlerini sağdan sola doğru $2^0, 2^1, 2^2, 2^3, 2^4, 2^5$ olarak yazalım:

$$(110101)_2 = 1 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$$

$$= 32 + 16 + 0 + 4 + 0 + 1 = 53$$

**2. Yol (Horner yöntemi):** En soldaki bitten başlayıp her adımda sonucu 2 ile çarpıp bir sonraki biti ekleyelim:

Başlangıç: 1
1. adım: $1 \cdot 2 + 1 = 3$
2. adım: $3 \cdot 2 + 0 = 6$
3. adım: $6 \cdot 2 + 1 = 13$
4. adım: $13 \cdot 2 + 0 = 26$
5. adım: $26 \cdot 2 + 1 = 53$

Her iki yöntem de beklediğimiz gibi 53 sonucunu verir. □

### 10 tabanından keyfi bir tabana geçiş: Ardışık Bölme

Varlık ispatındaki kurgu, doğrudan pratik dönüşüm yöntemini verir: Sayıyı $b$'ye böleriz; kalan, gösterimin en sağdaki basamağını ($a_0$) oluşturur. Elde edilen bölümü tekrar $b$'ye böldüğümüzde yeni kalan $a_1$ olur. Bu işleme bölüm 0 olana kadar devam edilir. Kalanlar sondan başa (aşağıdan yukarıya) doğru dizilerek sayı elde edilir.

**Örnek 15.4.** 158 *sayısını* 2 *ve* 7 *tabanlarında yazınız.*

*Çözüm.* Önce 7 tabanına dönüştürelim:

$$158 \div 7 = 22 \quad (\text{kalan } a_0 = 4)$$
$$22 \div 7 = 3 \quad (\text{kalan } a_1 = 1)$$
$$3 \div 7 = 0 \quad (\text{kalan } a_2 = 3)$$

Kalanları sondan başa doğru okursak: $158 = (314)_7$. Sağlamasını yapalım: $3 \cdot 7^2 + 1 \cdot 7^1 + 4 \cdot 7^0 = 3 \cdot 49 + 7 + 4 = 147 + 11 = 158$.

Şimdi aynı sayıyı 2 tabanına dönüştürelim:

$$\begin{aligned} 158 \div 2 &= 79 \quad \text{(kalan 0)} \\ 79 \div 2 &= 39 \quad \text{(kalan 1)} \\ 39 \div 2 &= 19 \quad \text{(kalan 1)} \\ 19 \div 2 &= 9 \quad \text{(kalan 1)} \\ 9 \div 2 &= 4 \quad \text{(kalan 1)} \\ 4 \div 2 &= 2 \quad \text{(kalan 0)} \\ 2 \div 2 &= 1 \quad \text{(kalan 0)} \\ 1 \div 2 &= 0 \quad \text{(kalan 1)} \end{aligned}$$

Kalanları aşağıdan yukarıya doğru yazarsak: $158 = (10011110)_2$. □

### $2^k$ Tabanları Arasındaki Doğrudan Geçiş (Gruplama İlkesi)

Bilgisayar biliminde ikili ($2^1$), sekizli ($2^3$) ve onaltılı ($2^4$) sistemlerin yaygın olmasının temel bir nedeni vardır: Bu tabanlar arasında 10 tabanına uğramadan, basamakları bloklayarak doğrudan geçiş yapılabilir.

$b = 2^k$ olsun. İkili tabandaki bir sayının her $k$ basamaklık bloğu, doğrudan $2^k$ tabanındaki tek bir basamağa karşılık gelir. Çünkü:

$$\sum_{i=0}^{k-1} a_i 2^i < \sum_{i=0}^{k-1} 1 \cdot 2^i = 2^k - 1$$

olup, bu değer tam olarak $2^k$ tabanındaki bir rakamın alabileceği değer aralığındadır (0 ile $2^k - 1$ arası).

- **İkili ↔ Sekizli ($2^3$):** İkili sayı sağdan sola doğru üçerli gruplara ayrılır (en soldaki eksik basamaklar sıfırla tamamlanır). Her 3 bitlik grup bir sekizli rakama çevrilir.
- **İkili ↔ Onaltılı ($2^4$):** İkili sayı sağdan sola doğru dörderli gruplara ayrılır. Her 4 bitlik grup (nibble) onaltılı bir rakama (0–9, $A$–$F$) çevrilir.

**Örnek 15.5.** $(11010111101)_2$ *sayısını sekizli ve onaltılı tabana çeviriniz.*

*Çözüm.* **Sekizli dönüşüm (üçerli gruplama):** Sağdan sola üçer üçer ayıralım:

$$(011 \mid 010 \mid 111 \mid 101)_2$$

Her bir grubu sekizli rakama dönüştürelim:

$$\begin{aligned} (011)_2 &= 3 \\ (010)_2 &= 2 \\ (111)_2 &= 7 \\ (101)_2 &= 5 \end{aligned}$$

Buradan sayı $(3275)_8$ bulunur.

**Onaltılı dönüşüm (dörderli gruplama):** Sağdan sola dörder dörder ayıralım:

$$(0110 \mid 1011 \mid 1101)_2$$

Her bir grubu onaltılı rakama dönüştürelim:

$$\begin{aligned}(0110)_2 &= 6\\ (1011)_2 &= 11 \implies B\\ (1101)_2 &= 13 \implies D\end{aligned}$$

Böylece $(6BD)_{16}$ elde edilir. □

### 15.1.3 Basamak Sayısı ve Tabanın Büyüklüğü

Bir $N$ pozitif tamsayısının 10 tabanındaki basamak sayısını $\lfloor \log_{10} N \rfloor + 1$ formülüyle hesaplarız. Bu kural herhangi bir $b \geq 2$ tabanı için de geçerlidir.

**Teorem 15.6** (Basamak Sayısı)**.** *$N$ pozitif bir tamsayı ve $b \geq 2$ olsun. $N$'nin $b$ tabanındaki basamak sayısı*

$$L_b(N) = \lfloor \log_b N \rfloor + 1$$

*ile verilir.*

*Kanıt.* $N$ sayısı $b$ tabanında $k+1$ basamaklı olsun. Bu durumda en yüksek basamağın katsayısı $a_k \geq 1$ olmak üzere:

$$b^k \leq a_k b^k \leq N = \sum_{i=0}^{k} a_i b^i \leq \sum_{i=0}^{k} (b-1) b^i = (b-1)\frac{b^{k+1}-1}{b-1} = b^{k+1} - 1 < b^{k+1}$$

Demek ki,

$$b^k \leq N < b^{k+1}$$

Eşitsizliğin $b$ tabanında logaritmasını alırsak ($f(x) = \log_b x$ kesin artan fonksiyondur):

$$k \leq \log_b N < k+1$$

Tam değer (taban) tanımı gereği:

$$k = \lfloor \log_b N \rfloor$$

Basamak sayısı $k+1$ olduğundan, toplam basamak sayısı $\lfloor \log_b N \rfloor + 1$ olarak bulunur. □

Bu bağıntı, bir sayının küçük tabanlarda (örneğin $b = 2$) daha çok, büyük tabanlarda (örneğin $b = 16$) daha az basamakla temsil edildiğini gösterir. $2^{10} = 1024 \approx 10^3$ yaklaşımından hareketle, bir sayının ikili tabandaki basamak sayısı onluk tabandaki basamak sayısının yaklaşık $\log_2 10 \approx 3.32$ katıdır.

### 15.1.4 Çözümlü Örnekler

Olimpiyat sınavlarında taban aritmetiği genellikle cebirsel denklemler, basamak çözümlemeleri veya bölünebilme kurallarıyla bir arada sorulur.

**Örnek 15.7.** $(ab)_8 = (ba)_9$ *eşitliğini sağlayan sıfırdan farklı* $a$ *ve* $b$ *rakamlarını bulunuz.*

*Çözüm.* İki farklı tabandaki bu ifadeyi ortak bir zemine çekmek için 10 tabanındaki polinom açılımlarını kullanalım. Ayrıca taban tanımlarından gelen basamak sınırlarını ($a, b < 8$ ve $a, b < 9$) unutmamalıyız.

$(ab)_8$ sayısının 8 tabanında iki basamaklı bir sayı belirtmesi için $1 \leq a \leq 7$ ve $0 \leq b \leq 7$ olmalıdır. Benzer şekilde $(ba)_9$ için $1 \leq b \leq 8$ ve $0 \leq a \leq 8$ gereklidir. Bu iki koşulun kesişimi:

$$1 \leq a \leq 7 \quad \text{ve} \quad 1 \leq b \leq 7$$

Açılımları yazarsak:

$$(ab)_8 = 8a + b$$

$$(ba)_9 = 9b + a$$

Eşitliği düzenleyelim:

$$8a + b = 9b + a \implies 7a = 8b$$

$\gcd(7, 8) = 1$ olduğundan (Bölüm 13), $7 \mid 8b \implies 7 \mid b$ olmalıdır. Fakat $1 \leq b \leq 7$ aralığında 7'nin tek katı $b = 7$'dir. $b = 7$ yazılırsa $7a = 8 \cdot 7 \implies a = 8$ bulunur.

Ancak $a = 8$ değeri $1 \leq a \leq 7$ aralığına düşmez; 8 tabanında 8 rakamı bulunamaz. Dolayısıyla bu eşitliği sağlayan hiçbir $a, b$ rakam çifti **yoktur**. □

**Örnek 15.8.** *Hangi* $b > 4$ *tabanında* $(1331)_b$ *sayısı bir tamküptür?*

*Çözüm.* 1331 sayısındaki $1, 3, 3, 1$ katsayıları, Bölüm 6'dan hatırlayacağımız binom katsayıları $\binom{3}{k}$ ile örtüşmektedir.

Sayının $b$ tabanındaki açılımını yazalım:

$$(1331)_b = 1 \cdot b^3 + 3 \cdot b^2 + 3 \cdot b + 1$$

Bu cebirsel ifadeyi çarpanlarına ayırırsak:

$$b^3 + 3b^2 + 3b + 1 = (b + 1)^3$$

Görüldüğü üzere, $b > 3$ koşulunu sağlayan **bütün** tabanlarda $(1331)_b$ sayısı $(b+1)$ tamsayısının küpüne eşittir. Dolayısıyla aranan tabanlar $b \geq 5$ (veya sorudaki kısıta göre $b > 4$) şartını sağlayan **tüm tamsayılardır**. □

### 15.1.5 Uygulama: Taban Dönüşüm Kodu

Aşağıdaki C programı, pozitif bir onluk tamsayının $b$ ($2 \leq b \leq 16$) tabanındaki gösterimini ardışık bölme algoritmasıyla hesaplar:

```c
#include <stdio.h>

void tabana_cevir(int n, int b) {
    char rakamlar[] = "0123456789ABCDEF";
    char sonuc[64];
    int indis = 0;

    if (n == 0) {
        printf("0\n");
        return;
    }

    while (n > 0) {
        sonuc[indis++] = rakamlar[n % b];
        n = n / b;
    }

    // Kalanlar tersten yazdirilmali
    printf("(%d) tabanindaki degeri: ", b);
    for (int i = indis - 1; i >= 0; i--) {
        putchar(sonuc[i]);
    }
    printf("\n");
}
```

**Örnek 15.9** (En Sık Yapılan Hatalar)**.** *Taban dönüşümlerinde sıklıkla karşılaşılan hatalar şunlardır:*

1. ***Kalanları düz sırada yazmak:*** *Ardışık bölmede ilk elde edilen kalan, sayının en küçük basamağıdır ($b^0$). Dolayısıyla dizi ters çevrilerek okunmalıdır.*
2. ***Geçersiz rakam kullanımı:*** *Taban $b$ iken basamaklarda $b$ veya daha büyük bir rakam bulunamaz (örneğin 5 tabanında $(152)_5$ tanımsızdır).*
3. ***Sıfırları atlamak:*** *Bölüm adımlarında kalan 0 çıktığında bunu diziye eklemeyi unutmak basamak kaymasına yol açar (örneğin $4 = (100)_2$ iken sıfırları yazmayıp 1 demek gibi).*

## 15.2 Tabanlarda Aritmetik ve İkili Sistemin Rolü

### 15.2.1 Farklı Tabanlarda Dört İşlem

Pozisyonel sayı sistemlerinde aritmetik algoritmalar tabandan bağımsızdır: Onluk sistemdeki alt alta toplama, çıkarma ve çarpma kuralları, basamak ağırlığı 10

yerine $b$ alınarak doğrudan uygulanır.

Toplama ve Elde (Carry) Mekanizması: Aynı basamaktaki iki rakam toplandığında elde edilen değer $S = a_i + c_i +$ elde olsun:

- Eğer $S < b$ ise basamağa $S$ yazılır, yeni elde 0 olur.
- Eğer $S \geq b$ ise basamağa $S \pmod b$ yazılır ve bir sonraki basamağa $\lfloor S/b \rfloor$ eldesi aktarılır.

**Örnek 15.10.** $(564)_7$ *ile* $(345)_7$ *sayılarını* 7 *tabanında toplayınız.*

*Çözüm.* İşlemi sağdan sola basamak basamak yürütelim:

- $7^0$ **basamağı:** $4 + 5 = 9$. Taban 7 olduğundan: $9 = 1 \cdot 7 + 2$. Basamağa 2 yazarız, elde $= 1$.
- $7^1$ **basamağı:** $6 + 4 + 1$ (elde) $= 11$. $11 = 1 \cdot 7 + 4$. Basamağa 4 yazarız, elde $= 1$.
- $7^2$ **basamağı:** $5 + 3 + 1$ (elde) $= 9$. $9 = 1 \cdot 7 + 2$. Basamağa 2 yazarız, elde $= 1$.
- $7^3$ **basamağı:** Yalnızca elde kalmıştır: 1.

Sonuç: $(1242)_7$. □

Çıkarma ve Borç (Borrow) Mekanizması: Bir basamaktaki eksilen rakam çıkan rakamdan küçükse, sol basamaktan "1 borç" alınır. Onluk tabanda borç alındığında basamağa 10 eklenirken, $b$ tabanında o basamağa $b$ eklenir.

**Örnek 15.11.** $(1001)_2 - (0111)_2$ *çıkarma işlemini ikili tabanda yapınız.*

*Çözüm.* Sağdan sola inceleyelim:

- $2^0$ **basamağı:** $1 - 1 = 0$.
- $2^1$ **basamağı:** $0 - 1$ yapılamaz, soldan borç aranır. $2^2$ basamağında 0 olduğundan borç $2^3$ basamağındaki 1'den alınır ve sağa aktarılır. $2^1$ basamağına 2 eklenir: $2 - 1 = 1$.
- $2^2$ **basamağı:** Borç aktarımı sonrasında burada 1 kalmıştır: $1 - 1 = 0$.
- $2^3$ **basamağı:** 1 borç verildiğinden geriye 0 kalmıştır: $0 - 0 = 0$.

Sonuç: $(0010)_2 = (10)_2$. Onluk sistemde sağlama: $9 - 7 = 2$. □

### 15.2.2 Genel Tabanlarda Bölünebilme Kuralları

Onluk sistemden bildiğimiz 9'a bölünebilme kuralı (rakamlar toplamının 9'a bölünmesi), $10 \equiv 1 \pmod 9$ bağıntısının bir sonucudur. Bölüm 14'te ele aldığımız modüler aritmetik araçlarını taban aritmetiğiyle birleştirdiğimizde genel bölünebilme kuralları kendiliğinden ortaya çıkar.

**Teorem 15.12** ($b-1$ ve $b+1$ ile Bölünebilme)**.** *$N = (a_k a_{k-1} \dots a_1 a_0)_b$ sayısı verilsin.*

1. $N \equiv \sum_{i=0}^{k} a_i \pmod{b-1}$. *Yani $N$, basamakları toplamı ile $(b-1)$ modülünde denktir.*
2. $N \equiv \sum_{i=0}^{k} (-1)^i a_i \pmod{b+1}$. *Yani $N$, birler basamağından başlayarak bir artı bir eksi alınarak kurulan almaşık basamak toplamı ile $(b+1)$ modülünde denktir.*

*Kanıt.* $b \equiv 1 \pmod{b-1}$ olduğundan her $i \geq 0$ için $b^i \equiv 1^i \equiv 1 \pmod{b-1}$ olur. Polinom açılımında her $b^i$ yerine 1 yazarsak:

$$N = \sum_{i=0}^{k} a_i b^i \equiv \sum_{i=0}^{k} a_i (1)^i = \sum_{i=0}^{k} a_i \pmod{b-1}$$

İkinci kısım için, $b \equiv -1 \pmod{b+1}$ olduğundan $b^i \equiv (-1)^i \pmod{b+1}$ elde edilir:

$$N = \sum_{i=0}^{k} a_i b^i \equiv \sum_{i=0}^{k} a_i (-1)^i = a_0 - a_1 + a_2 - a_3 + \dots \pmod{b+1}$$

İspat tamamlanır. □

Bu teorem pratik sonuçlar üretir:

- 10 tabanında: $10 - 1 = 9$ için basamaklar toplamı, $10 + 1 = 11$ için almaşık toplam kuralı çıkar.
- 2 tabanında: $b - 1 = 1$ aşikardır; ancak $b + 1 = 3$ için kullanışlı bir kural verir: İkili tabandaki bir sayının çift sıradaki bitleri toplamı ile tek sıradaki bitleri toplamının farkı 3'e bölünüyorsa, sayı 3'e tam bölünür.

**Örnek 15.13.** *İkili tabanda yazılmış $N = (101011)_2$ sayısı 3'e tam bölünür mü?*

*Çözüm.* Bitleri sağdan sola (0. bitten başlayarak) numaralandıralım:

- $a_0 = 1$ (çift)
- $a_1 = 1$ (tek)

- $a_2 = 0$ (çift)
- $a_3 = 1$ (tek)
- $a_4 = 0$ (çift)
- $a_5 = 1$ (tek)

Almaşık toplam:

$$(a_0 + a_2 + a_4) - (a_1 + a_3 + a_5) = (1 + 0 + 0) - (1 + 1 + 1) = 1 - 3 = -2$$

$-2 \not\equiv 0 \pmod 3$ olduğundan sayı 3'e tam bölünmez; kalan $-2 \equiv 1 \pmod 3$'tür. Onluk sistemde kontrol edelim: $(101011)_2 = 32 + 8 + 2 + 1 = 43$. Gerçekten $43 = 3 \cdot 14 + 1$ olup kalan 1'dir. □

### 15.2.3 İkili Sistemin Bilgisayar Bilimindeki Rolü

Bilgisayar biliminde algoritmik verimliliğin temelinde ikili sistem yer alır. Donanım düzeyinde çarpma ve bölme işlemleri görece maliyetliyken, bit kaydırma ve mantıksal işlemler tek bir saat çevriminde ($O(1)$) çalışır.

Bit Kaydırma (Bit Shifting):

- **Sola kaydırma (`x << k`):** Sayının ikili gösteriminin sağına $k$ tane sıfır ekler. Bu işlem sayıyı $2^k$ ile çarpmaya denktir:

$$x \ll k = x \cdot 2^k$$

- **Sağa kaydırma (`x >> k`):** Sayının son $k$ bitini atar. Bu işlem pozitif tamsayılarda $\lfloor x/2^k \rfloor$ bölmesine denktir:

$$x \gg k = \lfloor x/2^k \rfloor$$

Bit Düzeyinde İşlemler (Bitwise Operators): İki sayının ikili gösterimlerindeki karşılıklı bitler mantıksal kapılardan geçirilir:

1. **AND (`&`):** Yalnızca her iki bit de 1 ise 1 üretir.
2. **OR (`|`):** Bitlerden en az biri 1 ise 1 üretir.
3. **XOR (`^`):** Bitler birbirinden farklı ise 1, aynı ise 0 üretir (eldesiz ikili toplama).
4. **NOT (`~`):** Bitleri tersine çevirir ($0 \to 1, 1 \to 0$).

Kümelerin Bitmask ile Gösterimi: $n$ elemanlı bir evrensel kümenin $S = \{e_0, e_1, \ldots, e_{n-1}\}$ herhangi bir alt kümesi, $n$ bitlik bir ikili sayı ile birebir eşlenebilir: $i$. eleman kümede varsa $i$. bit 1, yoksa 0 yapılır. Örneğin $n = 4$ için $\{e_0, e_2\}$ alt kümesi $(0101)_2 = 5$ tamsayısı ile temsil edilir. Böylece küme işlemleri doğrudan bit operatörlerine dönüşür:

- Kesişim işlemi: `A & B`
- Birleşim işlemi: `A | B`
- Simetrik fark: `A ^ B`
- Boş küme: `0`
- Evrensel küme: `(1 << n) - 1`

Bu sayede $O(n)$ sürecek küme işlemleri tek bir makine komutuyla $O(1)$ sürede tamamlanır.

### 15.2.4 Hızlı Üs Alma (Binary Exponentiation)

İkili tabanın algoritmik problem çözmedeki en etkili uygulamalarından biri **Hızlı Üs Alma** algoritmasıdır. $a^n$ değerini ardışık çarpma ile hesaplamak $n-1$ çarpma, yani $O(n)$ zaman gerektirir. $n = 10^{18}$ gibi büyük girdiler için bu yaklaşım çalışmaz.

$n$ sayısını ikili tabanda açarsak:

$$n = \sum_{i=0}^{k} c_i 2^i \implies a^n = a^{\sum c_i 2^i} = \prod_{c_i=1} a^{2^i}$$

Her bir $a^{2^i}$ terimi, bir önceki terimin karesi alınarak ($a^{2^i} = (a^{2^{i-1}})^2$) tek bir çarpmayla elde edilebilir. Böylece üs alma işlemi $O(\log n)$ adıma iner.

**Örnek 15.14.** $3^{13}$ *sayısını hızlı üs alma yöntemiyle hesaplayınız.*

*Çözüm.* Üssü ikili tabanda yazalım: $13 = 8 + 4 + 1 = (1101)_2$. Buradan:

$$3^{13} = 3^8 \cdot 3^4 \cdot 3^1$$

Ardışık kareleri hesaplayalım:

$$3^1 = 3$$
$$3^2 = 3^2 = 9$$
$$3^4 = 9^2 = 81$$
$$3^8 = 81^2 = 6561$$

İkili gösterimde 1 olan basamaklara karşılık gelen değerleri çarparsak:

$$3^{13} = 3^8 \cdot 3^4 \cdot 3^1 = 6561 \cdot 81 \cdot 3 = 6561 \cdot 243 = 1\,594\,323$$

Normalde 12 çarpma gerekecekken, 3 kare alma ve 2 çarpma ile toplam 5 işlemde sonuca ulaşılır. □

Aşağıdaki sözde kod, hızlı üs almanın ikili bit kaydırmayla nasıl gerçeklendiğini özetler:

```
FUNCTION modularExponentiation(base, exponent, modulus)
    result = 1
    base = base MOD modulus

    WHILE exponent > 0 DO
        IF (exponent AND 1) == 1 THEN
            result = (result * base) MOD modulus
        END IF
        base = (base * base) MOD modulus
        exponent = exponent >> 1
    END WHILE

    RETURN result
END FUNCTION
```

### 15.2.5 Kombinatorik ve Sayı Teorisinde İkili Taban: Kummer ve Lucas Sezgisi

İkili taban, binom katsayılarının tekliği-çiftliği (paritesi) konusunda pratik sonuçlar sunar.

**Teorem 15.15** (Sierpiński / Lucas Parite Teoremi)**.** $\binom{n}{k}$ *binom katsayısının tek sayı olması için gerek ve yeter koşul, $k$'nin ikili gösterimindeki her bir* 1 *bitinin, $n$'nin ikili gösteriminde de aynı pozisyonda* 1 *olmasıdır (yani $k$ AND $n = k$; başka bir deyişle $k$, bitwise $n$'nin alt kümesi olmalıdır).*

*Fikir / Sezgi.* Bölüm 12'de gördüğümüz Legendre formülüne göre bir $m!$ içindeki $p$ asalının kuvveti $v_p(m!) = \sum_{j=1}^{\infty} \lfloor \frac{m}{p^j} \rfloor$ ile verilir. Özel olarak $p = 2$ için bu toplam:

$$v_2(m!) = m - S_2(m)$$

biçimine dönüşür; burada $S_2(m)$, $m$ sayısının ikili tabandaki rakamları toplamıdır (1 bitlerinin sayısı, diğer adıyla popcount).

Binom katsayısı $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ olduğundan, 2 ile bölünebilme derecesi:

$$v_2\left(\binom{n}{k}\right) = v_2(n!) - v_2(k!) - v_2((n-k)!)$$

$$= \big(n - S_2(n)\big) - \big(k - S_2(k)\big) - \big(n - k - S_2(n-k)\big)$$

$$= S_2(k) + S_2(n-k) - S_2(n)$$

olur. İkili tabanda $k$ ile $n-k$'yi toplarken her elde oluştuğunda, iki tane 1 biti birleşip üst basamağa tek bir 1 olarak geçer; yani toplam bit sayısı 1 azalır. Dolayısıyla $S_2(k) + S_2(n-k) - S_2(n)$, $k$ ile $n-k$ toplanırken oluşan **toplam elde sayısıdır** (Kummer Teoremi).

$\binom{n}{k}$'nın tek olması için $v_2 = 0$ olmalıdır; bu da toplama sırasında **hiç elde oluşmaması** anlamına gelir. İkili toplamda hiç elde çıkmaması ise ancak ve ancak $k$'nin 1 olduğu hiçbir basamakta $n-k$'nin 1 olmamasıyla mümkündür. Bu da $n$'in o basamakta zorunlu olarak 1 olmasını gerektirir. □

**Örnek 15.16.** $\binom{11}{k}$ *sayısının tek sayı olduğu kaç farklı* $k \in \{0, 1, \ldots, 11\}$ *değeri vardır?*

*Çözüm.* Teorem gereği $k$'nin ikili gösterimi, 11'in ikili gösteriminin bir alt kümesi olmalıdır. 11'i ikili tabana çevirelim:

$$11 = 8 + 2 + 1 = (1011)_2$$

11'in ikili yazımında 3 tane 1 biti vardır ($2^3, 2^1, 2^0$ basamakları). $k$'nin ikili yazımındaki 1 bitleri yalnızca bu üç pozisyondan seçilebilir; diğer basamaklar ($2^2$ ve daha büyükler) 0 kalmalıdır.

Bu üç pozisyonun her biri için iki seçenek vardır: 1 bitini alabiliriz veya almayız. Dolayısıyla yazılabilecek geçerli $k$ sayısı:

$$2^3 = 8$$

farklı değerdir. Bu değerler sırasıyla:

$$(0000)_2 = 0$$
$$(0001)_2 = 1$$
$$(0010)_2 = 2$$
$$(0011)_2 = 3$$
$$(1000)_2 = 8$$
$$(1001)_2 = 9$$
$$(1010)_2 = 10$$
$$(1011)_2 = 11$$

Böylece Pascal üçgeninin 11. satırında tam olarak 8 adet tek sayı, 4 adet çift sayı bulunduğu doğrulanır. □

### 15.2.6 Negatif Tabanlar ve Alternatif Sistemler

Yarışmalarda zaman zaman tabanın pozitif tamsayı olmadığı durumlar da karşımıza çıkabilir. Öne çıkan örneklerden biri **negatif tabanlar** ($b = -2$, negabinary) sistemidir. Negatif tabanların dikkat çekici özelliği, **eksi işaretine gerek kalmadan** hem pozitif hem de negatif tüm tamsayıları tek bir biçimde temsil edebilmesidir.

$b = -2$ olduğunda basamak değerleri:

$$(-2)^0 = 1, \quad (-2)^1 = -2, \quad (-2)^2 = 4, \quad (-2)^3 = -8, \quad (-2)^4 = 16, \ldots$$

olur. Kalanlar negatif olmayan $\{0, 1\}$ kümesinden seçilmelidir:

$$N = q \cdot (-2) + r, \quad r \in \{0, 1\}$$

Standart tamsayı bölmesinde kalan negatif çıkarsa (örneğin $-5 \div (-2) = 2$, kalan $-1$), bölüm 1 artırılır ve kalan pozitif yapılır:

$$-5 = 3 \cdot (-2) + 1 \implies q = 3, r = 1$$

**Örnek 15.17.** $-6$ *sayısını* $-2$ *tabanında (negabinary) yazınız.*

*Çözüm.* Kalanların daima 0 veya 1 olmasına dikkat ederek ardışık bölme uygulayalım:

$$\begin{aligned} -6 &= 3 \cdot (-2) + 0 \quad \text{(kalan 0)} \\ 3 &= (-1) \cdot (-2) + 1 \quad \text{(kalan 1)} \\ -1 &= 1 \cdot (-2) + 1 \quad \text{(kalan 1)} \\ 1 &= 0 \cdot (-2) + 1 \quad \text{(kalan 1)} \end{aligned}$$

Kalanları aşağıdan yukarıya doğru okursak:

$$-6 = (1110)_{-2}$$

Sağlamasını yapalım:

$$1 \cdot (-2)^3 + 1 \cdot (-2)^2 + 1 \cdot (-2)^1 + 0 \cdot (-2)^0 = -8 + 4 - 2 + 0 = -6$$

Sonuç doğrulanır. □

Taban aritmetiği yalnızca sayıların yazım biçimini belirlemekle kalmaz; ardışık bölme algoritmasından modüler aritmetiğe, bitwise işlemlerden dinamik programlamanın bitmask optimizasyonlarına kadar algoritmik düşüncenin temel yapı taşlarından birini oluşturur.

# Bölüm 16

# Logaritma

Şimdiye kadar bir sayının kuvvetini hep ileri yönde hesapladık: $2^5$'in kaç ettiğini bulmak için 2'yi beş kez çarpmak yeterlidir. Oysa sorular bazen tam ters yönden gelir. "2'nin kaçıncı kuvveti 32 eder?" diye sorulduğunda artık çarpma değil, *üs arama* işi yaparız. Aynı ihtiyaç sayılar büyüdükçe daha da belirginleşir: $2^{100}$ sayısının kaç basamaklı olduğunu, ya da sıralı bir dizide ikili arama yaparken kaç adım atacağımızı merak ettiğimizde de karşımıza aynı soru çıkar.

Bu ters sorunun cevabı **logaritma**dır. Logaritma, üs almanın ters işlemidir; tam olarak bu yüzden çarpma ile toplama arasında bir köprü kurar. Tarihsel olarak büyük çarpımları elle yapmak için icat edilmiş olması tesadüf değildir: logaritma sayesinde 100 basamaklı iki sayıyı çarpmak, iki sayıyı toplamaya indirgenir.

Bu bölümü bilinçli olarak kısa tuttuk. Amaç logaritmayı bütün ayrıntılarıyla işlemek değil; tanımını, çarpma–toplama ilişkisini ve taban değiştirmeyi rahatça kullanabilecek kadar sağlam biçimde öğrenmektir. Bu üç araç, kitabın ilerleyen bölümlerinde (özellikle Bölüm 21'de göreceğimiz $O(\log n)$ karmaşıklığında ve Bölüm 24'teki ikili aramada) sürekli karşımıza çıkacaktır.

## 16.1 Logaritma Nedir?

2'nin kuvvetlerini alt alta yazalım:

$$2^1 = 2, \qquad 2^2 = 4, \qquad 2^3 = 8, \qquad 2^4 = 16, \qquad 2^5 = 32, \qquad 2^6 = 64.$$

Bu dizide 32 sayısı beşinci kuvvete, 64 ise altıncı kuvvete karşılık gelir. Demek ki "2'nin kaçıncı kuvveti 32 eder?" sorusunun cevabı 5'tir; bunu $\log_2 32 = 5$ biçiminde yazarız ve "32'nin 2 tabanında logaritması 5" diye okuruz. Benzer biçimde $\log_2 64 = 6$ ve $\log_2 8 = 3$ olur.

Aynı fikir 10 tabanında daha da tanıdıktır. 10'un kuvvetleri $10, 100, 1000, \ldots$ olduğundan

$$\log_{10} 10 = 1, \qquad \log_{10} 100 = 2, \qquad \log_{10} 1000 = 3$$

eşitlikleri geçerlidir. Yani 10 tabanında logaritma, kabaca "sayıyı yazarken kaç basamak kullandığımızı" sayar; bu gözlemi birazdan bir uygulamada yeniden kullanacağız.

Somut örneklerden genel tanıma geçelim.

**Tanım 16.1** (Logaritma)**.** *$a > 0$ ve $a \neq 1$ bir taban, $x > 0$ bir sayı olmak üzere $y = \log_a x$ ifadesi*

$$\log_a x = y \iff a^y = x$$

*denklemini sağlayan y sayısını gösterir. Burada a'ya logaritmanın* ***tabanı*** *(base), x'e ise logaritmanın* ***argümanı*** *denir.*

Tanımı bir kez sindirdikten sonra şu üç sonuç doğrudan çıkar:

- **Tabanın logaritması 1'dir:** $a^1 = a$ olduğundan $\log_a a = 1$.
- **Birin logaritması 0'dır:** $a^0 = 1$ olduğundan $\log_a 1 = 0$.
- **Üs ile logaritma birbirini götürür:** $\log_a x = y$ ise $a^{\log_a x} = a^y = x$ olur. Yani $a^{\log_a x} = x$'tir.

Ayrıca $\log_a a^k = k$ eşitliği de tanımın hemen bir sonucudur; çünkü $a$'nın $k$'ıncı kuvveti $a^k$'dır.

**Uyarı 2.** *En sık yapılan hata, logaritmanın toplama üzerine dağıldığını sanmaktır. Genel olarak*

$$\log_a(x + y) \neq \log_a x + \log_a y$$

*olur. Somut bir sayı ile görelim:* $\log_2(4 + 4) = \log_2 8 = 3$ *iken* $\log_2 4 + \log_2 4 = 2 + 2 = 4$*'tür. Logaritmayı toplamaya değil çarpmaya çeviren kuralı bir sonraki alt konuda ispatlayacağız.*

**Not 1.** *Tanımdaki $a \neq 1$ koşulu keyfî değildir. Taban $a = 1$ seçilseydi $1^y = 1$ eşitliği her y için sağlanırdı; o zaman "1'in kaçıncı kuvveti x eder?" sorusunun tek bir cevabı olmazdı. Benzer biçimde argümanın $x > 0$ olması da zorunludur: $a^y$ ifadesi her zaman pozitiftir, dolayısıyla negatif bir sayının logaritması tanımlı değildir.*

**Örnek 16.2.** *Aşağıdaki logaritmaları tanımı kullanarak hesaplayınız:*

$$\textit{(a)}\ \log_3 81, \qquad \textit{(b)}\ \log_2 \frac{1}{32}, \qquad \textit{(c)}\ \log_5 1, \qquad \textit{(d)}\ \log_{10} 0{,}001.$$

*Çözüm.* (a) $81 = 3^4$ olduğundan $\log_3 81 = 4$.

(b) $\frac{1}{32} = 2^{-5}$ olduğundan $\log_2 \frac{1}{32} = -5$. Görüldüğü gibi logaritma negatif de olabilir; negatif olan, argümanın 1'den küçük olmasıdır.

(c) Her tabanda $a^0 = 1$ olduğundan $\log_5 1 = 0$.

(d) $0{,}001 = 10^{-3}$ olduğundan $\log_{10} 0{,}001 = -3$. □

**Örnek 16.3.** $\log_2 10$ *sayısının tam sayı olmadığını gösteriniz ve hangi iki ardışık tam sayı arasında bulunduğunu belirleyiniz.*

*Çözüm.* 2'nin kuvvetlerini anımsayalım: $2^3 = 8$ ve $2^4 = 16$. $8 < 10 < 16$ olduğundan, $2^y = 10$ eşitliğini sağlayan $y$ değeri 3 ile 4 arasında olmalıdır. Öte yandan $y$ bir tam sayı olsaydı $2^y$ ya 8 ya da en az 16 olurdu. Demek ki $3 < \log_2 10 < 4$ olup $\log_2 10$ tam sayı değildir. □

**Örnek 16.4** (Basamak Sayısı)**.** $2^{100}$ *sayısı onluk tabanda kaç basamaklıdır?*

*Çözüm.* $d$ basamaklı bir $N$ doğal sayısı, tam olarak $10^{d-1} \leq N < 10^d$ eşitsizliğini sağlayan sayıdır. Bu eşitsizliğin tüm taraflarının 10 tabanında logaritmasını alırsak

$$d - 1 \leq \log_{10} N < d$$

elde ederiz; yani $d = \lfloor \log_{10} N \rfloor + 1$ olur. Şimdi $N = 2^{100}$ için $\log_{10} 2 \approx 0{,}30103$ değerini kullanalım:

$$\log_{10} 2^{100} = 100 \cdot \log_{10} 2 \approx 100 \cdot 0{,}30103 = 30{,}103.$$

Tam kısmı 30 olduğundan basamak sayısı $30 + 1 = 31$'dir. Bu hesapta kullandığımız $\log_{10} 2^{100} = 100 \log_{10} 2$ eşitliğini bir sonraki alt konuda ispatlayacağız. □

## 16.2 Temel Kurallar: Çarpma, Bölme ve Kuvvet

Logaritmanın bütün gücü şu tek gözlemden gelir: *aynı tabanda iki kuvvet çarpılırken üsler toplanır.* Örneğin

$$2^3 \cdot 2^4 = 8 \cdot 16 = 128 = 2^7$$

olur ve buradaki 7 üssü, $3 + 4$ toplamından başka bir şey değildir. Aynı eşitliği logaritma dilinde yazalım:

$$\log_2(8 \cdot 16) = \log_2 128 = 7 = 3 + 4 = \log_2 8 + \log_2 16.$$

Görüldüğü gibi logaritma, soldaki *çarpma* işlemini sağdaki *toplama* işlemine çevirdi. Bu, tanımın bir sonucudur ve aşağıdaki teoremle genelleşir.

**Teorem 16.5** (Çarpım Kuralı)**.** *Her $a > 0$, $a \neq 1$ tabanı ve her $x, y > 0$ için*

$$\log_a(x \cdot y) = \log_a x + \log_a y$$

*eşitliği geçerlidir.*

*Kanıt.* $m = \log_a x$ ve $n = \log_a y$ diyelim. Logaritmanın tanımı gereği bu iki eşitlik

$$a^m = x \qquad \text{ve} \qquad a^n = y$$

demektir. Şimdi $x$ ile $y$'yi çarpalım:

$$x \cdot y = a^m \cdot a^n = a^{m+n}.$$

Elde ettiğimiz $a^{m+n} = xy$ eşitliğini yeniden tanıma uygularsak $\log_a(xy) = m + n$ bulunur; bu da $\log_a x + \log_a y$ demektir. □

Aynı akıl yürütme, bölme için de bölümün üssünün fark olduğu gözlemiyle ($a^m/a^n = a^{m-n}$) yürütülür.

**Sonuç 16.6** (Bölüm Kuralı)**.** *Her $x, y > 0$ için*

$$\log_a \frac{x}{y} = \log_a x - \log_a y$$

*olur. Özel olarak $x = 1$ seçilirse $\log_a \frac{1}{y} = -\log_a y$ çıkar.*

*Kanıt.* $m = \log_a x$, $n = \log_a y$ olsun; yine $x = a^m$ ve $y = a^n$'dir. Bölümü yazalım:

$$\frac{x}{y} = \frac{a^m}{a^n} = a^{m-n}.$$

Tanım gereği $\log_a \frac{x}{y} = m - n = \log_a x - \log_a y$ olur. $x = 1$ durumunda $\log_a 1 = 0$ olduğundan $\log_a \frac{1}{y} = -\log_a y$ elde edilir. □

**Teorem 16.7** (Kuvvet Kuralı)**.** *Her $k$ tam sayısı ve her $x > 0$ için*

$$\log_a x^k = k \cdot \log_a x$$

*eşitliği geçerlidir. Bu kural $k = 0$ (bu durumda $x^0 = 1$) ve negatif $k$ değerleri için de doğrudur.*

*Kanıt.* Önce $k \geq 1$ olsun. $x^k = \underbrace{x \cdot x \cdots x}_{k \text{ tane}}$ çarpımına çarpım kuralını $k - 1$ kez uygularsak

$$\log_a x^k = \underbrace{\log_a x + \log_a x + \cdots + \log_a x}_{k \text{ terim}} = k \log_a x$$

bulunur. $k = 0$ için sol taraf $\log_a 1 = 0$, sağ taraf $0 \cdot \log_a x = 0$ olduğundan eşitlik korunur. Son olarak $k$ negatifse $x^k = 1/x^{-k}$ yazıp bölüm kuralını ve az önce bulduğumuz pozitif durumu kullanalım:

$$\log_a x^k = -\log_a x^{-k} = -(-k) \log_a x = k \log_a x.$$

Böylece kural tüm tam sayı $k$ değerleri için geçerli olur. □

**Not 2.** *Kuvvet kuralı, karmaşıklık hesabında işimizi çok kolaylaştırır. Örneğin* $\log_2 n^2 = 2\log_2 n$ *olduğundan,* $n^2$ *büyüklüğündeki bir girdide "logaritma kadar adım" atmak ile* $n$ *büyüklüğündeki bir girdide atmak, sabit bir çarpan dışında aynı iştir. Büyük* $O$ *gösteriminin bu yüzden* $O(\log n)$ *derken tabanı belirtmemesi, bir sonraki alt konuda göreceğimiz taban değiştirme formülüyle netleşecektir.*

**Örnek 16.8.** *Aşağıdaki ifadeleri tek bir logaritma biçiminde yazıp değerini bulunuz:*

*(a)* $\log_{10} 2 + \log_{10} 5$, *(b)* $\log_3 54 - \log_3 2$, *(c)* $\log_2 8 + \log_2 16 - \log_2 4$.

*Çözüm.* (a) Çarpım kuralı doğrudan $\log_{10}(2 \cdot 5) = \log_{10} 10 = 1$ verir.

(b) Bölüm kuralına göre $\log_3 \frac{54}{2} = \log_3 27 = 3$ olur.

(c) Terimleri sırayla sadeleştirelim: $\log_2 8 = 3$, $\log_2 16 = 4$, $\log_2 4 = 2$. Buna göre ifade $3 + 4 - 2 = 5$'tir. Doğrulaması için $2^5 = 32$ olup $\log_2 32 = 5$'tir; çarpım kuralıyla da $\log_2 \frac{8 \cdot 16}{4} = \log_2 32$ elde edilir. □

**Örnek 16.9.** $\log_2 32^3$ *ifadesini hesaplayınız ve bulduğunuz sonucu doğrudan doğrulayınız.*

*Çözüm.* Kuvvet kuralına göre

$$\log_2 32^3 = 3\log_2 32 = 3 \cdot 5 = 15$$

bulunur. Doğrudan doğrulama: $32^3 = (2^5)^3 = 2^{15}$, dolayısıyla tanım gereği $\log_2 2^{15} = 15$'tir. Demek ki üssü öne indirmek, üssü ayrı ayrı hesaplayıp logaritma almaktan çok daha hızlıdır. □

**Örnek 16.10.** $2^{\log_2 5 + \log_2 3}$ *ifadesinin değerini bulunuz.*

*Çözüm.* Üsleri önce çarpım kuralıyla birleştirelim:

$$\log_2 5 + \log_2 3 = \log_2 15.$$

Böylece ifade $2^{\log_2 15}$ olur. Bir $x$ sayısı için $a^{\log_a x} = x$ olduğunu tanımdan biliyoruz; burada $a = 2$ ve $x = 15$ olduğundan cevap 15'tir. □

**Örnek 16.11.** $\log_2(x-2) + \log_2(x+2) = 5$ *denklemini çözünüz.*

*Çözüm.* Denklemin anlamlı olması için her iki argümanın da pozitif olması gerekir: $x - 2 > 0$ ve $x + 2 > 0$, yani $x > 2$. Çarpım kuralını uygulayalım:

$$\log_2(x-2)(x+2) = 5 \implies \log_2(x^2 - 4) = 5.$$

Tanım gereği $\log_2(x^2 - 4) = 5$, $x^2 - 4 = 2^5 = 32$ demektir. Buradan $x^2 = 36$, yani $x = 6$ veya $x = -6$ bulunur. Ancak $x > 2$ koşulu $x = -6$ değerini eler. Dolayısıyla tek çözüm $x = 6$'dır. Sağlama: $\log_2 4 + \log_2 8 = 2 + 3 = 5$. □

## 16.3 Taban Değiştirme

Şimdiye kadar bütün hesapları tabanı uygun seçerek yaptık. Ama bir ifadede 2, başka bir ifadede 4 tabanı geçtiğinde iki sayıyı karşılaştırmak için ortak bir dile ihtiyaç duyarız. Bu ortak dili kuran araç *taban değiştirme*dir.

Somut bir gözlemle başlayalım. $\log_2 8 = 3$ idi. Aynı soruyu 8 tabanında soralım: 8'in hangi kuvveti 2 eder? 8'in kuvvetleri $8, 64, \ldots$ biçiminde hızla büyüdüğüne göre üssün 1'den küçük olması gerekir. Gerçekten de $8^{1/3} = 2$ olduğundan

$$\log_8 2 = \frac{1}{3}$$

olur. Dikkat çekici olan şudur: sonuç, $\log_2 8 = 3$ sayısının tersidir.

Bir başka örnek, hem taban hem argümanın değiştiği durumu gösterir: $4^{3/2} = (\sqrt{4})^3 = 2^3 = 8$ olduğundan $\log_4 8 = \frac{3}{2}$'dir. Peki bu 3/2 değerini, elimizdeki $\log_2$ bilgisinden bulmanın genel bir yolu var mı? Aşağıdaki teorem tam olarak bunu söyler.

**Teorem 16.12** (Taban Değiştirme)**.** *$a, b > 0$, $a \neq 1$, $b \neq 1$ ve $x > 0$ olsun. Bu durumda*

$$\log_a x = \frac{\log_b x}{\log_b a}$$

*eşitliği geçerlidir.*

*Kanıt.* $y = \log_a x$ diyelim. Tanım gereği $a^y = x$'tir. Şimdi bu eşitliğin her iki tarafının da $b$ tabanında logaritmasını alalım:

$$\log_b a^y = \log_b x.$$

Sol tarafta kuvvet kuralını uygularsak $\log_b a^y = y \cdot \log_b a$ olur; burada $a \neq 1$ olduğundan $\log_b a \neq 0$'dır ve bölme yapılabilir:

$$y \cdot \log_b a = \log_b x \implies y = \frac{\log_b x}{\log_b a}.$$

$y$ yerine $\log_a x$ yazarak istediğimiz eşitliği elde ederiz. □

**Sonuç 16.13.** *Aynı koşullar altında*

$$\log_a b = \frac{1}{\log_b a}$$

*olur. Ayrıca $\log_a b \cdot \log_b c = \log_a c$ eşitliği geçerlidir.*

*Kanıt.* İlk eşitlik için taban değiştirme formülünde $x = b$ seçilir: $\log_a b = \frac{\log_b b}{\log_b a} = \frac{1}{\log_b a}$. İkinci eşitlik için taban değiştirme formülünü $\log_a b = \frac{\log_b b}{\log_b a}$ biçiminde yazalım; $\log_b b = 1$ olduğundan bu ifade $\frac{1}{\log_b a}$'ya indirgenir. Bunu $\log_b c$ ile çarpıp son

adımda taban değiştirme formülünü bir kez daha kullanalım:

$$\log_a b \cdot \log_b c = \frac{1}{\log_b a} \cdot \log_b c = \frac{\log_b c}{\log_b a} = \log_a c.$$

Bu son eşitlik, zincirleme sadeleşen çarpımlarda çok işe yarar. □

**Örnek 16.14.** *Aşağıdaki değerleri, taban değiştirme formülüyle* $\log_2$ *cinsinden hesaplayınız:*

$$\textit{(a)}\ \log_4 8, \qquad \textit{(b)}\ \log_{16} 2, \qquad \textit{(c)}\ \log_{1/2} 8.$$

*Çözüm.* (a) $\log_4 8 = \frac{\log_2 8}{\log_2 4} = \frac{3}{2}$, beklendiği gibi.

(b) $\log_{16} 2 = \frac{\log_2 2}{\log_2 16} = \frac{1}{4}$. Doğrulaması: $16^{1/4} = 2$.

(c) Taban 1'den küçük olduğunda da formül aynen çalışır: $\log_{1/2} 8 = \frac{\log_2 8}{\log_2 \frac{1}{2}} = \frac{3}{-1} = -3$. İkinci teyit: $\left(\frac{1}{2}\right)^{-3} = 2^3 = 8$. □

**Örnek 16.15.** $\log_2 5 + \log_4 25$ *ifadesini tek bir logaritma biçiminde yazınız.*

*Çözüm.* İki terimi de 2 tabanına çevirelim. Birinci terim zaten 2 tabanındadır. İkinci terim için taban değiştirme formülünü kullanalım:

$$\log_4 25 = \frac{\log_2 25}{\log_2 4} = \frac{2\log_2 5}{2} = \log_2 5.$$

(Doğrudan bir kısayol da vardır: hem taban hem argüman aynı anda kare alınırsa değer değişmez, yani $\log_{a^2} x^2 = \log_a x$.) Buna göre toplam

$$\log_2 5 + \log_4 25 = \log_2 5 + \log_2 5 = \log_2 25$$

olur. □

**Örnek 16.16.**

$$\log_2 3 \cdot \log_3 4 \cdot \log_4 5 \cdot \log_5 6 \cdot \cdots \cdot \log_{15} 16$$

*çarpımının değerini bulunuz.*

*Çözüm.* Sonuç, ilk bakışta çok karmaşık görünüyor; oysa bir önceki sonuçtaki zincirleme eşitlik problemi tek adımda çözer. Ardışık çarpanları ikili gruplayalım:

$$\log_2 3 \cdot \log_3 4 = \log_2 4, \qquad \log_4 5 \cdot \log_5 6 = \log_4 6, \quad \ldots$$

Her grupta "ara taban" sadeleşip yoluna devam ediyor. Zinciri baştan sona uygularsak ara terimlerin tamamı birbirini götürür ve geriye yalnızca ilk taban ile son argüman kalır:

$$\log_2 3 \cdot \log_3 4 \cdots \log_{15} 16 = \log_2 16 = 4.$$

Gerçekten de bu çarpım bir teleskopik sadeleşmedir: her $\log_k(k+1)$ çarpanı, kendisinden önceki terimin argümanını bir sonraki tam sayıya taşır. □

Kapanış olarak, taban değiştirmenin karmaşıklık dilindeki karşılığını görelim. $\log_a n = \frac{\log_b n}{\log_b a}$ eşitliğinde $\frac{1}{\log_b a}$ çarpanı $n$'den bağımsız bir *sabittir*. Bu yüzden ikili aramada adım sayısını $\log_2 n$ ile, sıralamada karşılaştırma sayısını $\log_2 n$ ile ifade etmemiz ne kadar doğalsa, hepsini $O(\log n)$ biçiminde yazmamız da o kadar doğrudur: tabanı değiştirmek yalnızca sabit bir çarpan ekler ve büyük $O$ gösterimi sabit çarpanları görmezden gelir (Bölüm 21'de göreceğimiz gibi). Bu gözlem, ileride $O(\log n)$ yazan bir karmaşıklığı okurken "hangi taban?" diye duraksamamanız için yeterlidir.

# Bölüm 17

# Parite ve Simetri

Kombinatorik ve algoritma tasarımında en etkili çözümler çoğu zaman uzun hesaplamaların değil, problemin altında yatan dengenin fark edilmesiyle ortaya çıkar. Bir durum uzayı çok geniş olabilir; milyarlarca durum arasından belirli bir hedefe ulaşılıp ulaşılamayacağını adım adım aramak (örneğin kaba kuvvet ile tüm yolları denemek) algoritmik açıdan pratik değildir. İşte bu noktada güçlü bir matematiksel filtreye başvururuz: *Parite* ve *simetri*.

Parite, bir nesnenin veya durumun ikiye bölünme karakterini (teklik-çiftlik durumunu) inceler. Bir sürecin her adımında bu nitelik korunuyorsa (bir değişmez oluşturuyorsa), başlangıç durumu ile hedef durumun pariteleri karşılaştırılarak hedefe ulaşmanın imkânsızlığı hızla gösterilebilir. Simetri ise durum uzayını birbiriyle eşdeğer parçalara bölerek hem sayma problemlerini küçültür hem de algoritmik stratejilerde rakibin hamlelerini aynalayarak kesin kazanç yolları sunar.

Bu bölümde; algoritma sorularında ve yarışmalarda sıkça karşımıza çıkan imkânsızlık ispatlarını, satranç tahtası ve ızgara renklendirmelerini ve simetri yardımıyla sayma tekniklerini ele alacağız.

## 17.1 Parite ve İmkânsızlık

Bir problemin çözümünü aramak yerine, böyle bir çözümün *asla var olamayacağını* göstermek bilgisayar biliminde önemli bir kazanımdır. Arama uzayını gereksiz dallardan temizlemek (budama / pruning) ve ulaşılamaz durumları baştan tespit etmek algoritmanın çalışma zamanını ciddi biçimde düşürür. Bu imkânsızlık ispatlarının en temel aracı paritedir.

### 17.1.1 Teklik-Çiftlik Aritmetiği ve Değişmezler

Günlük hayattan bildiğimiz basit tek-çift kuralları, Bölüm 14'te ele aldığımız modüler aritmetiğin (mod2) temelidir.

**Tanım 17.1** (Parite)**.** *Bir $n \in \mathbb{Z}$ tam sayısı için; eğer $n = 2k$ $(k \in \mathbb{Z})$ biçiminde yazılabiliyorsa n'ye* ***çift****, $n = 2k + 1$ $(k \in \mathbb{Z})$ biçiminde yazılabiliyorsa n'ye* ***tek*** *sayı denir. Bir sayının çift veya tek olma durumuna o sayının* ***paritesi*** *adı verilir. $n$ çift ise $n \equiv 0 \pmod 2$, tek ise $n \equiv 1 \pmod 2$ yazılır.*

Temel işlemler altında paritenin davranışı şu kurallara dayanır:

$$\begin{aligned} \text{Çift} \pm \text{Çift} &= \text{Çift}, & \text{Tek} \pm \text{Tek} &= \text{Çift}, \\ \text{Çift} \pm \text{Tek} &= \text{Tek}, & \text{Tek} \pm \text{Çift} &= \text{Tek}, \\ \text{Çift} \times \text{Herhangi Bir Tam Sayı} &= \text{Çift}, & \text{Tek} \times \text{Tek} &= \text{Tek}. \end{aligned}$$

Bu tablodan çıkan temel sonuç şudur: **İki sayının toplamı ile farkının paritesi her zaman aynıdır.** Çünkü:

$$(a + b) - (a - b) = 2b$$

farkı her zaman çifttir. Dolayısıyla $a + b$ tek ise $a - b$ de tek, $a + b$ çift ise $a - b$ de çifttir.

**Tanım 17.2** (Değişmez (Invariant))**.** *Bir sistemde izin verilen hamleler veya durum geçişleri boyunca değeri hiç değişmeyen matematiksel bir niceliğe veya özelliğe* ***değişmez*** *(invariant) denir (bu tekniğin genelleştirilmiş halini Bölüm 18'de ayrıntılı inceleyeceğiz).*

Eğer başlangıç durumunun sahip olduğu değişmez bir özellik hedef durumda bulunmuyorsa, başlangıçtan hedefe izin verilen kurallarla ulaşmak imkânsızdır.

**Örnek 17.3.** *Bir tahtada $1, 2, 3, \ldots, 2025$ sayıları yazılıdır. Her adımda tahtadan rastgele iki sayı $a$ ve $b$ silinir ve yerlerine bu iki sayının farkı olan $|a - b|$ yazılır. Bu işlem tahtada tek bir sayı kalana kadar tekrarlanıyor. Son kalan sayı 0 olabilir mi?*

*Çözüm.* İşlem her adımda tahtadaki sayı adedini 1 azaltır. Toplamda 2024 adım sonra tahtada tek bir sayı kalacaktır. Tüm olası silme sıralamalarını tek tek denemek imkânsızdır (2025! civarında dallanma vardır). O halde her adımda korunan bir nicelik (değişmez) aramalıyız. Silinen iki sayının yerine farkları yazıldığına göre, tahtadaki sayıların *toplamının* nasıl değiştiğine odaklanalım.

Tahtadaki tüm sayıların toplamına $S$ diyelim. Bir adımda $a$ ve $b$ silinip yerine $|a - b|$ yazıldığında yeni toplam:

$$S' = S - a - b + |a - b|$$

olur. $a$ ve $b$'nin büyüklük sırasına bakılmaksızın, mutlak değer $|a - b|$ ifadesi ya $a - b$ ya da $b - a$'dır. Her iki durumda da:

$$S - S' = (a + b) - |a - b|$$

ifadesi daima çift bir sayıdır. Nitekim $a \geq b$ ise $(a+b)-(a-b) = 2b$ (çift), $a < b$ ise $(a+b)-(b-a) = 2a$ (çift) olur.

Bu durum, $S$ ile $S'$ toplamlarının paritelerinin aynı olduğunu gösterir:

$$S' \equiv S \pmod 2.$$

Yani her adımda tahtadaki sayıların toplamının teklik-çiftliği **değişmez** kalır.

Başlangıçtaki toplamı hesaplayalım:

$$S_0 = \sum_{k=1}^{2025} k = \frac{2025 \times 2026}{2} = 2025 \times 1013.$$

İki tek sayının çarpımı tek olduğundan, $S_0$ tektir ($S_0 \equiv 1 \pmod 2$).

Son durumda tahtada tek bir $x$ sayısı kaldığında, tahtadaki sayıların toplamı doğrudan bu $x$ sayısına eşit olur. Değişmez prensibi gereğince:

$$x \equiv S_0 \equiv 1 \pmod 2$$

olmalıdır. Dolayısıyla son kalan sayı mutlaka bir **tek sayı** olmak zorundadır. 0 bir çift sayı olduğundan ($0 \equiv 0 \pmod 2$), son sayının 0 olması kesinlikle **imkânsızdır**. □

### 17.1.2 El Sıkışma Lemması ve Çift Sayıda Tek Derece

Paritenin graf teorisindeki en temel uygulaması El Sıkışma Lemması'dır (Handshaking Lemma).

**Teorem 17.4** (El Sıkışma Lemması)**.** *Sonlu sayıda insanın bulunduğu bir odada, tek sayıda kişiyle tokalaşmış olan kişilerin sayısı daima* ***çifttir****.*

*Kanıt.* Odada $n$ kişi bulunsun ve $i$. kişinin yaptığı tokalaşma sayısı $d_i$ olsun. Her tokalaşma eylemi iki kişi arasında gerçekleşir. Dolayısıyla tüm kişilerin tokalaşma sayılarının toplamı, toplam tokalaşma sayısının ($T$) tam olarak iki katıdır (bu "el sıkışma lemması"nı Bölüm 10'da çifte sayım bağlamında graf modeliyle kurmuştuk; burada aynı gözlemi parite aracına dönüştürüyoruz):

$$\sum_{i=1}^{n} d_i = 2T.$$

Eşitliğin sağ tarafı $2T$ olduğundan, toplam daima bir çift sayıdır. Bu toplamı, tokalaşma sayısı tek olanlar ("Tek") ve çift olanlar ("Çift") olarak iki gruba ayıralım:

$$\sum_{i \in \text{Tek}} d_i + \sum_{j \in \text{Çift}} d_j = 2T.$$

$\sum_{j\in\text{Çift}} d_j$ toplamı çift sayıların toplamı olduğu için çifttir. Çift bir sayıdan çift bir sayı çıkarıldığında sonuç çift kalır:

$$\sum_{i\in\text{Tek}} d_i = 2T - \sum_{j\in\text{Çift}} d_j \equiv 0 \pmod 2.$$

Tek sayıların toplamının çift olabilmesi ancak ve ancak toplanan tek sayıların adedinin **çift** olmasıyla mümkündür. Dolayısıyla tek sayıda kişiyle tokalaşanların sayısı çifttir. □

**Örnek 17.5.** *Bir bilgi yarışmasında 15 yarışmacı vardır. Her yarışmacının diğer yarışmacılardan tam olarak 7 tanesiyle daha önce tanışmış olması mümkün müdür?*

*Çözüm.* Yarışmacıları birer nokta, tanışıklıkları ise bu noktalar arasındaki çizgiler olarak düşünelim. Eğer bu durum mümkün olsaydı, her birinin tanışıklık sayısı (derecesi) $d_i = 7$ olan 15 yarışmacı bulunurdu. Yarışmacıların tamamı tek sayıda (7) tanışıklığa sahiptir. Dolayısıyla tek sayıda tanışıklığı olan kişilerin sayısı 15'tir. Ancak 15 bir tek sayıdır; oysa El Sıkışma Lemması gereğince tek dereceli kişilerin sayısı çift olmak zorundadır. Dolayısıyla böyle bir durumun gerçekleşmesi **mümkün değildir**. □

### 17.1.3 Alan ve Yol Paritesi (Grid Parity)

İki boyutlu ızgaralarda (grid) hareket eden robotlar veya piyonlar, her adımda koordinatlarını değiştirir. Bir kafes graf üzerindeki yollar parite analiziyle doğrudan sınırlandırılabilir.

**Örnek 17.6.** *Bir robot $5 \times 5$'lik bir ızgaranın sol alt köşesinde $(1,1)$ noktasında durmaktadır. Robot her adımda yalnızca komşu bir kareye (yukarı, aşağı, sağa, sola) gidebilmektedir. Robotun ızgaradaki her kareyi **tam olarak bir kez** ziyaret edip başladığı $(1,1)$ karesine geri dönmesi mümkün müdür?*

*Çözüm.* Bir kareyi tam bir kez ziyaret edip başa dönmek bir Hamilton çevrimi arayışıdır. İncelemeyi adım sayısına ve koordinatların toplamına dayandıralım.

Robotun bulunduğu kareyi $(x, y)$ koordinatlarıyla gösterelim ($1 \le x, y \le 5$). Robot her adımda ya $x$'i $\pm 1$ değiştirir ya da $y$'yi $\pm 1$ değiştirir. Her iki durumda da koordinatların toplamı olan $x + y$ değerinin paritesi değişir:

$$(x \pm 1) + y \equiv (x + y) + 1 \pmod 2, \quad x + (y \pm 1) \equiv (x + y) + 1 \pmod 2.$$

Başlangıç noktasında $x + y = 1 + 1 = 2$ (çift) değerindedir.

- 1. adımda: Tek bir kareye gider.
- 2. adımda: Çift bir kareye gider.

- Genel olarak, $k$ adım sonra robotun bulunduğu karenin koordinat toplamı $k$ ile aynı paritededir:

$$x_k + y_k \equiv x_0 + y_0 + k \equiv k \pmod 2.$$

Robotun $5 \times 5 = 25$ farklı karenin her birini tam olarak bir kez ziyaret etmesi için 24 adım atması, ardından başlangıç karesine geri dönmesi için 1 adım daha atarak toplam 25 adım atması gerekir.

Fakat 25 adım sonra robotun ulaştığı karenin koordinat toplamının paritesi:

$$x_{25} + y_{25} \equiv 25 \equiv 1 \pmod 2 \quad \text{(Tek)}$$

olmalıdır. Oysa başlangıç karesi $(1, 1)$ için koordinat toplamı $1 + 1 = 2$ (Çift) idi. Tek pariteye sahip bir noktada bulunan robotun, koordinat toplamı çift olan başlangıç karesinde olması imkânsızdır. Dolayısıyla böyle bir kapalı tur **yapılamaz**. □

Aşağıdaki algoritma parçası, tahtada rastgele yapılan fark hamlelerinin toplam paritesini nasıl koruduğunu doğrulamaktadır:

```
FUNCTION checkParity(numbers)
    list = COPY(numbers)
    initialParity = SUM(list) MOD 2

    WHILE LENGTH(list) > 1 DO
        indexA = RANDOM_INTEGER(0, LENGTH(list) - 1)
        valueA = REMOVE_AT(list, indexA)

        indexB = RANDOM_INTEGER(0, LENGTH(list) - 1)
        valueB = REMOVE_AT(list, indexB)

        difference = ABSOLUTE(valueA - valueB)
        APPEND difference TO list

        ASSERT (SUM(list) MOD 2) EQUALS initialParity
    END WHILE

    RETURN list[0]
END FUNCTION

sampleList = SEQUENCE(1 TO 5)
finalNumber = checkParity(sampleList)

IF finalNumber MOD 2 EQUALS 1 THEN
    parityString = "Odd"
ELSE
    parityString = "Even"
END IF
```

```
OUTPUT "Remaining number: ", finalNumber, ", Parity: ", parityString
```

**En Sık Yapılan Hata:** Bir problemin parite argümanıyla çözülebilmesi için durum geçişlerinin tamamında paritenin korunması gerekir. Sadece bazı hamlelerde korunan bir özellik değişmez (*invariant*) değildir. Ayrıca, bir hedefin parite testini geçmesi (örneğin hedefin de tek olması), o hedefe kesinlikle *ulaşılabileceğini* kanıtlamaz; parite yalnızca imkânsızlığı kanıtlamak için tek yönlü bir engeldir (gerek koşuldur, yeter koşul değildir).

## 17.2 Renklendirme Argümanları

Izgara problemleri iki boyutlu yapıda olduğundan, tek bir parite değişkeni her zaman yeterli bilgiyi taşımaz. Bir satranç tahtasını siyah ve beyaz karelere boyamak, aslında iki boyutlu bir parite fonksiyonu tanımlamaktır:

$$\text{Renk}(x, y) = (x + y) \bmod 2.$$

Bu iki renkli yapı yetersiz kaldığında, tahtayı 3, 4 veya daha fazla renkle periyodik olarak boyayarak daha kapsamlı değişmezler elde edebiliriz.

### 17.2.1 Kesik Satranç Tahtası ve Domino Kaplamaları

Klasik renklendirme yaklaşımının bilinen örneği, dominolarla ($1 \times 2$ boyutundaki taşlarla) alan kaplama problemidir.

**Tanım 17.7** (Domino ve Tromino)**.** *Bir $1 \times 2$'lik dikdörtgene **domino**; $1 \times 3$'lük dikdörtgene **düz tromino**; 3 birim kareden oluşan "L" şeklindeki parçaya ise **L-tromino** denir. Bir alanın bu taşlarla örtülmesine, taşların üst üste binmemesi ve tahtanın dışına taşmaması şartıyla **kaplama** (tiling) adı verilir.*

**Örnek 17.8.** *Standart bir $8 \times 8$ satranç tahtasından, karşılıklı iki çapraz köşe karesi kesilip atılıyor. Kalan 62 kareyi, her biri $1 \times 2$ boyutunda olan 31 adet domino taşı ile tamamen örtmek mümkün müdür?*

*Çözüm.* Kalan alan 62 birim karedir ve 31 domino tam olarak $31 \times 2 = 62$ birim karelik yer tutar. Alan hesabı açısından bir engel görünmemektedir. Ancak tahtanın geometrik yapısı parite dengesini etkiler.

Standart satranç tahtası renklendirmesini düşünelim:

1. $8 \times 8$'lik tahtada 32 adet siyah, 32 adet beyaz kare vardır.
2. Tahtaya yerleştirilen her $1 \times 2$'lik domino, tahtanın neresine ve hangi yönde (yatay ya da dikey) konulursa konsun, mutlaka **bir siyah ve bir beyaz** kareyi örter.

3. Dolayısıyla 31 adet domino yerleştirildiğinde, örtülen toplam kareler tam olarak 31 siyah ve 31 beyaz kare olmak zorundadır.

Şimdi kesilen köşeleri inceleyelim: Satranç tahtasında zıt çapraz köşeler aynı renge sahiptir. Örneğin sol alt köşe $(1,1)$ beyaz ise $(1+1=2)$, sağ üst köşe $(8,8)$ de beyazdır $(8+8=16)$. Bu iki beyaz köşe tahtadan çıkarıldığında geriye:

$$32 \text{ siyah ve } 30 \text{ beyaz kare}$$

kalır. Oysa 31 dominonun kaplayabilmesi için eşit sayıda (31'e 31) siyah ve beyaz kareye ihtiyaç vardır. Siyah ve beyaz kare sayıları eşit olmadığından, bu 62 kareyi dominolarla kaplamak **kesinlikle imkânsızdır**. □

### 17.2.2 Daha Fazla Renk: Çok Renkli Argümanlar

Her parça iki kare kaplamaz. Örneğin $1 \times 3$'lük düz trominolar üç kare kaplar. Standart 2-renkli dama tahtası boyaması bu taşlar için yetersiz kalır; çünkü bir domino gibi her tromino her renkten eşit sayıda almak zorunda değildir. Bu durumda renk sayısını artırmak gerekir.

**Örnek 17.9.** $10 \times 10$*'luk bir tahta,* $1 \times 4$*'lük tetrominolar (çubuklar) kullanılarak tamamen kaplanabilir mi?*

*Çözüm.* Tahtadaki toplam kare sayısı $10 \times 10 = 100$'dür. Her $1 \times 4$ tetromino 4 kare kaplar ve $100/4 = 25$ parça kullanılmalıdır; alan tam bölünmektedir. Standart iki renkle boyama yapsak, her $1 \times 4$ parça 2 siyah ve 2 beyaz örter, tahtada da 50 siyah 50 beyaz bulunur; buradan bir çelişki doğmaz. O halde parçanın uzunluğu olan 4'e uygun bir **4-renkli boyama** kuralım.

Kareleri $(i, j)$ koordinatlarıyla gösterelim $(0 \le i, j \le 9)$. Bir $(i, j)$ karesini şu kurala göre $\{0, 1, 2, 3\}$ renklerinden birine boyayalım:

$$\text{Renk}(i, j) = (i + j) \bmod 4.$$

Bu boyamada herhangi bir satırda veya sütunda yan yana gelen 4 ardışık kare, $0, 1, 2, 3$ mod değerlerinin dördünü de tam birer kez içerir. Dolayısıyla, tahtaya ister yatay ister dikey yerleştirilsin, **her** $1 \times 4$ **tetromino her bir renkten (0, 1, 2 ve 3) tam olarak birer kare örter.**

Eğer tahta 25 adet tetromino ile kaplanabiliyorsa, tahtada bulunan $0, 1, 2, 3$ renkli karelerin sayılarının **birbirine eşit ve 25'er adet** olması gerekir.

Şimdi tahtadaki renk dağılımını hesaplayalım. Tahtayı $4 \times 4$'lük kare bloklara bölelim:

- Bir $4 \times 4$'lük blokta her renkten tam olarak 4 adet bulunur.

- Tahta $10 \times 10$ boyutunda olduğundan, bunu dört adet $4 \times 4$'lük blok, iki adet $4 \times 2$'lik blok, iki adet $2 \times 4$'lük blok ve bir adet $2 \times 2$'lik köşe bloğu olarak parçalayabiliriz:
$$10 = 4 + 4 + 2.$$

Genişliği veya yüksekliği 4'ün katı olan her bölgede renkler tamamen dengelidir (her renkten eşit sayıda vardır). Renk dengesini bozabilecek tek bölge, sağ altta kalan $2 \times 2$'lik artık bölgedir.

Bu $2 \times 2$'lik bloğun koordinatları $i, j \in \{8, 9\}$ olsun:

$$\begin{aligned}
(8,8) &\implies 8+8 = 16 \equiv 0 \pmod 4,\\
(8,9) &\implies 8+9 = 17 \equiv 1 \pmod 4,\\
(9,8) &\implies 9+8 = 17 \equiv 1 \pmod 4,\\
(9,9) &\implies 9+9 = 18 \equiv 2 \pmod 4.
\end{aligned}$$

Görüldüğü gibi bu artık bölgede:

- 0 renginden 1 adet,
- 1 renginden 2 adet,
- 2 renginden 1 adet,
- 3 renginden **0 adet** vardır.

Renklerin 4'ün katı olan kısımlardaki dağılımı eşitti. Bu artık bölge yüzünden tahtanın tamamında:

- 1 rengine sahip karelerin sayısı, 3 rengine sahip karelerin sayısından 2 fazladır.
- Dolayısıyla 3 rengine sahip karelerin sayısı 25 olamaz (en fazla 24 olabilir).

Her tetromino her renkten eşit sayıda örtmek zorunda olduğuna göre, bu tahtayı $1 \times 4$'lük tetrominolarla kaplamak **olanaksızdır**. □

### 17.2.3 At Gezintisi ve Renk Değişimi

Satrançta atın (knight) hareketi, renklendirme ile çözülen tipik bir kural örneğidir.

**Teorem 17.10** (Atın Renk Değişimi)**.** *Bir satranç tahtasında standart kurallarla hareket eden bir at, her hamlesinde bulunduğu karenin rengini* ***kesinlikle değiştirir****. Siyah kareden beyaza, beyaz kareden siyaha geçer.*

*Kanıt.* Atın hareketi bir yönde 1 birim, dik yönde 2 birim ilerlemektir. Yani koordinat değişimi $(\Delta x, \Delta y) \in \{(\pm 1, \pm 2), (\pm 2, \pm 1)\}$ kümesindedir. Yeni karenin koordinat toplamı:

$$(x + \Delta x) + (y + \Delta y) = (x + y) + (\Delta x + \Delta y).$$

Her olası hamle için $\Delta x + \Delta y$ değerleri şunlardır:

$$\pm 1 \pm 2 \in \{-3, -1, 1, 3\}.$$

Bu değerlerin tamamı **tek** sayıdır. Dolayısıyla:

$$(x + \Delta x) + (y + \Delta y) \equiv (x + y) + 1 \pmod 2.$$

Karenin paritesi her hamlede tersine döner. Yani at daima zıt renkli bir kareye basmak zorundadır. □

**Örnek 17.11.** $5 \times 5$ *'lik bir satranç tahtasında bir at,* $(1, 1)$ *karesinden başlayıp her kareyi tam olarak bir kez ziyaret ederek yeniden* $(1, 1)$ *karesine dönebilir mi?*

*Çözüm.* Tahtayı standart satranç düzeninde boyayalım. At her hamlede farklı renkte bir kareye geçmek zorundadır; dolayısıyla ziyaret ettiği karelerin renkleri sırayla değişir.

Kapalı bir turda at başladığı kareye geri döner; yani tur başladığı renkte biter. Renkler sırayla değiştiğine göre, kapalı bir turda ziyaret edilen siyah ve beyaz kare sayıları birbirine eşit olmak zorundadır.

Oysa $5 \times 5 = 25$ kareli tahtada

$$\lceil 25/2 \rceil = 13 \text{ siyah}, \qquad \lfloor 25/2 \rfloor = 12 \text{ beyaz}$$

kare bulunur. Sayılar eşit olmadığından bu tahtada kapalı bir at turu yoktur. □

## 17.3 Simetri ile Sayma

Büyük kombinatorik uzaylarda her durumu tek tek listelemek yerine, uzayı birbirinin ayna görüntüsü olan eş parçalara ayırmak sayımı önemli ölçüde kolaylaştırır. Simetri, temelde bir **eşleme** (bijection / involution) fikridir.

### 17.3.1 Eşleme ve Yansıma Prensibi

Bir $S$ kümesinde öyle bir $f : S \to S$ dönüşümü tanımlayalım ki her $x \in S$ için $f(f(x)) = x$ olsun ($f$ bir involüsyondur) ve hiçbir $x$ için $f(x) = x$ olmasın (sabit nokta bulunmasın). Bu durumda $S$ kümesinin tüm elemanları $\{x, f(x)\}$ ikilileri halinde eşleşir. Buradan doğrudan şu sonucu çıkarırız:

$$|S| \text{ çift sayıdır ve } |S| = 2 \times (\text{çiftlerin sayısı}).$$

**Teorem 17.12** (Alt Kümelerin Parite Dengesi)**.** *$n \geq 1$ elemanlı sonlu bir kümenin tek sayıda elemana sahip alt kümelerinin sayısı ile çift sayıda elemana sahip alt kümelerinin sayısı birbirine eşittir ve her ikisi de $2^{n-1}$'dir:*

$$\sum_{k \text{ tek}} \binom{n}{k} = \sum_{k \text{ çift}} \binom{n}{k} = 2^{n-1}.$$

*Kanıt.* Cebirsel olarak bu eşitlik Bölüm 6'da gördüğümüz $(1-1)^n = 0$ binom açılımından çıkar. Simetri ve eşleme (bijection) yaklaşımı ise duruma doğrudan yapısal bir açıklama getirir.

Kümemiz $A = \{a_1, a_2, \ldots, a_n\}$ olsun. Özel bir eleman seçelim, örneğin $a_1$. $A$'nın herhangi bir $X$ alt kümesi için şu eşleme kuralını tanımlayalım:

$$f(X) = \begin{cases} X \setminus \{a_1\}, & \text{eğer } a_1 \in X \text{ ise,} \\ X \cup \{a_1\}, & \text{eğer } a_1 \notin X \text{ ise.} \end{cases}$$

Bu $f$ fonksiyonunun özellikleri şunlardır:

1. $f(f(X)) = X$'tir (kendi kendisinin tersidir).
2. $X$ kümesine $a_1$ eklenir ya da çıkarılır; dolayısıyla eleman sayısı $|f(X)| = |X| \pm 1$ olur.
3. Bir tek sayının 1 fazlası veya 1 eksiği çift; bir çift sayının 1 fazlası veya 1 eksiği tektir.

Dolayısıyla $f$, tek elemanlı her alt kümeyi benzersiz bir çift elemanlı alt kümeyle eşleştirir ve hiçbir eleman açıkta kalmaz.

Bu birebir eşleme, tek elemanlı alt kümelerin kümesi ile çift elemanlı alt kümelerin kümesinin eleman sayılarının tam olarak eşit olduğunu gösterir. Toplam $2^n$ alt küme olduğuna göre, her birinin sayısı $2^n/2 = 2^{n-1}$'dir. □

### 17.3.2 Palindrom Sayımı

Simetrinin dizgiler (stringler) üzerindeki en doğrudan uygulaması palindromlardır.

**Tanım 17.13** (Palindrom)**.** *Soldan sağa ve sağdan sola okunuşu aynı olan dizgilere veya sayılara **palindrom** denir. Bir $s = s_1 s_2 \ldots s_n$ dizgisinin palindrom olması, her $1 \leq i \leq n$ için:*

$$s_i = s_{n-i+1}$$

*olması demektir.*

Bir palindromun ilk yarısı belirlendiğinde, ikinci yarısı ayna simetrisi gereği **tek bir şekilde** belirlenir. Bu durum, serbestçe seçilebilecek bağımsız değişken sayısını (serbestlik derecesini) yarıya indirir.

**Teorem 17.14** (Palindrom Sayısı)**.** $\Sigma$ *alfabesinin boyutu* $|\Sigma| = k$ *olsun.* $n$ *uzunluğundaki palindrom dizgilerin sayısı:*

$$k^{\lceil n/2 \rceil}$$

*formülüyle verilir.*

*Kanıt.* Dizgimiz $s_1 s_2 \dots s_n$ olsun.

- $n$ çift ise ($n = 2m$): İlk $m$ karakter $(s_1, s_2, \dots, s_m)$ alfabedeki $k$ karakterden herhangi biri olarak serbestçe seçilebilir ($k^m$ yolla). Geriye kalan karakterler simetri kuralı gereğince $s_{m+1} = s_m, \dots, s_{2m} = s_1$ olarak zorunlu belirlenir. Toplam seçim sayısı $k^m = k^{n/2}$'dir.
- $n$ tek ise ($n = 2m + 1$): Ortadaki karakter $s_{m+1}$ de dahil olmak üzere ilk $m+1$ karakter serbestçe seçilebilir ($k^{m+1}$ yolla). Sonraki $m$ karakter simetrik eşleriyle eşleşir. Toplam seçim sayısı $k^{m+1} = k^{\lceil n/2 \rceil}$'dir.

Her iki durum birleştirildiğinde sonuç $k^{\lceil n/2 \rceil}$ olur. □

**Örnek 17.15.** $\{0, 1\}$ *ikili alfabesi üzerinde tanımlı 10 uzunluğundaki tüm bit dizgilerinden rastgele biri seçiliyor. Seçilen dizginin palindrom* ***olmama*** *olasılığı kaçtır?*

*Çözüm.* Palindrom olmayan dizgileri doğrudan saymak yerine, simetrinin korunduğu durumu (tümleyen olayı) sayıp tüm durumlardan çıkarmak daha pratiktir.

1. Toplam ikili dizgi sayısı: $2^{10} = 1024$.
2. Palindrom olan ikili dizgi sayısı: $n = 10$ çift sayı olduğundan serbestlik derecesi $10/2 = 5$'tir. Dolayısıyla ilk 5 bit serbestçe seçilir, son 5 bit bunların yansıması olur:
$$\text{Palindrom Sayısı} = 2^5 = 32.$$
3. Palindrom olmayan dizgilerin sayısı:
$$1024 - 32 = 992.$$
4. İstenen olasılık:
$$P = \frac{992}{1024} = \frac{31}{32} \quad (\approx \%96{,}875).$$

□

**Örnek 17.16.** $6 \times 6$ *'lık bir ızgaranın kareleri siyah ve beyaza boyanacaktır. Izgaranın merkezine göre* $180^\circ$ *döndürüldüğünde görüntüsünün değişmemesi (merkezi simetriye sahip olması) şartıyla kaç farklı boyama yapılabilir?*

*Çözüm.* 180° dönme simetrisi, her $(x, y)$ karesinin tahtanın merkezine göre simetriği olan kareyle aynı renge sahip olmasını gerektirir. Bu durum kareleri ikili denklik sınıflarına (yörüngelere) ayırır.

Satır ve sütun indislerini $1 \le i, j \le 6$ olarak alalım. $(i, j)$ karesinin tahta merkezine göre 180° döndürülmüş hali $(7-i, 7-j)$ karesidir. $6 \times 6$ tahtada $i = 7-i$ olamaz çünkü 7 tek sayıdır (merkezde tek başına kalan sabit bir kare yoktur).

Bu durumda tahtadaki 36 kare, birbirinin 180° dönmüş hali olan çiftler halinde gruplanır:

$$\{(i, j), (7-i, 7-j)\}.$$

Toplam çift sayısı:

$$\frac{36}{2} = 18.$$

Her bir çiftin rengini belirlemek bağımsız bir seçimdir ve 2 seçenek vardır (ikisi birden siyah veya ikisi birden beyaz). Bir kare boyandığında simetriği de aynı renge boyanmak zorundadır.

Dolayısıyla simetrik boyama sayısı:

$$2^{18} = 262\,144$$

farklı şekilde gerçekleştirilebilir. □

Aşağıdaki C kodu, verilen bir dizginin palindrom olup olmadığını simetri ekseninden dışa doğru Bölüm 22'de inceleyeceğimiz two pointers tekniğiyle kontrol eder:

```c
#include <stdio.h>
#include <stdbool.h>
#include <string.h>

// Iki isaretci ile simetri kontrolu: O(n) zaman, O(1) ekstra alan
bool palindrom_mu(const char *str, int n) {
    int sol = 0;
    int sag = n - 1;

    while (sol < sag) {
        if (str[sol] != str[sag]) {
            return false; // Simetri bozuldu
        }
        sol++;
        sag--;
    }
    return true; // Tum ayna ciftleri eslesti
}

int main() {
    char s1[] = "kayak";
    char s2[] = "tubitak";
```

```
    printf("%s: %s\n", s1, palindrom_mu(s1, strlen(s1)) ? "Palindrom" : "
    Degil");
    printf("%s: %s\n", s2, palindrom_mu(s2, strlen(s2)) ? "Palindrom" : "
    Degil");

    return 0;
}
```

### 17.3.3 Oyunlarda Simetri Stratejisi

Kombinatoryal oyun teorisinde simetri, genellikle ikinci oyuncu için belirleyici bir kazanma (veya kaybetmeme) stratejisi sunar. Tahtada merkezi veya eksensel bir simetri tanımlanabiliyorsa, rakibin her hamlesi simetrik noktaya yansıtılarak cevaplanabilir.

**Örnek 17.17** (Simetri Stratejisi)**.** *Yuvarlak bir masanın üzerine iki oyuncu sırayla eşit büyüklükte madeni 1 TL'lik paralar koymaktadır. Paralar üst üste gelemez ve masanın dışına taşamaz. Masaya geçerli bir hamleyle son parayı koyan oyuncu oyunu kazanır (hamle yapamayan kaybeder). Oyunu doğru stratejiyle oynayan birinci oyuncunun daima kazanabileceğini kanıtlayınız.*

*Çözüm.* **Strateji:** Masa dairesel olduğundan **merkezi simetriye** sahiptir. Birinci oyuncu şu adımları izler:

1. **İlk Hamle:** Birinci oyuncu ilk parasını tam olarak dairesel masanın **merkezine** yerleştirir.

2. **Sonraki Hamleler:** İkinci oyuncu masanın neresine bir para koyarsa koysun, birinci oyuncu bu paranın masanın merkezine göre tam simetriği olan noktaya kendi parasını koyar.

**Neden çalışır?** Masa merkeze göre simetrik olduğundan, ikinci oyuncunun boş bulup para koyduğu bir yerin simetriği de **kesinlikle boştur ve masanın üzerindedir**. Eğer bu simetrik nokta dolu olsaydı, simetri kuralı gereği ikinci oyuncunun oynadığı noktanın da daha önceden dolu olması gerekirdi; bu ise bir çelişkidir.

Böylece ikinci oyuncu ne zaman geçerli bir hamle yapabilse, birinci oyuncunun daima ona karşılık gelen geçerli bir hamlesi hazır bulunur. Paraların kapladığı toplam alan sonlu olduğundan oyun belirli bir adım sonra biter. Son hamleyi mutlaka birinci oyuncu yapacaktır ve oyunu **birinci oyuncu kazanır**. □

**En Sık Yapılan Hata:** Simetri stratejilerinde simetri merkezinin hamle yapılabilir bir nokta olup olmadığına dikkat edilmelidir. Örneğin simetri merkezi boş kalamıyorsa ya da oyun tahtasının boyutları çift ise (merkez bir kare değil de karelerin kesişim noktasıysa), birinci oyuncu merkeze taş koyamaz. Bu durumda simetri avantajı genellikle *ikinci oyuncuya* geçer.

## 17.4 Bölüm Özeti ve İpuçları

Bu bölümde, kombinatorik problemleri ve algoritmik durum uzaylarını basitleştiren iki temel aracı inceledik:

- **Parite:** Bir sayının 2'ye bölümünden kalandır. Toplam ve farkın paritesi aynıdır ($a+b \equiv a-b \pmod 2$). Süreç boyunca korunan bir parite (değişmez / invariant), ulaşılamaz durumları kesin bir dille ispatlar.
- **El Sıkışma Lemması:** Bir grafta tek dereceli düğümlerin sayısı daima çifttir. Tanışıklık, kenar ve derece sorularında ilk kontrol edilmesi gereken parite kuralıdır.
- **Renklendirme:** İki boyutlu ızgaralarda dominolar için 2 renk (dama tahtası), $1 \times k$ boyutundaki parçalar için $k$ renk periyodik boyama kullanılır. Alan yeterli olsa dahi, renklerin sayısal dengesizliği kaplamanın imkânsız olduğunu gösterir.
- **Simetri ve Eşleme:** Durum uzayını ayna çiftlerine ayırmak ($x \leftrightarrow f(x)$), eleman sayısının çift olduğunu veya iki kümenin boyutunun eşit olduğunu gösterir.
- **Palindromlar:** Simetri ekseni serbestlik derecesini yarıya indirir. $n$ uzunluğundaki bir dizgide bağımsız seçim sayısı $\lceil n/2 \rceil$'dir.
- **Simetrik Oyunlar:** Tahtada simetriyi elinde tutan oyuncu, rakibinin hamlelerini aynalayarak her zaman geçerli bir hamle bulmayı garanti eder.

# Bölüm 18

# İnvaryant ve Monovaryant

Olimpiyat matematiğinde ve algoritmik düşüncede sıkça karşılaşılan problemlerin önemli bir kısmı dinamik süreçlerle ilgilidir: Tahtaya sayılar yazılır, belirli kurallara göre silinip yerlerine yenileri konur; bir adada yaşayan farklı renkteki bukalemunlar karşılaşıp renk değiştirir; ya da bir grafın düğümleri arasındaki bağlantılar adım adım güncellenir. Bu tür süreçlerde karşımıza genellikle iki temel soru çıkar:

1. Belirli bir başlangıç durumundan arzu edilen bir hedef duruma ulaşmak mümkün müdür?

2. Bu süreç sonsuza kadar devam edebilir mi, yoksa sonlu sayıda adım sonra durmak zorunda mıdır?

Tüm olası hamle kombinasyonlarını denemek (kaba kuvvet / brute-force), durum uzayının hızla dallanıp büyümesi sebebiyle olanaksızdır. Bu noktada iki temel analitik araç devreye girer: Süreç boyunca hiçbir zaman değişmeyen bir büyüklüğü takip eden **değişmez (invaryant)** ve her adımda tek bir yönde (sürekli artan ya da sürekli azalan) değişen bir büyüklüğü izleyen **yarı-değişmez (monovaryant)**.

## 18.1 İnvaryant

Bir sürecin karmaşıklığı içinde kaybolmamak için, her hamlede korunan bir denge ararız. Bu yöntem, özellikle "$A$ durumundan $B$ durumuna ulaşılamayacağını" (imkânsızlığı) göstermek için en uygun araçtır.

### 18.1.1 Motivasyon ve Sezgi: Tahtadaki Sayılar

Fikri somut bir örnekle ele alalım:

> Tahtada $1, 2, 3, \ldots, 10$ sayıları yazılıdır. Her adımda herhangi iki $a$ ve $b$ sayısını silip yerlerine $|a - b|$ farkını yazıyoruz. Tahtada tek bir sayı kalana kadar bu işleme devam ediliyor. Son kalan sayının 0 olması mümkün müdür?

Birkaç rastgele deneme yapalım:

- İlk adımda 1 ve 2 silinip yerine $|1 - 2| = 1$ yazılabilir. Sayılarımız: $1, 3, 4, 5, 6, 7, 8, 9, 10$ olur.
- Başka bir tercihle 9 ve 10 silinip $|9 - 10| = 1$ yazılabilir.

Olası eşlemelerin sayısı çok fazladır. Fakat hamlenin sayıların toplamı üzerindeki etkisine odaklanalım. Tahtadaki tüm sayıların toplamını $S$ ile gösterelim. İki sayı seçip ($a$ ve $b$), bunları silip $|a - b|$ eklediğimizde yeni toplam $S'$ ne olur?

$$S' = S - a - b + |a - b|$$

Burada mutlak değerden kurtulmak için genelliği bozmadan $a \geq b$ kabul edebiliriz. O halde $|a - b| = a - b$ olur:

$$S' = S - a - b + (a - b) = S - 2b$$

Bu sonuç belirleyicidir: Yeni toplam, eski toplamdan çift bir sayı ($2b$) çıkarılarak elde edilmiştir. Başka bir deyişle:

$$S' \equiv S \pmod 2$$

Hamle ne olursa olsun, tahtadaki sayıların toplamının tekliği ya da çiftliği (Bölüm 17'de incelediğimiz parite özelliği) asla değişmez.

Başlangıçtaki toplamı hesaplayalım:

$$S_0 = 1 + 2 + 3 + \cdots + 10 = \frac{10 \times 11}{2} = 55$$

Görüldüğü üzere $S_0 = 55$ tek bir sayıdır ($55 \equiv 1 \pmod 2$). Sürecin her adımında parite korunduğundan, tahtada kalan son sayının da tek olması zorunludur. Bizden son sayının 0 olup olamayacağı sorulmuştu. 0 çift bir sayı olduğundan ($0 \equiv 0 \pmod 2$), bu hedefe ulaşmak **imkânsızdır**.

### 18.1.2 Tanım ve Temel Teorem

**Tanım 18.1** (İnvaryant / Değişmez)**.** *Bir dinamik sistemde, izin verilen herhangi bir durum geçişi (hamle) altında değerini koruyan matematiksel bir büyüklüğe ya da özelliğe* ***değişmez (invaryant)*** *denir.*

*Daha biçimsel olarak; durumlar kümesi $\mathcal{S}$, izin verilen hamle bağıntısı $T \subseteq \mathcal{S} \times \mathcal{S}$ olmak üzere, her $(s, s') \in T$ için $f(s) = f(s')$ koşulunu sağlayan bir $f : \mathcal{S} \to X$ fonksiyonuna sistemin bir invaryantı denir.*

**Teorem 18.2** (İmkânsızlık Prensibi)**.** *Bir sistemin başlangıç durumu $s_0$, ulaşılmak istenen hedef durumu $s_{hedef}$ ve sistemin bir invaryantı $f$ olsun. Eğer*

$$f(s_0) \neq f(s_{hedef})$$

*ise, izin verilen kurallarla $s_0$ durumundan $s_{hedef}$ durumuna ulaşmak imkânsızdır.*

*Kanıt.* Tümevarım ile açıktır. Başlangıçta $f(s_0) = c$ olsun. Yapılan her $k$. hamle sonunda ulaşılan durum $s_k$ ise, invaryant tanımı gereği $f(s_k) = f(s_{k-1}) = \ldots = f(s_0) = c$ olur. Dolayısıyla sistemin ulaşabileceği tüm durumların $f$ altındaki görüntüsü $c$ olmak zorundadır. $f(s_{\text{hedef}}) \neq c$ olduğundan, hedef duruma hiçbir sonlu adımda ulaşılamaz. □

### 18.1.3 Bukalemun Problemi

Olimpiyatlarda sıkça anılan klasik invaryant sorularından biri "üç renk" ya da bilinen adıyla "bukalemun" problemidir.

**Örnek 18.3** (Bukalemun Problemi (Kvant Dergisi, 1985))**.** *Bir adada 13 sarı, 15 yeşil ve 17 kırmızı bukalemun yaşamaktadır. Farklı renkteki iki bukalemun karşılaştığında, her ikisi de renklerini üçüncü renge dönüştürmektedir (örneğin bir sarı ile bir yeşil karşılaşırsa ikisi de kırmızı olur). Aynı renkteki iki bukalemun karşılaştığında ise renk değişimi olmamaktadır. Zaman içinde tüm bukalemunların aynı renge dönüşmesi mümkün müdür?*

*Çözüm.* Sarı, yeşil ve kırmızı bukalemunların anlık sayılarını sırasıyla $s$, $y$, $k$ ile gösterelim. Başlangıçta $(s, y, k) = (13, 15, 17)$'dir. Toplam bukalemun sayısı $13 + 15 + 17 = 45$'tir.

Hedeflenen durumlar şunlardır:

- Hepsi sarı: $(45, 0, 0)$
- Hepsi yeşil: $(0, 45, 0)$
- Hepsi kırmızı: $(0, 0, 45)$

Olası bir hamleyi inceleyelim. Bir sarı ve bir yeşil bukalemun karşılaştığında sayılar şu şekilde değişir:

$$s' = s - 1, \quad y' = y - 1, \quad k' = k + 2$$

Gruplar arasındaki farklara bakalım:

$$\begin{aligned} s' - y' &= (s-1) - (y-1) = s - y \\ s' - k' &= (s-1) - (k+2) = s - k - 3 \\ y' - k' &= (y-1) - (k+2) = y - k - 3 \end{aligned}$$

Farkların doğrudan korunduğu söylenemez; ancak farklar ya hiç değişmemekte $(s - y)$ ya da tam olarak 3 azalmaktadır ($s - k$ ve $y - k$). Bu durum Bölüm 14'te ele aldığımız modüler aritmetik yaklaşımına işaret eder. Herhangi iki renk grubunun sayısı arasındaki fark modulo 3'te sabit kalır:

$$s' - y' \equiv s - y \pmod 3, \quad s' - k' \equiv s - k \pmod 3, \quad y' - k' \equiv y - k \pmod 3$$

Başlangıç durumundaki farkları hesaplayalım:

$$\begin{aligned} s - y &= 13 - 15 = -2 \equiv 1 \pmod 3 \\ y - k &= 15 - 17 = -2 \equiv 1 \pmod 3 \\ k - s &= 17 - 13 = 4 \equiv 1 \pmod 3 \end{aligned}$$

Görüldüğü gibi başlangıçta hiçbir iki grubun farkı $0 \pmod 3$ değildir.

Oysa hedeflenen durumlarda (örneğin hepsi sarı olduğunda $(45, 0, 0)$):

$$y - k = 0 - 0 = 0 \equiv 0 \pmod 3$$

olmalıdır. Benzer şekilde diğer iki hedef durumda da sıfıra eşit olan iki grup bulunacağından, aralarındaki fark $0 \equiv 0 \pmod 3$ olmak zorundadır.

Sistemin invaryantı gereği, hiçbir adımda farkların modulo 3'teki değeri 1'den 0'a dönüşemez. Dolayısıyla tüm bukalemunların tek bir renkte toplanması imkânsızdır. □

### 18.1.4 Çözümlü Örnekler

**Örnek 18.4** (Kesilmiş Satranç Tahtası (Max Black, 1946))**.** $8 \times 8$ *boyutundaki standart bir satranç tahtasının karşılıklı iki çapraz köşesi kesilip atılıyor. Kalan* 62 *kareyi, her biri* $1 \times 2$ *boyutunda olan domino taşlarıyla eksiksiz örtmek neden imkânsızdır?*

*Çözüm.* Bu problemi Bölüm 17'de renklendirme yöntemiyle çözmüştük; aynı sonuca bir *değişmez* kurarak da ulaşabiliriz. Tahta standart olarak boyandığında 32 beyaz ve 32 siyah kare bulunur. Karşılıklı iki çapraz köşe aynı renkte olduğundan kesilmiş tahtada fark 2'dir.

Her domino tam olarak bir siyah ve bir beyaz kare örter. Dolayısıyla kaç domino kullanılırsa kullanılsın örtülen siyah ve beyaz kare sayıları eşit kalır; bu, sürecin bir **değişmezidir**. Örtmenin tamamlanabilmesi için farkın 0 olması gerekirdi; oysa başlangıçtaki fark 2'dir. Demek ki böyle bir örtme yoktur. □

**Örnek 18.5.** *Dairesel bir masanın etrafında 6 adet madeni para bulunmaktadır. Başlangıçta 5 tanesi tura (T), 1 tanesi yazı (Y) gelmiştir. Bir adımda, yan yana bulunan herhangi iki parayı aynı anda ters çevirme hakkına sahibiz (yani* $T \leftrightarrow Y$*). Sonlu sayıda adım sonunda tüm paraları tura yapmak mümkün müdür?*

*Çözüm.* Paraları saat yönünde $1, 2, 3, 4, 5, 6$ olarak numaralandıralım. Bir paranın durumunu $T = 0$ ve $Y = 1$ olarak kodlayalım. Yan yana iki paranın ters çevrilmesi durumunda toplam yazı sayısı nasıl değişir?

- İkisi de $Y$ ise, ikisi de $T$ olur: Yazı sayısı 2 azalır.
- Biri $Y$, biri $T$ ise, durumlar yer değiştirir: Yazı sayısı değişmez.
- İkisi de $T$ ise, ikisi de $Y$ olur: Yazı sayısı 2 artar.

Her üç durumda da yazıların toplam sayısı $\Delta Y \in \{-2, 0, +2\}$ kadar değişir. Yani toplam yazı sayısının paritesi bir invaryanttır.

Başlangıçta tam olarak 1 adet yazı vardır (tek sayı). Tüm paraların tura olması durumunda ise 0 adet yazı olacaktır (çift sayı). Parite asla değişmeyeceğinden, tek sayıda yazıdan 0 yazıya ulaşmak imkânsızdır. □

**Örnek 18.6.** *Tahtada sırasıyla $(a, b)$ sayı ikilisi yazılıdır. Bir adımda bu ikili silinip yerine $(a + b, a - b)$ ikilisi yazılabilmektedir. Başlangıçta tahtada $(1, \sqrt{2})$ ikilisi yazılı ise, sonlu sayıda adım sonra $(2, 2\sqrt{2})$ ikilisine ulaşılabilir mi?*

*Çözüm.* Adımları cebirsel olarak inceleyelim. Sayılar doğrusal bir dönüşüme uğramaktadır. Kareler toplamına bakacak olursak:

$$(a + b)^2 + (a - b)^2 = a^2 + 2ab + b^2 + a^2 - 2ab + b^2 = 2(a^2 + b^2)$$

Karelerin toplamı her adımda 2 katına çıkmaktadır. Bu tam bir invaryant olmasa da her $k$ adımda bir $2^k$ çarpanı üretir.

Şimdi doğrudan katsayı yapısını izleyelim. İlk ikili $(1, \sqrt{2})$'dir.

1. adım: $(1 + \sqrt{2}, 1 - \sqrt{2})$
2. adım: Bu ikiliden $((1 + \sqrt{2}) + (1 - \sqrt{2}), (1 + \sqrt{2}) - (1 - \sqrt{2})) = (2, 2\sqrt{2})$.

Görüldüğü gibi ikinci adımda doğrudan hedefe ulaşılır.

Bu örnek önemli bir yöntemsel uyarı barındırır: Her problemde doğrudan imkânsızlık göstermeye odaklanmamak gerekir. İnvaryant yöntemi imkânsızlık ispatlarında oldukça güçlüdür; ancak soru "ulaşılabilir mi?" diye sorduğunda yanıt olumlu da olabilir ve bu durumda yapılması gereken, hamle dizisini doğrudan inşa etmektir. □

**Örnek 18.7.** *Bir çember üzerinde 2024 tane sayı yazılıdır. Bu sayıların her biri ya $+1$ ya da $-1$'dir. Her adımda her sayının yerine, kendisi ile saat yönündeki komşusunun çarpımı aynı anda yazılmaktadır. Başlangıçtaki sayıların çarpımı $-1$ iken sonlu adım sonra tüm sayıların $+1$ olduğu bir duruma ulaşılabileceğini gösteriniz. Bu hedefe ulaşan somut bir başlangıç durumu kurunuz ve çarpım invaryantının bu soruyu neden çözmediğini açıklayınız.*

*Çözüm.* Çemberdeki sayıları saat yönünde $x_1, x_2, \ldots, x_n$ ($n = 2024$) olarak adlandıralım (indisler mod $n$ alınmaktadır). Bir adım uygulandığında yeni sayılar $x_i' = x_i x_{i+1}$ olur. Tüm sayıların çarpımını inceleyelim:

$$P' = (x_1x_2)(x_2x_3)\cdots(x_nx_1) = x_1^2x_2^2\cdots x_n^2 = 1$$

Her bir $x_i \in \{-1, 1\}$ olduğundan $x_i^2 = 1$'dir. Dolayısıyla ilk adımdan sonra sayıların çarpımı her zaman 1 olur. Ancak hedef durum olan "tüm sayıların $+1$ olması" halinde de çarpım 1 olduğundan, bu çarpım invaryantı hedefe ulaşmayı yasaklamaz; invaryantın yetersizliği tam olarak buradadır.

Dönüşümü ardışık adımlar için inceleyelim. İki adım üst üste uygulandığında:

$$x_i^{(2)} = x_i'x_{i+1}' = (x_ix_{i+1})(x_{i+1}x_{i+2}) = x_ix_{i+1}^2x_{i+2} = x_ix_{i+2}$$

elde edilir. Benzer biçimde ilerletilirse, genel olarak $2^k$ adım sonra:

$$x_i^{(2^k)} = x_i \cdot x_{i+2^k}$$

olur.

Şimdi başlangıç çarpımı $-1$ iken sonlu adımda tüm sayıların $+1$ yapılabileceğini somut bir örnekle gösterelim. $n = 2024$ için, indisleri $i \equiv 0 \pmod 8$ olan konumlara $-1$, diğer tüm konumlara $+1$ yerleştirelim. $2024 = 8 \cdot 253$ olduğundan dizilimde 253 tane $-1$ vardır ve başlangıçtaki çarpım:

$$P = (-1)^{253} = -1$$

olur. Tanımlanan dizilim 8'li periyodik olduğundan her $i$ için $x_{i+8} = x_i$ eşitliği sağlanır. Buradan $k = 3$ seçilirse, elde ettiğimiz bağıntı gereği 8 adım sonra:

$$x_i^{(8)} = x_i \cdot x_{i+8} = x_i^2 = +1$$

bulunur. Demek ki yalnızca 8 adım sonra çemberdeki tüm sayılar $+1$ olur. Böylece hedefe ulaşılamayacağı yönündeki iddia çürütülmüş olur; istenen durum mümkündür. □

### 18.1.5 Kod ile Simülasyon

İnvaryantın korunduğunu doğrulamak ve sezgi geliştirmek için bir simülasyon yazabiliriz. Bukalemun problemini simüle eden aşağıdaki sözde kodu inceleyelim:

```
FUNCTION chameleon_simulation(yellow, green, red, max_steps = 1000):
    state = [yellow, green, red]

    FOR step FROM 1 TO max_steps DO
        available_colors = empty list
        FOR index FROM 0 TO 2 DO
            IF state[index] > 0 THEN
```

```
                APPEND index TO available_colors
            END IF
        END FOR

        IF LENGTH(available_colors) < 2 THEN
            BREAK
        END IF

        SELECT TWO DISTINCT elements i, j RANDOMLY FROM available_colors

        state[i] = state[i] - 1
        state[j] = state[j] - 1
        state[3 - i - j] = state[3 - i - j] + 2

        d1 = (state[0] - state[1]) MOD 3
        d2 = (state[1] - state[2]) MOD 3

        ASSERT d1 == (yellow - green) MOD 3
        ASSERT d2 == (green - red) MOD 3
    END FOR

    RETURN state
END FUNCTION

initial_yellow = 13
initial_green = 15
initial_red = 17

final_state = chameleon_simulation(initial_yellow, initial_green,
    initial_red)
OUTPUT "Reached final state:", final_state
```

### 18.1.6 En Sık Yapılan Hatalar

- **Yalancı İnvaryant:** Bir özelliğin sadece birkaç hamlede korunduğunu görüp tüm hamlelerde geçerli olduğunu varsaymak. Bir ifadenin invaryant olduğunu iddia ediyorsanız, izin verilen istisnasız her hamle altında değerinin korunduğunu cebirsel olarak göstermelisiniz.

- **Ters Yön Hatası:** $f(s_{\text{hedef}}) = f(s_0)$ olmasının hedefe ulaşılabileceğini garantilemediğini unutmak. İnvaryant yalnızca imkânsızlığı kanıtlar; ulaşılabilirliği göstermek için açık bir hamle dizisi (yapıcı algoritma) ortaya konmalıdır.

## 18.2 Monovaryant

İnvaryant bir büyüklüğün sabitliğini kullanırken, **yarı-değişmez (monovaryant)** her adımda tek bir yöne doğru (monoton) değişen bir niceliktir.

Monovaryantın en temel işlevi, algoritmik süreçlerin **sonlanacağını (termination)** ispatlamasıdır. Bilgisayar biliminde bir algoritmanın sonsuz döngüye girmeden duracağını kanıtlamak, olimpiyatlarda ise verilen bir oyunun veya sürecin nihayete ereceğini göstermek monovaryant yaklaşımıyla çözüme kavuşturulur.

### 18.2.1 Motivasyon ve Sezgi: Komşularla Barışma

Klasik bir örnekle başlayalım:

> Bir mecliste $n$ parlamenter bulunmaktadır. Her parlamenterin meclis içinde en fazla 3 tane düşmanı vardır. Meclis başkanı, parlamenterleri iki farklı komisyona öyle bir ayırmak istemektedir ki, her parlamenterin kendi komisyonunda en fazla 1 düşmanı bulunsun. Böyle bir ayrım her zaman mümkün müdür?

Parlamenterleri $A$ ve $B$ komisyonlarına gelişigüzel dağıtarak başlayalım. Muhtemelen bazı parlamenterlerin kendi komisyonlarında 2 veya 3 düşmanı bulunacaktır.

Durumu düzeltmek için Bölüm 27'de ayrıntılarını ele alacağımız greedy bir kural belirleyelim:

> "Kendi komisyonunda 2 veya daha fazla düşmanı bulunan herhangi bir parlamenteri alıp diğer komisyona aktar."

Bir parlamenterin kendi grubunda $\geq 2$ düşmanı varsa ve toplamda en fazla 3 düşmanı olabiliyorsa, diğer grupta en fazla $3 - 2 = 1$ düşmanı bulunabilir. Dolayısıyla bu geçiş yapıldığında, o kişi kendi açısından daha az çatışmalı bir ortama geçmiş olur.

Burada temel soru şudur: Bir parlamenteri karşıya geçirdiğimizde, karşı gruptakilerin düşman sayısı artabilir; bu durum zincirleme bir etkiyle parlamenterlerin gruplar arasında sonsuza kadar gidip gelmesine yol açabilir mi?

Sonsuz döngü ihtimalini elemek için sistemin tamamını niteleyen bir büyüklük tanımlayalım:

$$E = \text{Aynı komisyonda bulunan düşman parlamenter çiftlerinin toplam sayısı}$$

Bir parlamenter $P$, kendi komisyonunda $d_{\text{iç}} \geq 2$ düşmana ve karşı komisyonda $d_{\text{dış}} \leq 1$ düşmana sahipken karşıya geçtiğinde $E$ nasıl değişir?

- $P$'nin eski komisyonundaki $d_{\text{iç}}$ adet düşmanıyla olan bağı kopar ($E$ değeri $d_{\text{iç}}$ kadar azalır).

- $P$'nin yeni komisyonundaki $d_{\text{dış}}$ adet kişiyle olan düşmanlıkları aynı komisyon içine girer ($E$ değeri $d_{\text{dış}}$ kadar artar).

Yeni toplam düşmanlık sayısı $E'$:

$$E' = E - d_{\text{iç}} + d_{\text{dış}}$$

Burada $d_{\text{iç}} \geq 2$ ve $d_{\text{dış}} \leq 1$ olduğundan:

$$E' \leq E - 2 + 1 = E - 1$$

Her hamlede $E$ değeri **en az 1 azalmaktadır**.

$E$ sayısı negatif olamaz ($E \geq 0$). Bir tam sayı her adımda azalarak alttan sınırlı bir kümede sonsuza kadar hareket edemez; bu nedenle süreç sonlu sayıda adım sonra durmak zorundadır. Süreç durduğunda kuralı uygulayacak hiçbir parlamenter kalmamış olur ki bu da her parlamenterin kendi komisyonunda en fazla 1 düşmanının kaldığı hedef durumun sağlandığını gösterir.

### 18.2.2 Tanım ve Teorem

**Tanım 18.8** (Monovaryant / Yarı-Değişmez)**.** *Bir sistemin her durum geçişinde monoton olarak kesin biçimde azalan (veya artan) ve alttan (veya üstten) sınırlı olan bir büyüklüğe* ***yarı-değişmez (monovaryant)*** *veya* ***Lyapunov fonksiyonu*** *denir.*

**Teorem 18.9** (Sonlanma İlkesi)**.** *Bir ayrık durum uzayında tanımlı $M$ monovaryantı tam sayı değerler alsın. Eğer her adımda:*

$$M(s_{k+1}) \leq M(s_k) - 1$$

*ve her durum $s$ için $M(s) \geq C$ (alttan sınırlı) ise, süreç en fazla $M(s_0) - C$ adım sonra durmak zorundadır.*

*Kanıt.* $k$ adım sonra $M(s_k) \leq M(s_0) - k$ olur. Eğer süreç $k > M(s_0) - C$ adım devam edebilseydi, $M(s_k) < M(s_0) - (M(s_0) - C) = C$ elde edilirdi ki bu durum $M$'nin alttan $C$ ile sınırlı olmasıyla çelişir. Dolayısıyla süreç durmak zorundadır. □

### 18.2.3 Çözümlü Örnekler

**Örnek 18.10.** *Düzlemde $n$ adet kırmızı nokta ve $n$ adet mavi nokta bulunmaktadır. Hiçbir üç nokta doğrusal değildir. Kırmızı noktalar ile mavi noktaları, çizilen doğru parçaları birbirini kesmeyecek şekilde birebir eşleyen doğru parçaları çizmenin daima mümkün olduğunu kanıtlayınız.*

*Çözüm.* Kırmızı noktalar kümesi $K$, mavi noktalar kümesi $M$ olsun. Bölüm 4'te gördüğümüz permütasyon kavramı uyarınca toplam $n!$ farklı birebir eşleme yapılabilir ve durum uzayı sonludur.

Olası herhangi bir eşlemede, kırmızı $R_i$ noktası ile mavi $B_{\pi(i)}$ noktası birleştirilsin ($\pi$, $\{1, \ldots, n\}$ kümesinin bir permütasyonudur).

İki doğru parçası kesiştiğinde üçgen eşitsizliğini kullanarak mesafeleri kısaltma yoluna gidebiliriz. Diyelim ki $R_1B_1$ ve $R_2B_2$ doğru parçaları bir $X$ noktasında kesişiyor olsun. Eşleşmeyi değiştirip $R_1$'i $B_2$'ye, $R_2$'yi $B_1$'e bağlayalım.

Tüm doğru parçalarının uzunlukları toplamını monovaryant olarak seçelim:

$$L = \sum_{i=1}^{n} |R_i B_{\pi(i)}|$$

Kesişen iki doğru parçasının uzunlukları toplamı:

$$\begin{aligned} |R_1B_1| + |R_2B_2| &= (|R_1X| + |XB_1|) + (|R_2X| + |XB_2|) \\ &= (|R_1X| + |XB_2|) + (|R_2X| + |XB_1|) \end{aligned}$$

Üçgen eşitsizliğine göre, $R_1, X, B_2$ doğrusal olmadığından:

$$|R_1B_2| < |R_1X| + |XB_2|$$

ve benzer şekilde:

$$|R_2B_1| < |R_2X| + |XB_1|$$

Taraf tarafa toplarsak:

$$|R_1B_2| + |R_2B_1| < |R_1B_1| + |R_2B_2|$$

Kesişen iki parçanın çapraz bağlanması durumunda, toplam uzunluk $L$ kesin olarak azalır.

Olası tüm eşlemelerin sayısı $n!$ olup sonludur. Her permütasyona karşılık gelen $L$ değerleri kümesi de sonlu bir kümedir. Sonlu bir reel sayı kümesinin mutlaka bir minimum elemanı vardır. Toplam uzunluğu minimum yapan eşlemede hiçbir kesişme bulunamaz; çünkü bir kesişme olsaydı yukarıdaki işlemle uzunluğu daha da azaltabilirdik. Bu da kesişimsiz bir eşlemenin varlığını ispatlar. □

**Örnek 18.11.** *Bir beşgenin köşelerine, toplamı pozitif olan beş tam sayı yazılmıştır. Ardışık üç köşede sırasıyla $x, y, z$ sayıları bulunsun. Eğer $y < 0$ ise şu hamle yapılabilir: bu üç sayı sırasıyla $x + y$, $-y$ ve $z + y$ ile değiştirilir. Bu hamle, köşelerden en az biri negatif olduğu sürece tekrarlanır. Sürecin her zaman sonlu sayıda adımda durduğunu kanıtlayınız. (Bu problem, 1986 Uluslararası Matematik Olimpiyatı'nın 3. sorusudur.)*

*Çözüm.* Köşelerdeki sayıları saat yönünde $x_1, x_2, \ldots, x_5$ ile gösterelim ve indeksleri mod 5'te ele alalım ($x_6 = x_1$ ve benzeri).

Önce toplamın değişmediğini görelim. $x_i < 0$ seçildiğinde $x_i$ yerine $-x_i$ yazılırken iki komşusuna $x_i$ eklenir; toplamdaki net değişim

$$\Delta S = (-x_i - x_i) + x_i + x_i = 0$$

olduğundan $S = x_1 + \cdots + x_5$ bir **değişmezdir** (invariant). Varsayım gereği $S > 0$'dır.

Şimdi pentagramın köşegenleri üzerinde

$$\Phi = \frac{1}{2}\sum_{i=1}^{5}(x_i - x_{i+2})^2$$

niceliğini tanımlayalım. Sayılar tam sayı olduğundan $\Phi$ negatif olmayan bir tam sayıdır.

Bir hamlenin $\Phi$ üzerindeki etkisini hesaplayalım. $D_j = x_j - x_{j+2}$ ve $t = -x_i > 0$ olsun. Hamlede $x_i$ yerine $-x_i$ yazıldığından $\Delta x_i = 2t$; komşularına $x_i = -t$ eklendiğinden $\Delta x_{i\pm1} = -t$ olur, diğer köşeler değişmez. Buradan

$$\Delta D_i = 2t, \quad \Delta D_{i+1} = -t, \quad \Delta D_{i+2} = t, \quad \Delta D_{i+3} = -2t, \quad \Delta D_{i+4} = 0$$

elde edilir. İkinci derece fark terimleri açıldığında

$$\Delta\Phi = \frac{1}{2}\Big(2t\,(2D_i - D_{i+1} + D_{i+2} - 2D_{i+3}) + 10t^2\Big)$$

olur. Terimler yazıldığında $x_i$ dışındaki her değişkenin tam olarak bir kez geldiği görülür:

$$2D_i - D_{i+1} + D_{i+2} - 2D_{i+3} = 5x_i - S.$$

$x_i = -t$ olduğundan

$$\Delta\Phi = t\,(5x_i - S) + 5t^2 = 5tx_i - tS + 5t^2 = -5t^2 - tS + 5t^2 = -tS < 0$$

bulunur; yani $\Phi$ her hamlede kesin olarak azalır. $\Phi$ negatif olmayan bir tam sayı, azalma miktarı $tS$ ise en az 1 olduğundan süreç en fazla $\Phi_0$ adım sürer ve sonlu adımda durmak zorundadır. □

**Örnek 18.12.** *Bir $8 \times 8$ satranç tahtasında başlangıçta bazı kareler doludur. Her saniye, en az iki dolu komşusu (ortak kenara sahip) bulunan boş bir kare daha dolmaktadır. Tahtanın tamamının dolabilmesi için başlangıçta en az kaç karenin dolu olması gerektiğini gösteriniz.*

*Çözüm.* Dolan kare sayısı her hamlede arttığı için taş sayısının kendisi bir yarıdeğişmez olarak kullanılamaz; bunun yerine dolgulu bölgenin **çevre uzunluğunu** ($P$) izleyelim.

Bir karenin 4 kenarı vardır. Boş bir kare dolduğunda:

- Yeni karenin $k$ adet dolu komşusu varsa, bu $k$ kenar daha önce dış çevreye aitti; hamleyle birlikte içeride kalır.
- Yeni karenin geriye kalan $4-k$ kenarı çevreye eklenir.

Çevredeki net değişim:

$$\Delta P = (4-k) - k = 4 - 2k$$

Yeni bir karenin dolabilmesi kural gereği $k \geq 2$ olmasını gerektirir; bu durumda

- $k = 2$ ise: $\Delta P = 4 - 4 = 0$ (çevre değişmez).
- $k = 3$ ise: $\Delta P = 4 - 6 = -2$ (çevre 2 azalır).
- $k = 4$ ise: $\Delta P = 4 - 8 = -4$ (çevre 4 azalır).

Görüldüğü gibi çevre hiçbir zaman artmaz: $P' \leq P$. Yani $P$ bir yarı-değişmezdir.

Tahtanın tamamı dolduğunda oluşan $8 \times 8$ büyük karenin çevresi $4 \times 8 = 32$'dir. Başlangıçtaki $m$ dolu karenin her birinin çevreye katkısı en fazla 4 olabileceğinden $P_0 \leq 4m$'dir. Çevre artamadığına göre:

$$32 = P_{\text{son}} \leq P_0 \leq 4m \implies 4m \geq 32 \implies m \geq 8$$

elde edilir. Böylece başlangıçta en az 8 karenin dolu olması gerektiği gösterilmiş olur. □

### 18.2.4 C Kodu ile Monovaryant Simülasyonu

Parlamenter probleminin sonlanmasını modelleyen algoritmanın C dilindeki temel karşılığı:

```c
#include <stdio.h>
#include <stdbool.h>
#define N 6 // Parlamenter sayisi

// Dusmanlik matrisi (1: dusman, 0: dost)
int dusman[N][N] = {
    {0, 1, 1, 1, 0, 0},
    {1, 0, 1, 1, 0, 0},
    {1, 1, 0, 1, 0, 0},
    {1, 1, 1, 0, 1, 0},
    {0, 0, 0, 1, 0, 1},
    {0, 0, 0, 0, 1, 0}
};

int grup[N]; // 0: Komisyon A, 1: Komisyon B

// Monovaryant: Ayni gruptaki toplam dusman cifti sayisi
```

```
int monovaryant_hesapla() {
    int toplam = 0;
    for (int i = 0; i < N; i++) {
        for (int j = i + 1; j < N; j++) {
            if (grup[i] == grup[j] && dusman[i][j] == 1) {
                toplam++;
            }
        }
    }
    return toplam;
}

void coz() {
    bool degisim = true;
    while (degisim) {
        degisim = false;
        for (int i = 0; i < N; i++) {
            int ic_dusman = 0;
            int dis_dusman = 0;
            for (int j = 0; j < N; j++) {
                if (dusman[i][j]) {
                    if (grup[i] == grup[j]) ic_dusman++;
                    else dis_dusman++;
                }
            }
            // Eger kendi grubunda 2 veya daha fazla dusman varsa diger
    gruba gec
            if (ic_dusman > dis_dusman) {
                grup[i] = 1 - grup[i]; // Grubu degistir
                degisim = true;
                printf("Monovaryant (Ic Dusmanliklar Toplami): %d\n",
    monovaryant_hesapla());
                break; // Her seferinde tek transfer yap
            }
        }
    }
}
```

### 18.2.5 En Sık Yapılan Hatalar

- **Tükenme Garantisi Olmayan Azalma:** Sonsuz bir kümede bir büyüklüğün sürekli azalması sürecin sonlanacağını tek başına garanti etmez. Örneğin $x_{k+1} = \frac{x_k}{2}$ dizisinde değerler sürekli azalır ve alttan 0 ile sınırlıdır; ancak adım sayısı sonsuzdur. Monovaryantın sonlanmayı garanti etmesi için adımlardaki azalma miktarının alttan pozitif bir sabitle sınırlı olması (örneğin tam sayılarda en az 1 azalması) ya da durum uzayının **sonlu** olması gerekir.

- **Zayıf Monotonluk:** $M(s_{k+1}) \leq M(s_k)$ olması sistemin bir döngüye girmesine engel olmayabilir ($M$ sabit kalırken durumlar değişebilir). Sonlanmayı kesinleştirmek için kesin eşitsizlik ($M(s_{k+1}) < M(s_k)$) sağlanmalı ya da eşitlik durumunda döngü oluşmadığı ayrıca gösterilmelidir.

# Bölüm 19

# Uç Değer ve Tersine Çalışma

Olimpiyat matematiğinde ve algoritmik düşüncede bazı problemler ilk bakışta çok karmaşık görünebilir. Ortada hareket eden onlarca nesne, sürekli değişen durumlar veya sonsuz büyüklükte kümeler varken, sistemin herhangi bir noktasına rastgele odaklanmak genellikle çözümü zorlaştırır.

Bu tür durumlarda iki temel strateji öne çıkar:

1. **Uç Değer İlkesi (Extremal Principle):** Karmaşık bir sürecin ortalama durumlarına değil; en büyüğüne, en küçüğüne, en uzaktakine veya en kısasına odaklanmak.
2. **Tersine Çalışma (Working Backwards):** Başlangıçtan hedefe doğru çok sayıda kola ayrılan seçenekler arasında kaybolmak yerine, varılmak istenen sondan başlayıp geriye doğru adımlamak.

Bu bölümde, bu iki düşünce biçiminin hem matematiksel ispatlarda hem de algoritma tasarımında karmaşık yapıları nasıl sadeleştirdiğini somut örnekler üzerinden ele alacağız.

## 19.1 Uç Değer İlkesi

Bir problemde incelenen nesnelerin tümü aynı anda karmaşık ilişkiler içinde bulunabilir. Ancak bu nesneleri ölçülebilir bir büyüklüğe (uzunluk, eleman sayısı, ağırlık, koordinat vb.) göre sıraladığımızda, kaçınılmaz olarak bir *sınır*—yani bir en küçük ya da en büyük eleman—ortaya çıkar. Uç değer ilkesi şu yalın gözleme dayanır: **Ekstremum noktasında hareket alanı kısıtlıdır; bu kısıtlılık ise aradığımız çelişkiyi veya yapıyı kurmamızı sağlar.**

### 19.1.1 İyi Sıralılık ve Minimal Karşı Örnek

Doğal sayıların temel bir özelliğiyle başlayalım.

**Tanım 19.1** (İyi Sıralılık İlkesi (Well-Ordering Principle))**.** *Doğal sayılar kümesinin ($\mathbb{N} = \{0, 1, 2, \dots\}$ veya $\mathbb{Z}^+ = \{1, 2, 3, \dots\}$) boş olmayan her alt kümesinin en küçük bir elemanı vardır.*

Bu ilke sezgisel olarak son derece açıktır; ancak Bölüm 2'de ele aldığımız çelişkiyle ispat yönteminin en etkili biçimlerinden birini oluşturur. İlkenin problem çözmedeki karşılığına **minimal karşı örnek yöntemi** (veya Fermat'nın adlandırdığı şekliyle *sonsuz iniş yöntemi*) denir.

Bir $P(n)$ iddiasının tüm pozitif tam sayılar için doğru olduğunu kanıtlamak istediğimizi varsayalım. Doğrudan ispat kurmak güçse, iddianın yanlış olduğunu varsayarız. O halde iddiayı sağlamayan (karşı örnek oluşturan) sayıların kümesi boştan farklıdır:

$$S = \{n \in \mathbb{Z}^+ \mid P(n) \text{ yanlıştır}\}$$

İyi sıralılık ilkesine göre, $S$ kümesinin **en küçük** bir elemanı olmak zorundadır. Bu elemana $m$ diyelim; yani $m$, iddiayı bozan en küçük tam sayıdır. Eğer sistemin kurallarını kullanarak $m$'den daha küçük ve yine iddiayı bozan bir $m' < m$ tam sayısı üretebilirsek, $m$'nin en küçük oluşuyla çelişiriz. Bu çelişki, $S$ kümesinin aslında boş olduğunu, dolayısıyla hiçbir karşı örneğin var olamayacağını gösterir.

**Örnek 19.2.** *$\sqrt{2}$ sayısının rasyonel bir sayı olmadığını minimal karşı örnek yöntemiyle gösteriniz.*

*Çözüm.* İddianın aksine $\sqrt{2}$ rasyonel olsun. O halde $\sqrt{2} = a/b$ eşitliğini sağlayan $a, b \in \mathbb{Z}^+$ sayıları bulunur. Bu denklemi sağlayan tüm olası paydaları içeren

$$S = \{b \in \mathbb{Z}^+ \mid \exists a \in \mathbb{Z}^+ \text{ öyle ki } a^2 = 2b^2\}$$

kümesini tanımlayalım. Varsayımımıza göre $S$ boş değildir. İyi sıralılık ilkesi gereği, $S$ kümesinin en küçük bir elemanı vardır; buna $b_0$ diyelim. Karşılık gelen pay da $a_0$ olsun:

$$a_0^2 = 2b_0^2$$

Bu eşitlik $a_0^2$'nin çift olduğunu gösterir. Bölüm 17'de incelediğimiz parite özellikleri uyarınca, bir tam sayının karesi çift ise kendisi de çifttir. Dolayısıyla bir $k \in \mathbb{Z}^+$ için $a_0 = 2k$ yazabiliriz. Eşitlikte yerine koyarsak:

$$(2k)^2 = 2b_0^2 \implies 4k^2 = 2b_0^2 \implies b_0^2 = 2k^2$$

elde edilir. Aynı gerekçeyle $b_0^2$ çift olduğundan $b_0$ da çifttir. Yani bir $m \in \mathbb{Z}^+$ için $b_0 = 2m$ yazılabilir:

$$(2m)^2 = 2k^2 \implies 4m^2 = 2k^2 \implies k^2 = 2m^2$$

Bu eşitlik $k$ ve $m$ pozitif tam sayıları için $k/m = \sqrt{2}$ olduğunu söyler.

Dolayısıyla $m \in S$ olmalıdır. Ancak $b_0 = 2m$ olduğundan $m < b_0$'dır. Bu durum, $b_0$'ın $S$ kümesindeki en küçük eleman olmasıyla çelişir. Bu çelişki baştaki kabulümüzü çürütür; $\sqrt{2}$ rasyonel olamaz. □

### 19.1.2 En Büyük Asal Sayı Neden Yoktur?

Bölüm 12'de asal sayıları incelerken gördüğümüz Öklid'in asalların sonsuzluğuna dair ispatı, özünde uç değer ilkesinin doğrudan bir uygulamasıdır.

**Teorem 19.3.** *Asal sayılar kümesi sonsuzdur; yani "en büyük asal sayı" yoktur.*

*Kanıt.* Asal sayılar kümesinin sonlu olduğunu varsayalım. Bu durumda sonlu sayıdaki tüm asalları küçükten büyüğe sıralayabiliriz:

$$p_1 = 2 < p_2 = 3 < p_3 = 5 < \cdots < p_k$$

Burada $p_k$, en büyük asal sayıdır.

Şimdi bu uç değerden faydalanarak yeni bir sayı kuralım. Bilinen tüm asalların çarpımının 1 fazlası olan $N$ sayısını tanımlayalım:

$$N = (p_1 \cdot p_2 \cdot p_3 \cdots p_k) + 1$$

Bölüm 12'de ifade ettiğimiz Aritmetiğin Temel Teoremi gereği $N > 1$ sayısı en az bir asal bölen içermelidir; bu bölen $q$ olsun.

- Eğer $q$, listemizdeki asallardan biriyse ($q \in \{p_1, p_2, \ldots, p_k\}$), o halde $q$ sayısı $p_1 \cdot p_2 \cdots p_k$ çarpımını tam böler.
- Fakat $q$ aynı zamanda $N$'yi de böldüğünden, bu iki sayının farkını da bölmek zorundadır:
$$q \mid \Big(N - (p_1 \cdot p_2 \cdots p_k)\Big) \implies q \mid 1$$

1'i bölen bir asal sayı bulunmadığından bu bir çelişkidir. O halde $q$ asalı, listemizdeki $p_1, \ldots, p_k$ asallarının hiçbirine eşit olamaz; yani listede yer almayan yeni bir asal sayıdır. Üstelik $q > p_k$ olmak zorundadır.

Bu durum, $p_k$'nın en büyük asal olması kabulüyle çelişir. Dolayısıyla en büyük asal sayı yoktur; asal sayılar kümesi sonsuzdur. □

### 19.1.3 Kombinatorikte Maksimum ve Minimumun Gücü

Uç değer ilkesi yalnızca sayılar teorisinde değil, graf teorisinde ve sonlu kombinatorik yapılarda da önemli kolaylıklar sunar. Bir sistemde rastgele değerler yerine **maksimum** ya da **minimum** değere sahip elemanı seçtiğimizde, o elemanın komşuları veya ilişkili olduğu diğer bileşenler hakkında doğrudan eşitsizlikler elde ederiz.

**Örnek 19.4.** *Bir çember etrafına $n$ tane reel sayı dizilmiştir. Her bir sayı, sağındaki ve solundaki iki komşusunun aritmetik ortalamasına eşittir. Bu sayıların hepsinin birbirine eşit olduğunu kanıtlayınız.*

*Çözüm.* Sayıları $x_1, x_2, \ldots, x_n$ olarak seçip denklemleri taraf tarafa toplamak yeni bir bilgi vermez. Bunun yerine kümedeki en büyük sayıya odaklanalım.

Çemberdeki sayıların kümesi sonlu olduğundan, bu kümenin en az bir maksimum elemanı bulunur. Bu elemana $M$ diyelim. $M$'nin sağındaki komşusu $a$, solundaki komşusu $b$ olsun.

Koşul gereği:

$$M = \frac{a+b}{2} \iff 2M = a + b$$

Fakat $M$ dizideki en büyük eleman olduğundan, tanım gereği $a \leq M$ ve $b \leq M$'dir.

Eğer $a < M$ olsaydı, $a + b < M + M = 2M$ olurdu ve eşitlik bozulurdu. Benzer şekilde $b < M$ de olamaz. Dolayısıyla tek olasılık:

$$a = M \quad \text{ve} \quad b = M$$

olmasıdır.

Böylece maksimum bir elemanın her iki komşusunun da zorunlu olarak $M$'ye eşit olduğu görülür. Aynı akıl yürütmeyi komşuların komşularına zincirleme biçimde uyguladığımızda, çember üzerindeki tüm sayıların $M$'ye eşit olduğu anlaşılır. Sonuç olarak çemberdeki bütün sayılar birbirine eşittir. □

**Örnek 19.5.** *Bir satranç turnuvasında her oyuncu diğer tüm oyuncularla tam olarak bir maç yapmıştır (beraberlik yoktur; her maçta biri kazanmış, diğeri kaybetmiştir). Turnuvada öyle bir $T$ oyuncusunun var olduğunu kanıtlayınız ki; diğer her $X$ oyuncusu için ya $T$, $X$'i doğrudan yenmiş olsun ya da $T$'nin yendiği bir $Y$ oyuncusu $X$'i yenmiş olsun. (Kısacası $T$, herkese en fazla iki adımda ulaşabilmektedir.)*

*Çözüm.* Koşulu sağlamaya en uygun adaya, yani turnuvada galibiyet sayısı maksimum olan oyuncuya odaklanalım.

Turnuvada en çok maç kazanan oyuncuyu seçip $T$ olarak adlandıralım. $T$'nin doğrudan yendiği oyuncuların kümesine $W$, $T$'yi yenen oyuncuların kümesine ise $L$ diyelim.

$T$'nin galibiyet sayısı $|W| = k$ olsun. Tanım gereği turnuvadaki hiçbir oyuncunun galibiyet sayısı $k$'dan fazla olamaz.

İspatlamak istediğimiz iddia şudur: $L$ kümesindeki her oyuncu, $W$ kümesindeki en az bir oyuncu tarafından yenilmiştir.

Aksini varsayalım: $L$ kümesinde öyle bir $X$ oyuncusu bulunsun ki, $W$'deki hiç kimse $X$'i yenememiş olsun. Beraberlik olmadığına göre, bu durum $X$'in $W$'deki bütün oyuncuları yendiği anlamına gelir.

Şimdi $X$'in yendiği kişi sayısını belirleyelim:

1. $X$, $T$'yi yenmiştir ($X \in L$ olduğundan).

2. $X$, $W$ kümesindeki tüm oyuncuları yenmiştir (varsayım gereği $|W| = k$ kişi).

Dolayısıyla $X$, en az $k + 1$ galibiyet almıştır.

Fakat bu bir çelişkidir; çünkü turnuvada en fazla galibiyet alan oyuncu $T$ idi ve yalnızca $k$ galibiyeti vardı. Bir oyuncunun $k + 1$ galibiyete ulaşması olanaksızdır.

Bu çelişki varsayımı çürütür. Demek ki $L$'deki her $X$ oyuncusu, $W$'deki en az bir $Y$ oyuncusu tarafından yenilmek zorundadır. Böylece $T \to Y \to X$ bağlantısı kurulur ve iddia kanıtlanmış olur. □

```
FUNCTION findTournamentLeader(winMatrix)
    n = LENGTH(winMatrix)
    INITIALIZE winCounts ARRAY OF SIZE n

    FOR i FROM 0 TO n - 1 DO
        winCounts[i] = 0
        FOR j FROM 0 TO n - 1 DO
            winCounts[i] = winCounts[i] + winMatrix[i][j]
        END FOR
    END FOR

    bestPlayer = 0
    maxWins = winCounts[0]

    FOR i FROM 1 TO n - 1 DO
        IF winCounts[i] > maxWins THEN
            maxWins = winCounts[i]
            bestPlayer = i
        END IF
    END FOR

    RETURN bestPlayer
END FUNCTION
```

## En Sık Yapılan Hata

Uç değer ilkesini kullanırken en sık düşülen yanılgı, **sonsuz kümelerde uç değerin varlığını peşinen varsaymaktır**.

Örneğin, reel sayılarda $(0, 1)$ açık aralığında en küçük ya da en büyük eleman tanımlı değildir. Bir problemde uç değer seçilecekse, şu iki güvenceden en az biri sağlanmalıdır:

1. Küme **sonludur** (boş olmayan her sonlu kümenin minimumu ve maksimumu mevcuttur).

2. Küme, **doğal sayılar** gibi aşağıdan sınırlı ve ayrık bir kümedir (İyi Sıralılık İlkesi gereği minimum elemanı vardır).

## 19.2 Tersine Çalışma

Bir labirentin giriş kapısından bakıldığında önünüzde onlarca koridor uzanır ve her koridor yeni yol ayrımlarına açılır. Ancak labirentin çıkış kapısından geriye doğru bakıldığında, çıkışa bağlanan yolların sayısı genellikle çok daha azdır.

Algoritmik problem çözmede **tersine çalışma (working backwards)**, özellikle şu iki durumda temel bir stratejidir:

1. Hedef durum belirgin ve kısıtlıyken, başlangıç durumunun çok fazla seçenek barındırması.
2. İleriye doğru yapılan işlemlerin dallanmayı hızla artırması, buna karşılık geriye doğru adımların tekil (deterministik) olması.

### 19.2.1 Oyun Teorisi ve Geriye Dönük Çıkarım

Tersine çalışmanın en belirgin uygulama alanlarından biri iki kişilik kombinatorik oyunlardır. Hamleleri baştan itibaren incelemek yerine, oyunun bittiği andan (terminal durum) geriye doğru durumlar sınıflandırılır:

- **Kayıp Durum (L - Losing State):** Sırası gelen oyuncunun yaptığı her hamle rakibini bir kazanç durumuna götürüyorsa (veya geçerli hiçbir hamlesi kalmamışsa), bu bir kayıp durumudur.
- **Kazanç Durum (W - Winning State):** Sırası gelen oyuncunun, rakibini bir kayıp durumuna ($L$) sokabileceği *en az bir* hamlesi bulunuyorsa, bu bir kazanç durumudur.

**Örnek 19.6.** *Masanın üzerinde 21 adet taş bulunmaktadır. İki oyuncu sırayla hamle yapar. Sırası gelen oyuncu masadan 1, 2 veya 3 taş alabilir. Son taşı alan oyuncu oyunu kazanır. İlk başlayan oyuncunun kesin bir kazanma stratejisi var mıdır?*

*Çözüm.* Oyuna baştan başlamak yerine bitiş anından, yani masada 0 taş kaldığı durumdan geriye doğru ilerleyelim:

- **0 Taş:** Hamle kalmamıştır, sırası gelen oyuncu kaybetmiştir. → **L**
- **1, 2, 3 Taş:** Sırası gelen oyuncu tek hamlede 0 taşa ulaşabilir ($L$ durumuna geçebilir). Dolayısıyla bunlar kazanç durumudur. → **W**
- **4 Taş:** Sırası gelen oyuncu 1, 2 veya 3 taş aldığında masada sırasıyla 3, 2 veya 1 taş bırakır. Bu durumların hepsi $W$'dir. Rakibe zorunlu olarak kazanç durumu devredildiği için 4 taş bir kayıp durumudur. → **L**
- **5, 6, 7 Taş:** Sırası gelen oyuncu sırasıyla 1, 2 veya 3 taş alarak masada 4 taş ($L$) bırakabilir. Demek ki bunlar kazanç durumudur. → **W**

- **8 Taş:** Yapılan her hamle masada 7, 6 veya 5 taş ($W$) bırakır. Dolayısıyla 8 bir kayıp durumudur. $\to$ **L**

Bölüm 14'te ele aldığımız modüler aritmetik diliyle ifade edersek, 4'ün katı olan durumlar ($n \equiv 0 \pmod 4$) birer **Kayıp Durum** ($L$), diğer tüm durumlar ise **Kazanç Durum** ($W$)'dur.

Başlangıçta masada 21 taş vardır ve $21 \equiv 1 \pmod 4$ olduğundan 21 bir $W$ durumudur.

**Kazandıran Strateji:** İlk oyuncu ilk hamlesinde 1 taş alarak masada 20 taş bırakır ($L$ durumu). Bundan sonra rakip $k \in \{1, 2, 3\}$ taş aldığında, birinci oyuncu her defasında $4 - k$ taş alarak masadaki taş sayısını 4'ün katı tutmayı sürdürür. Son taşı birinci oyuncu alır ve oyunu kazanır. □

### 19.2.2 İşlemleri Tersine Çevirmek

Bazen bir dizi aritmetik işlem uygulanıp bir hedefe ulaşılması istenir. İleriye doğru arama yapmak genişleyen bir ağaç üretirken, hedeften geriye gitmek arama uzayını önemli ölçüde daraltır.

**Örnek 19.7.** *Başlangıçta elimizde $x = 1$ sayısı vardır. Her adımda sayıyı ya 2 ile çarpabiliriz ($x \to 2x$) ya da sayıya 1 ekleyebiliriz ($x \to x + 1$). 1 sayısından 100 sayısına en az kaç adımda ulaşabiliriz?*

*Çözüm.* 1'den başlayarak ilerlemek her adımda ikiye dallanan bir arama ağacı oluşturur. Oysa hedeften, yani 100'den geriye doğru gidersek seçenekler netleşir:

- İleri hamleler: $(\times 2)$ ve $(+1)$.
- Ters hamleler: $(\div 2)$ [yalnızca sayı çiftse] ve $(-1)$.

Geriye doğru adımlarken amaç sayıyı olabildiğince hızlı küçültmektir. Çift bir sayıyı 2'ye bölmek, 1 çıkarmaktan daha hızlı küçülme sağlar. Dolayısıyla sayı çift olduğu sürece 2'ye bölünmeli; tek olduğunda ise 2'ye bölünemeyeceği için mecburen 1 çıkarılmalıdır.

Geriye doğru adımları uygulayalım:

$$100 \xrightarrow{\div 2} 50 \xrightarrow{\div 2} 25 \xrightarrow{-1} 24 \xrightarrow{\div 2} 12$$
$$\xrightarrow{\div 2} 6 \xrightarrow{\div 2} 3 \xrightarrow{-1} 2 \xrightarrow{\div 2} 1$$

Geriye doğru yapılan hamle dizisi $100 \to 50 \to 25 \to 24 \to 12 \to 6 \to 3 \to 2 \to 1$ olup toplam **8 hamle** sürer.

Bu yolu tersine çevirerek ileriye doğru yazalım:

$$1 \xrightarrow{\times 2} 2 \xrightarrow{+1} 3 \xrightarrow{\times 2} 6 \xrightarrow{\times 2} 12 \xrightarrow{\times 2} 24 \xrightarrow{+1} 25 \xrightarrow{\times 2} 50 \xrightarrow{\times 2} 100$$

Tersine çalışma sayesinde deneme yanılmaya gerek kalmadan en kısa yol doğrudan elde edilir. □

### 19.2.3 Klasik Bir Problem: Yumurta Kırma Problemi

Tersine çalışma ve uç değer dengesinin bilgisayar bilimlerindeki klasik uygulamalarından biri "Yumurta Kırma" (Egg Dropping) problemidir.

**Örnek 19.8.** *Elimizde birbirinin tıpatıp aynısı olan* 2 *adet dayanıklı yumurta ve* 100 *katlı bir bina bulunmaktadır. Yumurtalar belirli bir K katından (kritik kat) veya daha yüksek bir kattan atıldığında kırılmakta, K'nın altındaki katlardan atıldığında ise kırılmamaktadır (kırılmayan yumurta tekrar kullanılabilir). Amacımız, en kötü senaryoda (worst-case) yapacağımız atış sayısını* ***en aza indirmektir****. Kritik katı kesin olarak bulabilmek için en kötü durumda en az kaç atış yapmamız gerekir?*

*Çözüm.* İlk olarak tek bir yumurtamız olsaydı ne yapacağımızı düşünelim. Bu durumda risk alma şansımız kalmazdı; 1. kattan başlayıp $1, 2, 3, \ldots, 100$ şeklinde sırayla yukarı çıkmak gerekirdi. Aksi halde yumurta örneğin doğrudan 10. katta kırılırsa, kritik katın 1 ile 9 arasındaki değerlerden hangisi olduğu belirlenemezdi. Dolayısıyla 1 yumurta ile 100 kat için en kötü senaryoda 100 atış yapılması zorunludur.

Elimizde 2 yumurta varken ilk yumurtayı belirli aralıklarla (örneğin 10'ar kat atlayarak) test edebiliriz:

- İlk yumurtayı 10, 20, 30, ...katlarından atarız.
- Diyelim ki 30. katta kırıldı. Kritik katın 21 ile 29 arasında olduğu anlaşılır.
- Kalan tek yumurtayla 21'den itibaren birer birer deneriz $(21, 22, \ldots, 29)$.

Bu sabit adımlı yaklaşımda en kötü senaryo, ilk yumurtanın 100. katta kırılması (10 atış) ve ikinci yumurtanın 91'den 99'a kadar test edilmesidir (9 atış); toplam $10 + 9 = 19$ atış gerekir.

Daha iyi bir sonuca ulaşmak için uç değer dengesi ve tersine düşünme birlikte kullanılır.

Sabit adımlı yaklaşımda bir dengesizlik söz konusudur: İlk yumurta erken bir katta kırıldığında az, geç bir katta kırıldığında çok atış yapılır. En kötü durumu optimize edebilmek için her senaryoda **toplam atış sayısını dengede tutmak** gerekir.

Hedeflenen maksimum atış sayısı $x$ olsun:

- Birinci atış $x$. kattan yapılır. Kırılırsa, kalan $x - 1$ atış hakkıyla ikinci yumurta $1, 2, \ldots, x-1$ katlarında sırayla denenir. Toplam atış: $1 + (x - 1) = x$.
- Birinci atışta kırılmazsa, geriye en fazla $x - 1$ atış hakkı kalır. Bu nedenle bir sonraki adım $x$ kadar değil, $x - 1$ **kat** yukarı atılmalıdır; yani ikinci atış $x + (x - 1)$ katında yapılır. Kırılırsa, aradaki $(x - 2)$ kat ikinci yumurtayla taranır. Toplam atış: $2 + (x - 2) = x$.

- Kırılmadığı sürece bir sonraki adım $x-2$ kat yukarı taşınır.

Her adımda kalan hak 1 azaldığı için taranan kat aralığı da 1 daraltılır. Böylece $x$ hamlede test edilebilecek maksimum kat sayısı, hedeften geriye doğru ardışık toplamdır:

$$x + (x-1) + (x-2) + \cdots + 2 + 1 = \frac{x(x+1)}{2}$$

Bu kapasitenin 100 katı kapsaması yeterlidir:

$$\frac{x(x+1)}{2} \geq 100 \implies x(x+1) \geq 200$$

$x$ bir tam sayı olduğuna göre:

- $x = 13 \implies 13 \times 14 = 182 < 200$ (yetersiz)
- $x = 14 \implies 14 \times 15 = 210 \geq 200$ (yeterli)

Demek ki en kötü durumda **en az 14 atış** yaparak kritik kat kesin olarak saptanabilir.

Atış planının kat numaraları sırasıyla şöyledir: $14,\ 14+13=27,\ 27+12=39,\ 39+11=50,\ 60,\ 69,\ 77,\ 84,\ 90,\ 95,\ 99,\ 100.$ □

Aşağıdaki sözde kod, verilen kat sayısı için gerekli minimum deneme sayısını formülle ve adımların belirlenmesiyle hesaplar:

```
FUNCTION minEggDrops(totalFloors)
    x = 1
    WHILE (x * (x + 1)) / 2 < totalFloors DO
        x = x + 1
    END WHILE
    RETURN x
END FUNCTION

totalFloors = 100
result = minEggDrops(totalFloors)
PRINT "Minimum drops for " + totalFloors + " floors: " + result

PRINT "Floors to drop from:"
currentFloor = 0
step = result

WHILE currentFloor < totalFloors DO
    currentFloor = currentFloor + step
    PRINT MIN(currentFloor, totalFloors)
    step = step - 1
END WHILE
```

### En Sık Yapılan Hata

Tersine çalışma yönteminde yapılan en yaygın hata, **tersine çevrilen işlemlerin öncelik sırasını karıştırmaktır**.

Örneğin, ileriye doğru çalışan bir dönüşüm şu olsun:

$$f(x) = 2x + 3$$

Bu işlemin tersi, $x$'i 2'ye bölüp ardından 3 çıkarmak değildir. İşlemlerin tersi işlem sırasının tam tersiyle uygulanır:

$$f^{-1}(y) = \frac{y - 3}{2}$$

Önce son uygulanan işlem geri alınır (3 çıkarılır), ardından bir önceki adıma dönülür (2'ye bölünür). İşlem sırasındaki bir hata, geriye dönük kurgunun tümünü geçersiz kılar.

# Bölüm 20

# Algoritma Nedir

## 20.1 Problem, Girdi-Çıktı ve Algoritma

Bilgisayar biliminin temel sorusu "Bilgisayarlar nasıl programlanır?" değil, "Hangi problemler hangi yöntemlerle, ne kadar sürede ve kesinlikle çözülebilir?" sorusudur. Kod yazmak bir inşaat ustasının tuğla dizmesine benzer; mimariyi belirleyen, yapının sağlam durmasını ve amaca ulaşmasını sağlayan asıl unsur ise düşünce planıdır. İşte bu düşünce planına **algoritma** diyoruz.

Gündelik yaşamda algoritma kavramını anlamanın en pratik yolu bir **yemek tarifi** düşünmektir. Elinizde belirli malzemeler vardır: un, yumurta, şeker ve süt. Amacınız ise kabarık, lezzetli bir kek elde etmektir. Bir kek tarifi size adım adım ne yapmanız gerektiğini söyler:

1. 3 yumurta ile 1 su bardağı şekeri beyaz köpük olana dek çırp.
2. 1 su bardağı sütü ve yarım su bardağı sıvı yağı ekleyip 1 dakika karıştır.
3. 2.5 su bardağı un ve bir paket kabartma tozunu eleyerek karışıma ilave et.
4. Önceden 180 dereceye ısıtılmış fırında 35 dakika pişir.

Bu benzetmede:

- **Girdi (Input):** Elimizdeki un, şeker, yumurta gibi başlangıç nesneleridir.
- **Çıktı (Output):** Fırından çıkan pişmiş kek, yani ulaşmak istediğimiz sonuçtur.
- **Tarif:** Malzemeleri çıktıya dönüştüren sıralı, açık ve net adımlar kümesidir.

Ancak mutfaktaki tarifler ile bilgisayar bilimindeki algoritmalar arasında önemli bir fark vardır: Mutfak tarifleri genellikle insan sezgisine güvenir. "Kulak memesi kıvamına gelene kadar yoğur", "bir tutam tuz ekle" ya da "kızarana kadar

fırınla" gibi ifadeler insan için anlamlıdır; çünkü insan gözüyle, dokunuşuyla veya tecrübesiyle bu belirsizliği giderir. Bilgisayarın ise ne tecrübesi, ne gözü, ne de sezgisi vardır. Bilgisayar son derece hızlı, fakat bütünüyle "harfi harfine" itaat eden bir işlemcidir. Bu nedenle bir algoritmanın adımları hiçbir muğlaklığa yer bırakmayacak biçimde kesin olmalıdır.

**Tanım 20.1** (Hesaplamalı Problem ve Örneklem)**.** *Bir **hesaplamalı problem** (computational problem), girdiler kümesi ile her geçerli girdi için beklenen çıktılar arasındaki matematiksel ilişkinin genel tanımıdır.*

*Problemin belirli ve somut bir girdisine o problemin bir **örneklemi** (instance) denir.*

Örneğin, "Verilen bir tamsayı listesini küçükten büyüğe sıralama" bir problemdir. Bu problemin somut bir örneklemi ise $[7, 2, 9, 1, 5]$ listesidir; bu örneklem için beklenen çıktı $[1, 2, 5, 7, 9]$ olmalıdır.

**Tanım 20.2** (Algoritma)**.** *Bir **algoritma**, iyi tanımlanmış bir girdi kümesini alıp, sonlu sayıda adımda, açıkça belirlenmiş kuralları işleterek istenen çıktıyı üreten iyi tanımlanmış hesaplama adımları dizisidir.*

Modern bilgisayar biliminin öncülerinden Donald Knuth, bir sürecin tam anlamıyla bir "algoritma" sayılabilmesi için beş temel kriteri sağlaması gerektiğini belirtir:

1. **Girdi (Input):** Algoritmanın dışarıdan aldığı sıfır veya daha fazla iyi tanımlı veri.
2. **Çıktı (Output):** Algoritmanın ürettiği, girdiyle belirli bir matematiksel ilişki içinde olan en az bir veri.
3. **Belirlilik (Definiteness):** Her adım kesin, açık ve tek anlamlı olmalıdır. "Biraz bekle" veya "rastgele bir sayı seç" (hangi dağılımdan olduğu belirtilmedikçe) algoritma adımı olamaz.
4. **Sonluluk (Finiteness):** Algoritma her geçerli girdi için sonlu sayıda işlem adımı sonrasında durmalıdır. Sonsuza kadar çalışan bir döngü algoritma değildir.
5. **Etkinlik (Effectiveness):** Her adım, prensipte bir insanın eline bir kâğıt ve kalem alarak sonlu bir sürede kusursuzca yürütebileceği kadar temel olmalıdır.

Şimdi bu kavramları pekiştirmek için somuttan soyuta birkaç örnek inceleyelim.

**Örnek 20.3.** *Girdi olarak verilen iki pozitif $a$ ve $b$ tam sayısının toplamını hesaplamak istiyoruz. İlkokuldan beri bildiğimiz "eldenin aktarılması" yöntemi bir algoritma mıdır?*

*Çözüm.* Evet, insanlık tarihinin en eski ve temel algoritmalarından biridir.

- **Girdi:** $a$ ve $b$ sayılarının basamak dizilimleri.
- **Çıktı:** $c = a + b$ sayısının basamak dizilimi.
- **Süreç:**
  1. Sayıları birler basamağı aynı hizada olacak biçimde alt alta yaz. Elde değerini $e = 0$ olarak başlat.
  2. Sağdan sola doğru her sütundaki rakamları ve eldeki $e$ değerini topla: $T = basamak_1 + basamak_2 + e$.
  3. Yeni basamağı $T \bmod 10$ olarak sonuca yaz; yeni eldeyi $e = \lfloor T/10 \rfloor$ yap.
  4. Tüm basamaklar bittiğinde elde $e > 0$ kalmışsa onu en sola ekle ve dur.

Bu adımlar belirlidir, sonludur ve her sütunda temel aritmetik işlemler yapıldığı için etkindir. □

**Örnek 20.4.** *Bir torbanın içinde ağırlıkları birbirinden farklı $n$ adet taş bulunmaktadır ($n \geq 1$). Yalnızca iki taşı tartıp hangisinin daha ağır olduğunu söyleyebilen iki kefeli bir denge terazisi kullanarak en ağır taşı bulan bir algoritma tasarlayınız.*

*Çözüm.* Torbada tek bir taş varsa ($n = 1$), en ağır taş doğrudan odur. Birden fazla taş varsa, ilk taşı elimize "şu ana kadar gördüğümüz en ağır taş" adayı olarak alırız. Sonra torbadan teker teker diğer taşları çeker, elimizdeki adayla tartarız. Çektiğimiz taş daha ağır çıkarsa eski adayı kenara bırakıp yeni taşı aday yaparız; aksi halde yeni taşı kenara atarız.

Algoritmanın adımları şöyledir:

1. Taşları $T_1, T_2, \ldots, T_n$ olarak etiketle.
2. En ağır adayını belirle: $EnAgir \leftarrow T_1$.
3. $i = 2$'den $n$'ye kadar sırasıyla şu adımı tekrarla:
   - $T_i$ ile $EnAgir$ taşını terazide karşılaştır.
   - Eğer $T_i > EnAgir$ ise, güncellemeyi yap: $EnAgir \leftarrow T_i$.
4. Torbada taş kalmadığında $EnAgir$ taşını çıktı olarak ver ve dur.

Bu algoritma her $i$ adımında tam olarak 1 tartı yapar; toplamda $n - 1$ tartı ile kesin olarak sonuca ulaşır ve her zaman durur. □

**Örnek 20.5.** *Pozitif bir $n$ tam sayısı verildiğinde aşağıdaki kuralı uygulayalım:*

- *Eğer $n = 1$ ise DUR.*

- *Eğer $n$ çift ise, yeni sayı $n/2$ olsun.*
- *Eğer $n$ tek ise, yeni sayı $3n + 1$ olsun.*

*Bu işlem dizisi, verilen herhangi bir $n$ girdisi için bir algoritma mıdır?*

*Çözüm.* Bu ünlü kural, matematikte **Collatz Problemi** (veya $3n + 1$ problemi) olarak bilinir. Örneğin $n = 6$ ile başlarsak:

$$6 \to 3 \to 10 \to 5 \to 16 \to 8 \to 4 \to 2 \to 1 \quad \text{(Durur)}$$

Bugüne kadar bilgisayarlarla test edilen sayılar için bu dizi her zaman 1'e ulaşmış ve sonlanmıştır. Ancak günümüz matematiğinde, istisnasız her pozitif $n$ tam sayısı için bu dizinin sonlu adımda 1'e ulaşıp ulaşmadığı henüz **ispatlanamamıştır**.

Dolayısıyla, sonluluk (finiteness) şartının her girdi için sağlandığı matematiksel olarak kanıtlanamadığından, bu prosedürün genel anlamda bir algoritma olup olmadığı henüz kesinleşmiş değildir. Bir yöntemin sezgisel olarak çalışması yetmez; her geçerli girdi için duracağının garanti edilmesi gerekir. □

**Uyarı 3.** ***En sık yapılan hata:*** *Algoritmayı belirli bir programlama diliyle (C, Python, Java) özdeşleştirmektir. Programlama dilleri birer araçtır; C dili 1970'lerde ortaya çıkmıştır, fakat Bölüm 13'te ele aldığımız Öklid'in EBOB algoritması 2300 yaşındadır. Algoritma, dilin ve donanımın ötesinde saf bir matematiksel fikirdir.*

## 20.2 Sözde Kod (Pseudo-Code)

Bir algoritmayı başkasına aktarırken iki uç tuzakla karşılaşırız:

1. **Doğal dilin muğlaklığı:** Türkçe veya İngilizce serbest metin yazdığımızda cümleler yanlış anlamalara açıktır. "Sayının karekökünü al, duruma göre devam et" gibi ifadeler kesinlik taşımaz.
2. **Gerçek programlama dillerinin sözdizimi yükü:** Bir fikri hemen C veya C++ ile yazmaya kalktığımızda; noktalı virgüller, kütüphane eklemeleri (`#include`), bellek türleri (`int`, `long long`) ve derleyici ayrıntıları arasında algoritmanın asıl mantığı kaybolabilir.

Bu iki ucun dengesi **sözde kod** (pseudocode) kullanmaktır. Sözde kod, herhangi bir derleyicinin katı sözdizimi kurallarına bağlı olmayan, fakat mantıksal akışı (dallanma, döngü, atama) bir bilgisayar programı kadar net ifade eden yarı-biçimsel bir talimat dilidir.

Sözde kod yazarken kabul gören temel yapılar şunlardır:

- **Değer Atama:** Bir değişkene değer vermek için sola doğru bir ok ($\leftarrow$) veya `:=` simgesi kullanılır. Örneğin, $x \leftarrow 5$ ifadesi "$x$'in yeni değeri 5 olsun" demektir.
- **Koşul (Dallanma): EĞER** ... **İSE** ... **DEĞİLSE** blokları.
- **Döngüler:** Belirli bir koşul sağlandığı sürece tekrarlanan **İKEN**... **YAP** veya sınırları belirli **İÇİN**... **YAP** blokları.
- **Girdi ve Çıktı: GİRDİ** ve **DÖNDÜR** (veya **YAZDIR**) komutları.
- **Girintileme (Indentation):** Hangi adımların hangi bloğun içinde olduğunu göstermek için satırlar sağa kaydırılarak yazılır.

Şimdi sözde kodun kullanımını basit bir problemle görelim.

**Örnek 20.6** (İki Değişkenin Değerini Takas Etme / Swap)**.** *Elinizde iki değişken var: $a$ ve $b$. Başlangıçta $a = 5$ ve $b = 9$ olsun. Bu iki değişkenin değerlerini öyle değiştirin ki işlem sonunda $a = 9$ ve $b = 5$ olsun.*

*Çözüm.* İlk bakışta akla şu adımlar gelebilir:

$$a \leftarrow b$$
$$b \leftarrow a$$

Bu adımları tek tek yürütelim (trace edelim):

1. Başlangıç: $a = 5, b = 9$.
2. $a \leftarrow b$ adımı çalışır: $b$'nin değeri 9 olduğundan $a$'nın yeni değeri 9 olur. Artık hafızada $a = 9, b = 9$ vardır. Dikkat ederseniz $a$'nın eski değeri olan 5 kayboldu.
3. $b \leftarrow a$ adımı çalışır: $a$'nın değeri 9 olduğundan $b$'ye yine 9 atanır.
4. Sonuç: $a = 9, b = 9$. Takas başarısız oldu.

**Fiziksel Benzetme:** Sol elinizde bir bardak portakal suyu, sağ elinizde bir bardak elma suyu olduğunu varsayalım. Bardakları kırmadan ve sıvıları karıştırmadan suların yerini nasıl değiştirirsiniz? Üçüncü ve boş bir bardağa ihtiyaç vardır. Portakal suyunu boş bardağa dökünce ilk bardak boşa çıkar. Elma suyunu bu bardağa aktarırsınız. Son olarak üçüncü bardaktaki portakal suyunu sağ bardağa boşaltırsınız.

Algoritmada da aynı şekilde üçüncü bir **geçici değişken** (temporary variable, *gecici*) kullanırız: □

Bu fikrin sözde kodu şu şekildedir:

| **Algoritma:** DEGER_TAKAS$(a, b)$ | |
|---|---|
| **Girdi:** | İki değişken $a$ ve $b$ |
| **Çıktı:** | Değerleri yer değiştirmiş $a$ ve $b$ |
| 1: | $gecici \leftarrow a$ *(a'nın orijinal değeri yedeklenir)* |
| 2: | $a \leftarrow b$ *(b'nin değeri a'ya yazılır)* |
| 3: | $b \leftarrow gecici$ *(yedekteki eski a değeri b'ye aktarılır)* |
| 4: | **döndür** $(a, b)$ |

**Örnek 20.7** (Dizide En Büyük Elemanı Bulma)**.** *$n$ elemanlı bir $A = [A[1], A[2], \ldots, A[n]]$ tam sayı dizisinin en büyük elemanını bulan algoritmayı sözde kod ile yazınız.*

*Çözüm.* Adım adım kurgulayalım:

1. Girdi boş olamaz, $n \geq 1$ olduğunu varsayalım.
2. En büyük adayımızı dizinin ilk elemanı yapalım: $eb \leftarrow A[1]$.
3. 2'den $n$'ye kadar her bir $i$ indisi için $A[i]$'yi inceleyelim.
4. Eğer $A[i] > eb$ ise, yeni rekor $A[i]$'dir; dolayısıyla $eb \leftarrow A[i]$ güncellemesini yapalım.
5. Döngü bittiğinde $eb$ değerini döndürelim.

□

Bunu biçimsel sözde kod olarak yazalım:

| **Algoritma:** EN_BUYUK_BUL$(A, n)$ | |
|---|---|
| **Girdi:** | $n$ elemanlı $A[1 \ldots n]$ tam sayı dizisi $(n \geq 1)$ |
| **Çıktı:** | Dizideki en büyük tam sayı değeri |
| 1: | $eb \leftarrow A[1]$ |
| 2: | **için** $i \leftarrow 2$ **'den** $n$ **'ye kadar yap:** |
| 3: | **eğer** $A[i] > eb$ **ise:** |
| 4: | $eb \leftarrow A[i]$ |
| 5: | **döndür** $eb$ |

**Örnek 20.8** (Öklid Algoritması ile EBOB Hesabı)**.** *Verilen iki pozitif $a$ ve $b$ tam sayısının en büyük ortak bölenini ($\gcd(a, b)$) bulan Öklid algoritmasını sözde kod olarak yazınız ve $a = 48, b = 18$ için adım adım izleme tablosu (trace table) oluşturunuz.*

*Çözüm.* Bölüm 13'te ayrıntılarını gördüğümüz üzere Öklid'in temel gözlemi şudur: Eğer $a = q \cdot b + k$ (burada $k = a \bmod b$ kalandır) şeklinde yazılırsa, $a$ ile $b$'nin ortak bölenleri ile $b$ ile $k$'nin ortak bölenleri tamamen aynıdır. Dolayısıyla:

$$\gcd(a, b) = \gcd(b, a \bmod b)$$

Kalan 0 olduğunda bölen sayı aradığımız EBOB'dur. □

Sözde kod:

| **Algoritma:** OKLID_EBOB$(a, b)$ | |
|---|---|
| **Girdi:** | İki pozitif tam sayı $a$ ve $b$ |
| **Çıktı:** | $\gcd(a, b)$ |
| 1: | **iken** $b \neq 0$ **yap:** |
| 2: | $kalan \leftarrow a \bmod b$ |
| 3: | $a \leftarrow b$ |
| 4: | $b \leftarrow kalan$ |
| 5: | **döndür** $a$ |

Şimdi $a = 48$ ve $b = 18$ girdileri için algoritmanın bellek durumunu adım adım izleyelim:

| **Adım** | **Koşul** ($b \neq 0$) | $kalan = a \bmod b$ | Yeni $a$ | Yeni $b$ |
|---|---|---|---|---|
| Başlangıç | — | — | 48 | 18 |
| 1. Döngü | $18 \neq 0$ (Doğru) | $48 \bmod 18 = 12$ | 18 | 12 |
| 2. Döngü | $12 \neq 0$ (Doğru) | $18 \bmod 12 = 6$ | 12 | 6 |
| 3. Döngü | $6 \neq 0$ (Doğru) | $12 \bmod 6 = 0$ | 6 | 0 |
| Döngü Sonu | $0 \neq 0$ (Yanlış) | Algoritma çıkar | **Sonuç: 6** | 0 |

Sonuç olarak $\gcd(48, 18) = 6$ doğru biçimde bulunmuştur.

**Uyarı 4.** ***En sık yapılan hata:*** *Sözde kod yazarken "sihirli adımlar" atmaktır. Örneğin, sözde kodun içine tek bir satırda "A dizisindeki en büyük elemanı bul" veya "n sayısının asal çarpanlarını ayır" yazıp geçmek çözümü gizlemektir. Sözde kod, o işlemin mantıksal alt adımlarını (döngü ve karşılaştırmaları) net biçimde sergilemelidir.*

## 20.3 Değişken, Koşul ve Döngü

Bir algoritmanın hesaplama yapabilmesi için üç temel yapı taşına ihtiyacı vardır:

1. Bilgiyi saklamak (**Değişken**),
2. Duruma göre yol ayrımına gitmek (**Koşul / Dallanma**),
3. Adımları tekrar etmek (**Döngü**).

Bilgisayar biliminde Böhm-Jacopini teoremi gereği, Turing-tamı olan herhangi bir hesaplama yalnızca bu üç yapının (sıralı işlem, koşullu dallanma, koşullu döngü) birleşimiyle ifade edilebilir.

### 20.3.1 Değişkenler ve Atama Mantığı

Matematikte $x = x + 1$ ifadesi bir çelişkidir; hiçbir gerçel sayı kendisinin bir fazlasına eşit olamaz. Ancak bilgisayar biliminde:

$$x \leftarrow x + 1$$

ifadesi bir denklik veya eşitlik değil, bir **eylemdir (atama işlemi)**.

**Tanım 20.9** (Değişken ve Atama)**.** *Bir **değişken**, bilgisayar belleğinde belirli bir değere ev sahipliği yapan isimlendirilmiş bir saklama alanıdır (kutu).*

*Değişken ← İfade atama işlemi iki aşamada gerçekleşir:*

1. *Sağ taraftaki ifadenin anlık değeri hesaplanır.*
2. *Elde edilen sonuç, sol taraftaki değişkenin bellekteki kutusuna yazılır (eski değer silinir).*

Dolayısıyla $x \leftarrow x + 1$ demek, "$x$'in kutusundaki mevcut sayıyı oku, 1 ekle, çıkan sonucu tekrar o kutunun içine koy" demektir.

### 20.3.2 Koşullar (Dallanma)

Algoritmanın akışı tek bir doğru çizgi üzerinde gitmek zorunda değildir; verinin durumuna göre farklı yollar seçilmelidir.

- **eğer** koşul **ise:** Koşul doğruysa belirtilen işlemleri yap.
- **değilse:** Koşul yanlışsa bu alternatif işlemleri yap.

### 20.3.3 Döngüler ve Döngü Değişmezi (Loop Invariant)

Döngüler, belirli bir işlem öbeğini tekrar tekrar çalıştırmamızı sağlar. İki temel döngü çeşidi vardır:

1. **Koşullu Döngü (İken / While):** Önceden kaç kez döneceğini tam bilmediğimiz, bir koşul bozulana kadar süren döngüler.
2. **Sayaçlı Döngü (İçin / For):** Bir sayacın belirli bir başlangıç değerinden bitiş değerine kadar adım adım ilerlediği döngüler.

Bölüm 18'de incelediğimiz invaryant (değişmez) ilkesi döngülerin analizinde de karşımıza çıkar. Bir döngünün doğruluğunu garanti eden bu mantıksal araca **döngü değişmezi** (loop invariant) denir.

**Tanım 20.10** (Döngü Değişmezi)**.** *Döngü değişmezi, döngünün her yinelenmesinden (iterasyonundan) önce ve sonra doğru kalan mantıksal bir önermedir.*

Döngü değişmezi yöntemi, Bölüm 2'de gördüğümüz tümevarım ilkesiyle doğrudan örtüşür ve üç aşamada incelenir:

1. **Başlangıç (Temel Adım):** Döngü ilk kez başlamadan önce önerme doğrudur.
2. **Sürdürme (Tümevarım Adımı):** Döngünün bir adımının başında önerme doğruysa, o adım tamamlandığında da önerme doğru kalmaya devam eder.
3. **Sonlanma:** Döngü bittiğinde, korunan bu değişmez önerme bize aradığımız problemin doğru çözümünü verir.

Şimdi bu yapıları somut problemlerle inceleyelim.

**Örnek 20.11** (Toplam Hesabı ve Döngü Değişmezi)**.** *1'den n'ye kadar olan tam sayıların toplamını hesaplayan bir döngü tasarlayınız ve doğruluğunu döngü değişmezi kullanarak ispatlayınız.*

*Çözüm.* Bir prefix sum değişkeni $T$ tutalım. Başlangıçta $T = 0$ olsun. Bir sayaç $i = 1$'den $n$'ye kadar her adımda $T$'ye eklensin.

Sözde kod:

| **Algoritma:** TOPLAM($n$) | |
|---|---|
| 1: | $T \leftarrow 0$ |
| 2: | $i \leftarrow 1$ |
| 3: | **iken** $i \leq n$ **yap:** |
| 4: | $\quad T \leftarrow T + i$ |
| 5: | $\quad i \leftarrow i + 1$ |
| 6: | **döndür** $T$ |

**Doğruluk İspatı (Döngü Değişmezi):** Döngünün her adımının başında şu iddiada bulunalım: **Değişmez (Invariant):** Döngü koşulu kontrol edilirken $T = \sum_{k=1}^{i-1} k$ eşitliği geçerlidir.

- **Başlangıç:** Döngü başlamadan önce $i = 1$ ve $T = 0$'dır. Boş toplam tanımdan 0 olduğundan $\sum_{k=1}^{0} k = 0$'dır. Eşitlik $0 = 0$ olup başlangıçta doğrudur.
- **Sürdürme:** Herhangi bir adımın başında $T = \sum_{k=1}^{i-1} k$ doğru olsun. Döngü gövdesi çalıştığında: Yeni toplam:

$$T_{yeni} = T_{eski} + i = \left( \sum_{k=1}^{i-1} k \right) + i = \sum_{k=1}^{i} k$$

Ardından sayaç $i_{yeni} = i + 1$ yapılır. Böylece bir sonraki adımın başında:

$$T_{yeni} = \sum_{k=1}^{i_{yeni}-1} k$$

olur. Değişmez bir sonraki iterasyona aynen korunarak aktarılmıştır.

- **Sonlanma:** Döngü $i = n + 1$ olduğu anda sonlanır ($i \leq n$ bozulur). Değişmezimizde $i$ yerine $n + 1$ yazarsak:

$$T = \sum_{k=1}^{(n+1)-1} k = \sum_{k=1}^{n} k$$

elde edilir. Döngü durduğunda $T$ tam olarak 1'den $n$'ye kadar olan sayıların toplamına eşittir.

□

**Örnek 20.12** (Basamaklar Toplamı)**.** *Pozitif bir $n$ tam sayısının onluk tabandaki basamaklarının toplamını hesaplayan bir algoritma yazınız.*

*Çözüm.* Fikri somut bir sayıyla görelim: $n = 374$. Sayının birler basamağını koparmak için 10'a bölerek kalana bakarız: $374 \bmod 10 = 4$. Bu basamağı toplama ekleriz. Basamağı sayıdan ayırmak için sayıyı 10'a böleriz: $\lfloor 374/10 \rfloor = 37$. Aynı işlemi tekrarlarız:

- $37 \bmod 10 = 7$ (toplama ekle), $\lfloor 37/10 \rfloor = 3$.
- $3 \bmod 10 = 3$ (toplama ekle), $\lfloor 3/10 \rfloor = 0$.
- Sayı 0 olduğunda döngü biter.

Toplam: $4 + 7 + 3 = 14$. □

Sözde kod:

| **Algoritma:** BASAMAK_TOPLAMI($n$) | |
|---|---|
| **Girdi:** | Pozitif tam sayı $n$ |
| **Çıktı:** | $n$'nin basamakları toplamı |
| 1: | $toplam \leftarrow 0$ |
| 2: | **iken** $n > 0$ **yap:** |
| 3: | $\quad basamak \leftarrow n \bmod 10$ |
| 4: | $\quad toplam \leftarrow toplam + basamak$ |
| 5: | $\quad n \leftarrow \lfloor n/10 \rfloor$ |
| 6: | **döndür** $toplam$ |

**Örnek 20.13** (Fibonacci Dizisi Hesabı)**.** *Bölüm 9'da yineleme bağıntıları bağlamında incelediğimiz Fibonacci dizisi $F_0 = 0$, $F_1 = 1$ ve her $n \geq 2$ için $F_n = F_{n-1} + F_{n-2}$ şeklinde tanımlanır. Verilen bir $n$ tam sayısı ($n \geq 2$) için $F_n$'i hesaplayan bir algoritma tasarlayınız.*

*Çözüm.* Fibonacci dizisinde bir sonraki terimi bulmak için daima son iki değer yeterlidir. Eski terimleri $a$ ve $b$ değişkenlerinde tutalım: başlangıçta $a = F_0 = 0$, $b = F_1 = 1$. Bir sonraki terim $yeni = a + b$ olur. Ardından bir adım sağa kayarız: $a$'nın yerine eski $b$ geçer, $b$'nin yerine ise az önce hesaplanan $yeni$ değeri atanır. Bu kaydırma işlemini $n - 1$ defa tekrarladığımızda $b$ değişkeni tam olarak $F_n$'e eşit olur. □

Sözde kod:

| **Algoritma:** FIBONACCI($n$) | |
|---|---|
| **Girdi:** | Pozitif tam sayı $n$ ($n \geq 1$) |
| **Çıktı:** | $n$. Fibonacci sayısı $F_n$ |
| 1: | **eğer** $n = 1$ **ise döndür** 1 |
| 2: | $a \leftarrow 0$ |
| 3: | $b \leftarrow 1$ |
| 4: | **için** $i \leftarrow 2$ **'den** $n$ **'ye kadar yap:** |
| 5: | $\quad yeni \leftarrow a + b$ |
| 6: | $\quad a \leftarrow b$ |
| 7: | $\quad b \leftarrow yeni$ |
| 8: | **döndür** $b$ |

**Örnek 20.14** (Sadece Çıkarma Kullanarak Bölme ve Kalan Bulma)**.** *Çarpma ve bölme operatörlerini kullanmadan, yalnızca çıkarma ve toplama ile pozitif iki tam sayı $A$ ve $B$ için $A$'nın $B$'ye bölümünden elde edilen bölüm ($Q$) ve kalan ($R$) değerlerini bulan algoritmayı tasarlayınız.*

*Çözüm.* Bölüm 11'de ele aldığımız bölme mantığını anımsarsak; bölme işlemi esasen art arda yapılan bir çıkarma işlemidir. $A$'dan $B$'yi kaç kez çıkarabiliyorsak bölüm odur; geriye çıkaramayacağımız kadar küçük bir miktar kaldığında o miktar kalandır. Örneğin $A = 14, B = 3$:

- $14 - 3 = 11$ (1. çıkarma)
- $11 - 3 = 8$ (2. çıkarma)
- $8 - 3 = 5$ (3. çıkarma)
- $5 - 3 = 2$ (4. çıkarma)
- $2 < 3$ olduğundan daha fazla çıkarma yapılamaz.
- Çıkarma sayısı: 4 (Bölüm $Q = 4$), Kalan: 2 ($R = 2$).

□

Sözde kod:

| **Algoritma:** TAM_BOLME$(A, B)$ | |
|---|---|
| **Girdi:** | Pozitif tam sayılar $A$ ve $B$ |
| **Çıktı:** | Bölüm $Q$ ve Kalan $R$ ($A = Q \cdot B + R$ ve $0 \leq R < B$) |
| 1: | $Q \leftarrow 0$ |
| 2: | $R \leftarrow A$ |
| 3: | **iken** $R \geq B$ **yap:** |
| 4: | $\quad R \leftarrow R - B$ |
| 5: | $\quad Q \leftarrow Q + 1$ |
| 6: | **döndür** $(Q, R)$ |

Burada döngü değişmezimiz şudur: Döngünün her adımında $A = Q \cdot B + R$ bağıntısı tam olarak korunur. Döngü bittiğinde ise $R < B$ şartı sağlanır; bu da bölme algoritmasının tekliğini kanıtlar.

### 20.3.4 Sözde Koddan Gerçek Koda Geçiş

Tasarladığımız algoritmaları bilgisayara işletmek için bir programlama diline dökeriz. Yukarıda tasarladığımız Öklid EBOB algoritmasının C dilindeki karşılığına ve ardından sözde kod biçimine bakalım:

```c
// C Dili ile EBOB Algoritmasi
#include <stdio.h>

int ebob(int a, int b) {
    while (b != 0) {
        int kalan = a % b;
        a = b;
        b = kalan;
    }
    return a;
}

int main() {
    int x = 48, y = 18;
    printf("EBOB(%d, %d) = %d\n", x, y, ebob(x, y));
    return 0;
}
```

Aynı algoritmanın sözde kod karşılığı ise şöyledir:

```
FUNCTION gcd(a, b)
    WHILE b != 0 DO
        remainder = a MOD b
        a = b
        b = remainder
    END WHILE
    RETURN a
END FUNCTION
```

```
x = 48
y = 18
PRINT "GCD(" + x + ", " + y + ") = " + gcd(x, y)
```

Görüldüğü üzere mantık ve algoritma tamamen aynıdır; yalnızca dillerin yazım kuralları (sözdizimi) farklılık gösterir.

**Uyarı 5.** ***En sık yapılan hata (Off-by-one Hatası):*** *Döngü sınırlarında "bir eksik ya da bir fazla" adım atmaktır. Örneğin $n$ elemanlı bir diziyi gezerken sayacı $0$'dan mı başlatmalıyız yoksa $1$'den mi? Şartımız $i < n$ mi olmalıdır yoksa $i \leq n$ mi? C dilinde diziler $0$'dan başladığı için sınırlar $0 \leq i < n$ olurken, matematikte ve sözde kodda genellikle $1 \leq i \leq n$ tercih edilir. Bu sınırları birbirine karıştırmak kodlama ve algoritma sorularında en yaygın mantık hatalarındandır.*

# Bölüm 21

# Karmaşıklık ve Big O

Yazdığımız bir programın doğru çalışması tek başına yeterli değildir; programın verilen zaman ve bellek sınırları içinde de sonuç üretmesi gerekir. TÜBİTAK 1. Aşama sınavında ve programlama yarışmalarında problemler genellikle katı süre sınırlarına (çoğunlukla 1 saniye) ve bellek sınırlarına (örneğin 256 MB) sahiptir. Aynı problemi çözen iki farklı algoritmadan biri sonucu saniyenin binde birinde bulurken, diğeri kabul edilebilir sürelerin çok ötesinde çalışabilir.

Algoritmaların çalışma hızını ve bellek tüketimini; donanımdan, işletim sisteminden ya da programlama dilinden bağımsız biçimde ölçmek temel bir gerekliliktir. Bu doğrultuda önce temel işlem adımlarını saymayı, ardından bu davranışı matematiksel olarak ifade eden Big O notasyonunu ve yarışmalardaki pratik sınırları ele alacağız.

## 21.1 İşlem Sayısı

Bir algoritmanın ne kadar hızlı çalıştığını saniye cinsinden ölçmek ilk başta mantıklı görünebilir. Ancak aynı programı eski bir dizüstü bilgisayarda çalıştırdığımızda 5 saniye, modern bir sunucuda çalıştırdığımızda ise 0.5 saniye sürebilir. Dahası, arka planda müzik çalması veya işletim sisteminin bir güncelleme yapması bile ölçülen süreyi değiştirebilir.

Bize donanımdan bağımsız, evrensel bir ölçü gerekir: **Algoritmanın girdi boyutuna bağlı olarak yürüttüğü temel işlem sayısı**.

### 21.1.1 Temel İşlem Nedir?

Bir bilgisayarın işlemcisinin sabit sürede gerçekleştirebildiği en küçük adımlara *temel işlem* diyoruz. Bunlar:

- İki sayıyı toplamak, çıkarmak, çarpmak veya bölmek ($a + b$, $x \times y$),
- İki değeri karşılaştırmak ($a < b$, $x == y$),

- Bir değişkene değer atamak ($x \leftarrow 5$),
- Dizinin belirli bir indisine erişmek ($A[i]$).

Şimdi somut bir problem üzerinden adım sayısını tam olarak hesaplayalım.

**Örnek 21.1.** *$n$ elemanlı bir tam sayı dizisindeki en büyük elemanı bulmak istiyoruz. Aşağıdaki algoritmanın toplam işlem sayısını $n$ cinsinden belirleyelim.*

```
int en_buyuk = A[0];
for (int i = 1; i < n; i++) {
    if (A[i] > en_buyuk) {
        en_buyuk = A[i];
    }
}
```

*Çözüm.* Adımları tek tek sayalım:

1. `int en_buyuk = A[0];`: 1 dizi erişimi ve 1 atama (2 işlem).
2. Döngü başlangıcı: `int i = 1;` (1 atama).
3. Döngü koşulu: `i < n` kontrolü her adımda yapılır. Döngü $n-1$ kez döner, son kontrolle birlikte $n$ kez karşılaştırma yapılır.
4. İndis artırımı: `i++` işlemi $n-1$ kez yapılır.
5. Karşılaştırma: `A[i] > en_buyuk` koşulu her döngü adımında 1 dizi erişimi ve 1 karşılaştırma olmak üzere 2 işlem gerektirir, toplam $2(n-1)$ işlem.
6. Güncelleme: `en_buyuk = A[i]` satırı dizinin içeriğine bağlı olarak en iyi durumda 0 kez, en kötü durumda (dizi kesin artan ise) $n-1$ kez çalışır. Her çalışmada 1 dizi erişimi ve 1 atama (2 işlem) vardır.

En kötü durumda (worst-case) toplam işlem sayısı:

$$T(n) = 2 + 1 + n + (n-1) + 2(n-1) + 2(n-1) = 6n - 2$$

En iyi durumda (best-case, ilk eleman en büyükse):

$$T(n) = 2 + 1 + n + (n-1) + 2(n-1) + 0 = 4n$$

Görüldüğü üzere girdi boyutu $n$ iki katına çıktığında, yürütülen işlem sayısı da yaklaşık iki katına çıkmaktadır. Katsayılar (4 veya 6) donanıma veya derleyici optimizasyonuna göre değişebilir; ancak büyümenin $n$ ile *doğrusal* olduğu gerçeği sabittir. □

### 21.1.2 İç İçe Döngüler ve Büyüme Hızı

Şimdi iki farklı algoritmayı karşılaştıralım. Amacımız 1'den $n$'e kadar olan sayıların toplamını hesaplamak olsun.

**1. Yöntem (Tek Döngü):**

```
long long toplam = 0;
for (int i = 1; i <= n; i++) {
    toplam += i;
}
```

Burada döngü $n$ defa döner. Her adımda 1 toplama ve 1 atama yapılır. Yaklaşık işlem sayısı $c_1 \cdot n$ kadardır (burada $c_1$ küçük bir sabittir).

**2. Yöntem (Matematiksel Formül):**

```
long long toplam = (long long)n * (n + 1) / 2;
```

Burada $n = 10$ olsa da, $n = 10^9$ olsa da sadece 1 toplama, 1 çarpma ve 1 bölme yapılır. İşlem sayısı girdi boyutundan tamamen bağımsızdır; sabit bir $c_2$ sayısıdır.

**3. Yöntem (Gereksiz İç İçe Döngü):** Aynı toplamı, $i$ sayısını 1'den $i$'ye kadar 1'leri toplayarak hesaplayan yapay bir algoritma düşünelim:

```
long long toplam = 0;
for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= i; j++) {
        toplam += 1;
    }
}
```

İçteki döngü $i = 1$ iken 1 kez, $i = 2$ iken 2 kez, ..., $i = n$ iken $n$ kez çalışır. Toplam adım sayısı:

$$1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2} = \frac{1}{2}n^2 + \frac{1}{2}n$$

Girdi boyutu büyüdükçe bu üç algoritmanın adım sayılarını kıyaslayalım:

| $n$ | 2. Yöntem ($c_2$) | 1. Yöntem ($2n$) | 3. Yöntem ($\frac{1}{2}n^2 + \frac{1}{2}n$) |
|---:|---:|---:|---:|
| 10 | 3 | 20 | 55 |
| 1 000 | 3 | 2 000 | 500 500 |
| 100 000 | 3 | 200 000 | 5 000 050 000 ($5 \times 10^9$) |
| 1 000 000 | 3 | 2 000 000 | 500 000 500 000 ($5 \times 10^{11}$) |

$n = 100\,000$ olduğunda 1. yöntem modern bir bilgisayarda birkaç milisaniyede biterken, 3. yöntem saniyeler sürer. $n = 1\,000\,000$ olduğunda 3. yöntemin bitmesi dakikalar alacaktır.

Buradan çıkaracağımız temel sonuç şudur: **Girdi $n$ büyüdükçe, fonksiyonun en yüksek dereceli terimi tüm davranışı belirler.** $T(n) = \frac{1}{2}n^2 + \frac{1}{2}n$

ifadesinde $n = 10^6$ için $\frac{1}{2}n^2 = 5 \times 10^{11}$ iken, $\frac{1}{2}n = 5 \times 10^5$'tir. Küçük dereceli terim, toplamın yanında ihmal edilebilir düzeydedir. Benzer biçimde öndeki $\frac{1}{2}$ katsayısı da $n^2$'nin büyüme hızının yanında belirleyici rolünü kaybeder.

Bu gözlem bizi algoritmik karmaşıklığın standart diline götürür: Big O notasyonu.

## 21.2 Big O Notasyonu

### 21.2.1 Motivasyon ve Biçimsel Tanım

Bir algoritmanın işlem sayısı tam olarak $T(n) = 3n^2 + 14n + 120$ olsun. Başka bir algoritmanınki ise $T'(n) = n^2 + 200n + 5$ olsun. İkisi de temelde $n^2$ ile ölçeklenir; yani girdi 10 katına çıktığında işlem sayısı kabaca 100 katına çıkar. Analiz yaparken ayrıntılardaki $14n$ veya sabit 120 gibi küçük terimlerle ve baştaki katsayılarla uğraşmak istemeyiz. Algoritmanın *asemptotik* (yani $n$ sonsuza giderkenki) üst sınırını belirlemek isteriz.

**Tanım 21.2** (Big O (Büyük O))**.** *$f(n)$ ve $g(n)$, pozitif tam sayılardan pozitif reel sayılara tanımlı iki fonksiyon olsun. Eğer her $n \geq n_0$ için*

$$f(n) \leq c \cdot g(n)$$

*olacak biçimde pozitif reel bir $c$ sabiti ve pozitif bir $n_0$ tam sayısı varsa,*

$$f(n) = O(g(n))$$

*yazılır ve "$f(n)$, $g(n)$ mertebesindedir" denir.*

Bu tanım şunu söyler: Yeterince büyük $n$ değerleri için ($n \geq n_0$), $f(n)$ fonksiyonu $g(n)$'in sabit bir katını asla geçemez. $g(n)$, $f(n)$ için bir **asemptotik üst sınırdır**.

**Teorem 21.3.** *$P(n) = a_k n^k + a_{k-1} n^{k-1} + \cdots + a_1 n + a_0$ derecesi $k$ olan bir polinom ve $a_k > 0$ olsun. Bu durumda $P(n) = O(n^k)$'dir.*

*Kanıt.* Her $n \geq 1$ için üçgen eşitsizliğini kullanarak:

$$|P(n)| \leq |a_k| n^k + |a_{k-1}| n^{k-1} + \cdots + |a_0|$$

$n \geq 1$ olduğundan her $i \leq k$ için $n^i \leq n^k$ yazabiliriz. Dolayısıyla:

$$|P(n)| \leq (|a_k| + |a_{k-1}| + \cdots + |a_0|) n^k$$

Burada $c = |a_k| + |a_{k-1}| + \cdots + |a_0|$ ve $n_0 = 1$ seçilirse tanımdaki eşitsizlik sağlanır. Demek ki $P(n) = O(n^k)$'dir. □

Bu teorem sayesinde karmaşık polinomlarla tek tek uğraşmak gerekmez: En büyük dereceli terimi alır, katsayısını 1 yaparız. Örneğin:

$$5n^3 + 100n^2 - 7n + 42 = O(n^3)$$

### 21.2.2 Temel Karmaşıklık Sınıfları ve Sezgileri

Olimpiyatlarda en sık karşılaşacağımız karmaşıklık sınıflarını küçükten büyüğe sıralayalım:

$$O(1) < O(\log n) < O(\sqrt{n}) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n) < O(n!)$$

Şimdi bu sınıfların sezgisel anlamlarını inceleyelim.

#### 1. Sabit Zaman: $O(1)$

İşlem sayısı girdinin büyüklüğünden bağımsızdır. Girdi ister 5 ister 5 milyar olsun, işlem sabit sayıda adımda tamamlanır.

- Bir dizinin $k$. elemanına erişmek: `A[k]`
- İki sayının tek mi çift mi olduğunu kontrol etmek: `x % 2 == 0`
- Aritmetik bir formülü hesaplamak: `n * (n + 1) / 2`

#### 2. Logaritmik Zaman: $O(\log n)$

Bir algoritma her adımda arama uzayını veya girdiyi sabit bir oranda (genellikle yarı yarıya) küçültüyorsa, o algoritmanın karmaşıklığı logaritmiktir.

Örnek olarak bir sayıyı sürekli 2'ye böldüğümüzü düşünelim:

```
int sayac = 0;
while (n > 1) {
    n = n / 2;
    sayac++;
}
```

$n = 16$ için değerler: $16 \to 8 \to 4 \to 2 \to 1$ (4 adım). Genel olarak $2^k \leq n$ ise döngü yaklaşık $\lfloor \log_2 n \rfloor$ adım sürer. Big O gösteriminde logaritmanın tabanı yazılmaz; çünkü taban değişimi yalnızca sabit bir çarpan getirir: $\log_a n = \frac{\log_b n}{\log_b a} = c \cdot \log_b n$. Dolayısıyla $O(\log_2 n) = O(\log_{10} n) = O(\log n)$'dir.

$O(\log n)$ algoritmaları son derece yavaş büyür. $n = 10^{18}$ gibi çok büyük bir girdi için bile $\log_2(10^{18}) \approx 60$ adımdır.

#### 3. Doğrusal Zaman: $O(n)$

Girdideki her elemana sabit sayıda (örneğin 1 veya 2 kez) bakılıyorsa algoritma doğrusaldır.

- $n$ elemanlı bir dizinin toplamını bulmak.
- Sırasız bir dizide belirli bir değeri aramak (lineer arama).

**4. Doğrusal-Logaritmik Zaman:** $O(n \log n)$

Genellikle bir problemi logaritmik derinlikte parçalara ayırıp her seviyede $O(n)$ iş yapan "böl ve yönet" algoritmalarında karşımıza çıkar. Yaygın sıralama algoritmaları (Merge Sort, Hızlı Sıralama) bu sınıftadır. $n = 10^6$ için $n \log_2 n \approx 2 \times 10^7$ işlem demektir ve bu olimpik süre sınırları için oldukça güvenli bir çalışma hacmidir.

**5. Karesel Zaman:** $O(n^2)$

Genellikle iç içe iki döngünün tüm çiftleri taradığı durumlarda ortaya çıkar.

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        // Sabit sayida islem O(1)
    }
}
```

Tüm $(i, j)$ ikililerinin sayısı $n \times n = n^2$'dir. $n = 1\,000$ için $n^2 = 10^6$ (hızlı), fakat $n = 100\,000$ için $n^2 = 10^{10}$ olur ve süre sınırını aşar.

**6. Üstel ve Faktöriyel Zaman:** $O(2^n)$ **ve** $O(n!)$

Bölüm 4 ve 5'te gördüğümüz sayma kurallarıyla bağlantılı olarak, bir kümenin tüm alt kümelerini gezmek $2^n$, tüm permütasyonlarını gezmek ise $n!$ adım gerektirir. Bu algoritmalar yalnızca çok küçük $n$ değerleri ($n \leq 20$ veya $n \leq 11$) için pratik sürede çalışabilir.

### 21.2.3 Çözümlü Örnekler: Karmaşıklık Nasıl Hesaplanır?

**Örnek 21.4.** *Aşağıdaki kod parçasının zaman karmaşıklığını Big O cinsinden bulunuz.*

```
for (int i = 1; i <= n; i *= 2) {
    for (int j = 1; j <= n; j++) {
        toplam++;
    }
}
```

*Çözüm.* İç ve dış döngülerin adım sayılarını birbirinden bağımsız biçimde inceleyebiliriz:

- Dıştaki döngü: $i$ değişkeni $1, 2, 4, 8, \ldots$ şeklinde ikiye katlanarak ilerliyor. Dolayısıyla $i \leq n$ koşulu sağlandığı sürece döngü adımı sayısı $k$, $2^{k-1} \leq n$ eşitsizliğine uyar. Yani dış döngü tam olarak $\lfloor \log_2 n \rfloor + 1$ kez döner.
- İçteki döngü: Dış döngünün her bir adımında, $j$ değişkeni 1'den $n$'e kadar birer birer artar. Yani iç döngü her zaman $n$ kez çalışır.

Toplam işlem sayısı iç ve dış döngü adımlarının çarpımıdır:

$$(\lfloor \log_2 n \rfloor + 1) \times n = O(n \log n)$$

□

**Örnek 21.5.** *Aşağıdaki kod parçasının zaman karmaşıklığını belirleyiniz.*

```
for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= n; j += i) {
        toplam++;
    }
}
```

*Çözüm.* İçteki döngünün adım sayısı $i$'ye bağlı olduğundan sınırları doğrudan çarpmak yerine her $i$ değeri için adımları tek tek toplayalım:

- $i = 1$ için: $j$ birer birer artar, döngü $\frac{n}{1}$ kez çalışır.
- $i = 2$ için: $j$ ikişer ikişer artar, döngü $\frac{n}{2}$ kez çalışır.
- $i = 3$ için: $j$ üçer üçer artar, döngü $\frac{n}{3}$ kez çalışır.
- ...
- $i = n$ için: $j$ $n$'er artar, döngü $\frac{n}{n} = 1$ kez çalışır.

Toplam adım sayısı:

$$S = \frac{n}{1} + \frac{n}{2} + \frac{n}{3} + \cdots + \frac{n}{n} = n\left(1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}\right)$$

Parantez içindeki ifade harmonik seridir ($H_n$). Bilindiği üzere $H_n = \ln n + \gamma + O(1/n)$ olup büyüme hızı $O(\log n)$'dir.

Dolayısıyla toplam adım sayısı:

$$S = n \cdot O(\log n) = O(n \log n)$$

Bu analiz, Bölüm 12'de göreceğimiz Eratosthenes kalburunun karmaşıklık hesabının da temelini oluşturur. □

**Örnek 21.6.** *Aşağıdaki fonksiyonun zaman karmaşıklığını bulunuz.*

```
void islem(int n) {
    while (n > 0) {
        for (int i = 0; i < n; i++) {
            printf("*");
        }
        n /= 2;
    }
}
```

*Çözüm.* İlk bakışta iç içe bir yapı görüp doğrudan $O(n \log n)$ demek sık yapılan bir hatadır. Adımları toplayarak görelim:

- 1. adımda $n$ yıldız basılır.
- 2. adımda $n/2$ yıldız basılır.
- 3. adımda $n/4$ yıldız basılır.
- ...

Toplam işlem sayısı:

$$T(n) = n + \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \cdots + 1$$

$n$ parantezine alırsak:

$$T(n) = n\left(1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots\right)$$

Sonsuz geometrik serinin toplamı $\sum_{i=0}^{\infty} \left(\frac{1}{2}\right)^i = \frac{1}{1-1/2} = 2$'dir. Dolayısıyla:

$$T(n) < 2n$$

Yani toplam işlem sayısı $2n$'i asla geçemez. O halde bu algoritmanın karmaşıklığı $O(n \log n)$ değil, $O(n)$'dir. □

**Örnek 21.7** (Döngü Gövdesindeki Gizli Maliyetler)**.** *Aşağıdaki kod parçasının karmaşıklığını inceleyelim:*

```
result = empty list
FOR EACH element IN A DO
    IF element NOT IN result THEN
        APPEND element TO result
    END IF
END FOR
```

*Çözüm.* Tek bir döngü görüldüğünde kodun $O(n)$ olduğunu varsaymak sık yapılan bir yanılgıdır.

İçerideki arama adımı (`element NOT IN result`), bir listenin içinde baştan sona eleman arar. `result` listesinde $k$ eleman varken bu arama işlemi $O(k)$ sürer. Dizi tekil elemanlardan oluşuyorsa $k$, 0'dan $n-1$'e kadar büyür. Toplam maliyet:

$$0 + 1 + 2 + \cdots + (n-1) = \frac{n(n-1)}{2} = O(n^2)$$

Programlama dillerinin yerleşik fonksiyonlarını (örneğin dizide arama, dilimleme ya da kopyalama) kullanırken o işlemlerin arka plandaki maliyetini mutlaka hesaba katmalıyız. □

## 21.3 Pratik Limitler

Teorik Big O analizini somut yarışma koşullarına uyarlayalım. Karşımızda girdi boyutu $n$ olarak verilmiş ve süre sınırı 1.0 saniye olan bir problem olduğunda, hangi karmaşıklıktaki bir algoritmayı tasarlamalıyız?

### 21.3.1 Temel Eşik: $10^8$ İşlem / Saniye

**Tanım 21.8** ($10^8$ Kuralı)**.** *Modern yarışma sunucularında (Codeforces, TÜBİTAK sınav sistemi, AtCoder vb.) standart bir C/C++ programı* ***1 saniyede yaklaşık*** $10^8$ ***(100 milyon) basit işlem*** *gerçekleştirir.*

Bu kural mutlak bir sınır olmamakla birlikte yürütülen işlemlerin türüne göre esneklik gösterir:

- Basit toplama, çıkarma, bit işlemleri (bitwise) ve dizi erişimleri çok hızlıdır; 1 saniyede $2 \times 10^8$ hatta $4 \times 10^8$ adet çalışabilir.
- Çarpma nispeten hızlıdır; bölme (`/`) ve mod alma (`%`) işlemleri ise işlemci için daha maliyetli olup 1 saniyede ancak $5 \times 10^7$ civarında yürütülebilir.
- Kayan noktalı sayı fonksiyonları (`sqrt`, `sin`, `pow`) belirgin biçimde daha yavaştır.

Bununla birlikte, kaba bir ön tahmin için $10^8$ eşiği en güvenilir başlangıç noktasıdır.

### 21.3.2 Girdi Boyutuna ($n$) Göre Algoritma Tahmini

TÜBİTAK 1. Aşama sınavında problem metnindeki $n$ sınırlarına bakarak hedeflenen algoritmanın karmaşıklığı doğrudan tahmin edilebilir. Aşağıdaki tablo bu kararda temel rehberdir:

| **Girdi Boyutu ($n$)** | **Beklenen Karmaşıklık** | **Tipik Algoritma / Yaklaşım** |
|---|---|---|
| $n \leq 10$ | $O(n!)$ | Tüm permütasyonları deneme |
| $n \leq 20$ | $O(2^n)$ veya $O(2^n \cdot n)$ | Tüm alt kümeleri gezme, bitmask DP |
| $n \leq 500$ | $O(n^3)$ | Floyd-Warshall, 3 iç içe döngü |
| $n \leq 5\,000$ | $O(n^2)$ | Kaba kuvvet ikilileri tarama, basit DP |
| $n \leq 200\,000$ | $O(n \log n)$ veya $O(n)$ | Sıralama, ikili arama, segment ağacı |
| $n \leq 10^7$ | $O(n)$ | Basit tek geçişli döngü, kalbur |
| $n > 10^9$ | $O(\log n)$ veya $O(1)$ | Matematiksel formül, hızlı üs alma |

**Örnek 21.9.** *Bir problemde $n = 100\,000$ tam sayı veriliyor. Tüm $(i, j)$ indis çiftleri ($1 \leq i < j \leq n$) için $|A[i] - A[j]|$ farklarının en küçüğünü bulmamız isteniyor. Hangi yaklaşımı seçmeliyiz?*

*Çözüm.* **1. Yaklaşım (Kaba Kuvvet):** Tüm çiftleri iç içe iki döngüyle deneriz. Bölüm 5'te gördüğümüz kombinasyon formülüyle, tüm çiftlerin sayısı $\binom{n}{2} = \frac{n(n-1)}{2}$'dir. $n = 10^5$ için:

$$\frac{10^5 \times (10^5 - 1)}{2} \approx \frac{10^{10}}{2} = 5 \times 10^9 \text{ işlem}$$

$10^8$ kuralına göre bu işlem yaklaşık $\frac{5 \times 10^9}{10^8} = 50$ saniye sürer. Süre sınırı 1 saniye olduğundan bu yöntem **Zaman Sınırı Aşımı** (Time Limit Exceeded - TLE) verecektir.

**2. Yaklaşım (Sıralama):** Sayıları önce küçükten büyüğe sıralarız. Birbirine en yakın iki sayı, sıralı dizide kesinlikle ardışık olmak zorundadır.

1. Diziyi sırala: $O(n \log n)$
2. Yan yana duran elemanların farkına tek bir döngüde bak: $O(n)$

Toplam karmaşıklık $O(n \log n)$ olur. $n = 10^5$ için:

$$n \log_2 n \approx 10^5 \times 17 \approx 1.7 \times 10^6 \text{ işlem}$$

Bu da yaklaşık 0.02 saniye sürer ve süre sınırının oldukça altında kalır. □

### 21.3.3 Alan (Bellek) Karmaşıklığı

Algoritmalar yalnızca zaman değil, hafıza da tüketir. Yarışmalarda tipik bellek sınırı 128 MB ile 512 MB arasındadır (en yaygın olanı 256 MB).

Bellek analizinde temel veri tiplerinin kapladığı yerleri bilmek gerekir:

- `bool` / `char`: 1 bayt
- `int`: 4 bayt
- `long long` / `double`: 8 bayt

Bellek miktarını megabayta çevirmek için:

$$1 \text{ KB} = 1024 \text{ bayt}, \quad 1 \text{ MB} = 1024 \text{ KB} \approx 10^6 \text{ bayt}$$

Pratik kural olarak:

- $10^7$ elemanlı bir `int` dizisi: $10^7 \times 4$ bayt $\approx 40$ MB yer tutar.
- $10^8$ elemanlı bir `int` dizisi: $\approx 400$ MB yer tutar (ve 256 MB sınırında hafıza aşımı verir).
- $5000 \times 5000$ boyutunda iki boyutlu bir `int` matrisi: $2.5 \times 10^7 \times 4 \approx 100$ MB yer tutar.

**Örnek 21.10** (Alan-Zaman Takası (Space-Time Tradeoff))**.** *Algoritma tasarımında sıklıkla ek bellek kullanılarak hız kazanılır veya tam tersine bellek tasarrufu adına işlem süresi artırılır.*

*Bir dizideki sayıların daha önce görülüp görülmediğini denetlemek için:*

- *Ekstra bellek kullanmazsak: Her yeni eleman için dizinin başına bakarız* $\rightarrow O(n^2)$ *zaman,* $O(1)$ *ek bellek.*
- *Bir frekans dizisi (veya doğruluk tablosu) tutarsak: Sayıyı gördüğümüzde* `var_mi[x] = true` *yaparız* $\rightarrow O(n)$ *zaman, sayının alabileceği en büyük değer (*$M$*) kadar* $O(M)$ *ek bellek.*

**Örnek 21.11** (Rekürsiyon Yığını (Call Stack))**.** *Yalnızca tanımladığınız diziler değil, çağırdığınız rekürsif (rekürsif) fonksiyonlar da bellek tüketir. Her fonksiyon çağrısı işletim sisteminin çağrı yığını (call stack) bölgesinde yer tutar.*

*Örneğin:*

```
void dfs(int u) {
    for (int v : komsular[u]) {
        if (!ziyaret[v]) dfs(v);
    }
}
```

*Graf bir zincir biçimindeyse ve $n = 200\,000$ ise, `dfs` fonksiyonu iç içe $200\,000$ kez çağrılır. Her çağrı yığında yaklaşık 32–64 bayt tuttuğunda toplam yığın boyutu $200\,000 \times 48 \approx 10$ MB seviyesine ulaşır. Pek çok yarışma platformunda ve Windows sistemlerinde varsayılan yığın limiti 8 MB civarındadır. Program genel bellek sınırını aşmamış olsa dahi **Stack Overflow** (Yığın Taşması) hatası alarak çöker.*

### 21.3.4 Bölüm Özeti ve Kontrol Listesi

Bir problemi çözerken şu adımları alışkanlık haline getiriniz:

1. Soru metnindeki $n$ sınırına bakarak hedef karmaşıklığı belirleyin ($n = 10^5$ ise $O(n^2)$ bir algoritma tasarlamak yerine doğrudan $O(n \log n)$ veya $O(n)$ çözümlere yönelin).
2. Kodunuzdaki iç içe döngülerin sınırlarını ve yerleşik dil fonksiyonlarının maliyetlerini tek tek hesaba katın.
3. İşlem sayınızın $10^8$ sınırını aşıp aşmadığını kaba bir hesapla test edin.
4. Tanımladığınız dizilerin toplam boyutunun 256 MB limitini geçmediğinden emin olun.

# Bölüm 22

# Diziler ve Temel Teknikler

Algoritmik problemlerin büyük bir kısmında veriler tek bir sayı veya değişken olarak değil, belirli bir sıra dahilinde peş peşe dizilmiş değerler topluluğu olarak karşımıza çıkar. Bir yarışmadaki katılımcıların puanları, bir hisse senedinin gün gün fiyatı ya da bir sensörden saniye saniye okunan sıcaklık değerleri bu duruma doğal birer örnektir.

Bu bölümde, programlamanın en temel doğrusal veri yapısı olan dizileri ele alıyor; ardından zaman sınırına takılmamak için sıkça başvurulan iki temel tekniği inceliyoruz: *prefix sums* (*prefix sums*) ve *two pointers* (*two pointers*).

## 22.1 Diziler, İndis ve Tarama

### 22.1.1 Motivasyon: Neden Dizi Kullanırız?

Elinizde 5 adet tam sayı olduğunu ve bunların ortalamasını bulmak istediğinizi varsayalım. Kolaylıkla $a, b, c, d, e$ adında beş değişken tanımlayıp toplamlarını 5'e bölebilirsiniz. Peki ya 100 000 adet sayının ortalamasını, en büyüğünü veya sıralanmış halini bulmanız gerekirse? Yüz bin ayrı değişken ismi tanımlamak hem pratik değildir hem de algoritmik olarak anlamsızdır.

Bize gereken şey; bellekte art arda duran, tek bir isimle erişebildiğimiz ve her bir elemanına sırasıyla birer numara (indis) vererek ulaşabildiğimiz bir yapıdır. İşte bu yapıya **dizi** (*array*) adı verilir.

**Tanım 22.1** (Dizi ve İndis)**.** *Aynı türden verilerin bellekte ardışık hücrelerde saklandığı, her bir elemanına* 0*'dan (veya matematiksel yazımda* 1*'den) başlayan bir tamsayı ile doğrudan erişilebilen veri yapısına* ***dizi*** *denir. Bir elemanın dizideki konumunu belirten bu tam sayıya* ***indis*** *(index) adı verilir.*

Bir $A$ dizisinin $i$'inci elemanı programlamada genellikle `A[i]`, matematikte ise alt indisle $A_i$ veya $A[i]$ olarak gösterilir. C ve Python gibi dillerde dizilerin ilk

elemanı daima 0. indiste yer alır:

$$A = [A_0, A_1, A_2, \ldots, A_{n-1}]$$

### 22.1.2 Doğrudan Erişim ve Bellek Mantığı

Dizilerin en belirgin avantajı, kaçıncı sırada olursa olsun herhangi bir elemana anında, yani $O(1)$ zamanda erişebilmesidir.

Bellek mimarisinde her bir veri hücresinin bir adresi vardır. $n$ elemanlı bir tam sayı dizisi oluşturduğumuzda, işletim sistemi bellekte yan yana duran bloklar tahsis eder. İlk elemanın bellek adresi $\text{adres}_0$ ve her bir tam sayının kapladığı alan $w$ bayt (örneğin 32-bit tamsayılar için $w = 4$ bayt) ise, $i$'inci elemanın adresi şu formülle doğrudan hesaplanır:

$$\text{adres}_i = \text{adres}_0 + i \times w$$

İşlemci bu çarpma ve toplama işlemini tek bir makine çevriminde gerçekleştirir; aradaki elemanları tek tek saymak zorunda kalmaz. Bu nedenle $A[0]$'a erişmekle $A[99999]$'a erişmek aynı süreyi alır.

### 22.1.3 Diziyi Döngüyle Gezme (Tarama)

Dizi algoritmalarının temeli, elemanları belirli bir sıra ile incelemeye, yani *tarama* (*traversal*) işlemine dayanır. Bir diziyi baştan sona tek bir döngüyle geçtiğimizde, $n$ elemanın her birine tam bir kez uğrarız; bu da $O(n)$ karmaşıklık demektir.

**Örnek 22.2.** *$n$ elemanlı bir $A$ tam sayı dizisindeki en büyük elemanı ve bu elemanın ilk görüldüğü indisi bulunuz.*

*Çözüm ve Düşünce Yolu.* Dizideki sayıları önceden bilmediğimiz için her sayıyı en az bir kez incelemek zorundayız. Karşılaştığımız en büyük değeri hafızada tutabiliriz. İlk başta gördüğümüz tek eleman olan $A[0]$'ı geçici olarak "en büyük" kabul ederiz. Ardından 1'inci indisten son indise kadar diziyi sırayla gezeriz. Gezinti sırasında elimizdeki değeri aşan bir $A[i]$ ile karşılaşırsak, en büyük değeri ve indisini güncelleriz.

Bu işlem, Bölüm 18'de incelediğimiz *yarı-değişmez* (*monovariant*) mantığını taşır: $k$'ıncı adım tamamlandığında, elimizdeki değer $A[0 \ldots k]$ alt dizisinin en büyük değeridir.

```
int en_buyuk = A[0];
int en_buyuk_indis = 0;

for (int i = 1; i < n; i++) {
    if (A[i] > en_buyuk) {
        en_buyuk = A[i];
        en_buyuk_indis = i;
```

```
    }
}
```

Her adımda yalnızca sabit sayıda işlem ($O(1)$) yapıldığından ve döngü $n-1$ kez döndüğünden toplam zaman karmaşıklığı $O(n)$'dir. □

**Örnek 22.3.** *Bir $A$ dizisinin kendi içinde tersine çevrilmesi (reverse): Ek bir dizi kullanmadan, $A = [A_0, A_1, \ldots, A_{n-1}]$ dizisini $A = [A_{n-1}, A_{n-2}, \ldots, A_0]$ haline getiriniz.*

*Çözüm.* İlk eleman ($A[0]$) son elemanla ($A[n-1]$), ikinci eleman ($A[1]$) sondan bir öncekiyle ($A[n-2]$) yer değiştirmelidir. Genel kural olarak $i$'inci eleman ($n-1-i$)'inci elemanla takas edilir (*swap*).

Burada döngü sınırına dikkat edilmelidir: Dizinin sonuna kadar gidersek, ilk yarıda takas ettiğimiz elemanları ikinci yarıda tekrar takas edip eski hallerine döndürürüz. Dolayısıyla yalnızca dizinin ortasına ($\lfloor n/2 \rfloor$) kadar ilerlememiz yeterlidir:

```
FUNCTION reverseArray(A)
    n = LENGTH(A)
    FOR i FROM 0 TO FLOOR(n / 2) - 1
        SWAP A[i] AND A[n - 1 - i]
    END FOR
END FUNCTION
```

Döngü $\lfloor n/2 \rfloor$ adım attığı için zaman karmaşıklığı $O(n)$, ek bellek kullanımı ise $O(1)$'dir. □

**Örnek 22.4** (Frekans Dizisi / Sayma Tekniği)**.** *Elemanları 0 ile $K$ arasında değişen $n$ elemanlı bir dizide, her sayının kaçar kez tekrar ettiğini $O(n+K)$ zamanda bulunuz.*

*Çözüm.* İki iç içe döngü kurup her eleman için diziyi baştan sona taramak $O(n^2)$ zaman alırdı. Bunun yerine dizinin elemanlarını doğrudan birer *indis* olarak kullanabiliriz. Boyutu $K+1$ olan ve tüm elemanları 0'la başlatılmış bir `sayac` dizisi oluşturalım:

```
// sayac dizisinin boyutu K + 1 olmali ve sifirlanmalidir
for (int i = 0; i < n; i++) {
    sayac[A[i]]++;
}
```

Bu işlemde her bir $A[i]$ için `sayac[A[i]]` değerini 1 artırırız. Böylece tek bir taramayla ($O(n)$) tüm frekanslar belirlenmiş olur. □

### En Sık Yapılan Hatalar: Sınır Taşması (Out-of-Bounds)

Dizilerle çalışırken yarışmalarda en çok karşılaşılan ve programın anında çökmesine (Run Time Error) ya da tanımsız davranmasına (*undefined behavior*) yol açan hata **indis sınır taşması**dır.

- $n$ elemanlı bir dizide geçerli indisler $0, 1, \ldots, n-1$'dir. `A[n]` indisine erişmeye çalışmak tipik bir "bir fazlası hatası"dır (*off-by-one error*).
- Komşu elemanları kontrol ederken (örneğin $A[i]$ ile $A[i+1]$ veya $A[i-1]$'i kıyaslarken) döngü sınırlarını daraltmayı unutmak: Döngü $i = 0$'dan $n-1$'e kadar dönüyorsa, döngü içinde $A[i+1]$ yazmak $i = n-1$ iken $A[n]$'e erişir ve taşma üretir.

## 22.2 Prefix Sums

### 22.2.1 Motivasyon: Aralıksız Sorgular ve Zaman Çıkmazı

Bir fabrikada çalışan bir makinenin $n$ gün boyunca harcadığı elektrik enerjisi bir $A$ dizisinde verilmiş olsun. Fabrika yönetimi sık sık şu soruyu yöneltiyor: "$L$'inci gün ile $R$'inci gün arasında toplam ne kadar elektrik harcandı?"

Eğer bu soru toplam $q$ defa sorulursa ve biz her soru için $L$'den $R$'ye kadar olan sayıları bir döngü ile toplarsak ne olur?

$$\text{Toplam} = \sum_{i=L}^{R} A[i]$$

Her bir aralık sorgusu en kötü durumda $O(n)$ adım sürer. $q$ adet sorgu için harcanan toplam zaman $O(q \times n)$ olur. $n = 100\,000$ ve $q = 100\,000$ olan bir problemde işlem sayısı yaklaşık $10^{10}$ seviyesine ulaşır. Modern bir işlemci bir saniyede yaklaşık $10^8$ işlem yapabildiğinden, bu kaba kuvvet (*brute-force*) yaklaşım 100 saniye sürecek ve sınavdaki 1.0 saniyelik zaman sınırını katbekat aşacaktır.

Dizideki sayılar sabitken her seferinde aynı aralıkları tekrar tekrar toplamak yerine bu toplamları önceden hazırlayabiliriz.

### 22.2.2 Sezgi ve Genel Kural

Somut bir örnek üzerinden düşünelim. Bir yol üzerinde 0'ıncı kilometreden başlayarak ilerleyen bir aracın belirli noktalardaki kilometre taşlarını biliyor olalım:

$$A = [3,\ 1,\ 4,\ 1,\ 5,\ 9,\ 2]$$

Bu aracın 2. eleman ile 5. eleman arasındaki (yani $4, 1, 5, 9$) mesafesini bulmak istiyoruz.

Yolun en başından itibaren gidilen toplam mesafeleri önceden kaydetmiş olsaydık:

- Baştan 5. elemana kadar olan toplam mesafe: $3 + 1 + 4 + 1 + 5 + 9 = 23$
- Baştan 1. elemana kadar olan toplam mesafe: $3 + 1 = 4$

Aradığımız $[2, 5]$ aralığı, aslında ilk 5 elemanın toplamından ilk 1 elemanın toplamının çıkarılmasıyla bulunur:

$$\text{Aralık Toplamı} = 23 - 4 = 19$$

Kontrol edelim: $4 + 1 + 5 + 9 = 19$. Tek bir çıkarma işlemiyle sonuca ulaştık.

**Tanım 22.5** (Prefix Sums Dizisi)**.** *Boyutu $n$ olan bir $A$ dizisi verildiğinde, $P[0] = 0$ olmak üzere her $1 \leq k \leq n$ için*

$$P[k] = \sum_{i=0}^{k-1} A[i] = A[0] + A[1] + \cdots + A[k-1]$$

*şeklinde tanımlanan $n+1$ elemanlı $P$ dizisine $A$'nın **prefix sum dizisi** (prefix sum array) denir.*

**Notasyon ve 1-Tabanlı Tercih:** Prefix sums dizilerinde $P$'nin boyutunu $n+1$ seçip $P[0] = 0$ tanımlamak büyük bir indeksleme kolaylığı sağlar. Böylece ilk $k$ elemanın toplamı doğrudan $P[k]$ olur.

**Teorem 22.6** (Aralık Toplamı İlkesi)**.** *0-tabanlı indekslenmiş bir $A$ dizisinin $[L, R]$ indis aralığındaki elemanlarının toplamı, $P$ prefix sum dizisi kullanılarak $O(1)$ zamanda şu formülle hesaplanır:*

$$\sum_{i=L}^{R} A[i] = P[R+1] - P[L]$$

*Kanıt.* Tanım gereği:

$$P[R+1] = A[0] + A[1] + \cdots + A[L-1] + A[L] + \cdots + A[R]$$

ve

$$P[L] = A[0] + A[1] + \cdots + A[L-1]$$

dir. İki eşitliği taraf tarafa çıkardığımızda:

$$P[R+1] - P[L] = \left(\sum_{i=0}^{R} A[i]\right) - \left(\sum_{i=0}^{L-1} A[i]\right) = \sum_{i=L}^{R} A[i]$$

elde edilir. Bu hesaplama yalnızca tek bir çıkarma işleminden ibaret olduğundan çalışma zamanı $O(1)$'dir. □

### 22.2.3 Prefix Sums İnşası

$P$ dizisini her adımda baştan toplayarak kurarsak inşa süreci $O(n^2)$ sürer. Ancak bir önceki toplamı kullanarak yeni elemanı eklemek doğal bir yineleme verir:

$$P[i] = P[i-1] + A[i-1] \quad (1 \le i \le n)$$

Bu ilişki sayesinde $P$ dizisi $O(n)$ zamanda, tek bir geçişte kurulur.

```
// A dizisi n elemanli, P dizisi n + 1 elemanli
P[0] = 0;
for (int i = 1; i <= n; i++) {
    P[i] = P[i - 1] + A[i - 1];
}

// L ile R araligindaki toplami sorgulamak icin (0 <= L <= R < n):
long long aralik_toplami = P[R + 1] - P[L];
```

Böylece $q$ sorguluk problemimizin toplam süresi $O(q \times n)$ yerine $O(n+q)$ olur ve sorgular hızla yanıtlanır.

**Örnek 22.7.** *Sıfırlardan ve birlerden oluşan bir dizide, verilen $[L, R]$ aralığında kaç tane `$1$` olduğunu $O(1)$ zamanda bulan algoritmayı tasarlayınız.*

*Çözüm.* Yalnızca 0 ve 1'lerden oluşan bir dizide elemanların toplamı, dizideki 1'lerin sayısına tam olarak eşittir. Dolayısıyla dizinin prefix sumını aldığımızda, herhangi bir $[L, R]$ aralığındaki 1'lerin sayısı doğrudan $P[R+1] - P[L]$ formülüyle elde edilir. □

**Örnek 22.8** (Toplamı $K$ Olan Alt Dizi Sayısı)**.** *Negatif sayılar da içerebilen $n$ elemanlı bir $A$ dizisinde, toplamları tam olarak $K$'ya eşit olan ardışık alt dizilerin sayısını bulunuz. ($n \le 200\,000$)*

*Çözüm ve Düşünce Yolu.* Bu fikir şuradan gelir: Bir $[L, R]$ alt dizisinin toplamı $P[R+1] - P[L]$'dir. Bizden istenen koşul:

$$P[R+1] - P[L] = K \iff P[L] = P[R+1] - K$$

Bu matematiksel dönüşüm çözümü oldukça sadeleştirir. Diziyi soldan sağa doğru tararken ($R$'yi 0'dan $n-1$'e büyütürken), o ana kadar gördüğümüz prefix sums'ın frekansını bir tabloda saklarsak; her adımda "Geçmişte değeri $P[R+1] - K$ olan kaç tane $P[L]$ gördüm?" sorusunu doğrudan yanıtlayabiliriz:

1. `toplam_adet = 0`, `suanki_toplam = 0`.
2. Boş bir frekans tablosu açıp `frekans[0] = 1` yazarız (hiç eleman almadığımız boş önek toplamı 0'dır).
3. Dizinin her $x$ elemanı için:

- `suanki_toplam += x`
- Aradığımız eski değer `hedef = suanki_toplam - K`.
- Eğer `hedef` tabloda varsa, `toplam_adet += frekans[hedef]`.
- `frekans[suanki_toplam]` değerini 1 artır.

Bu algoritma diziyi tek bir geçişte, $O(n)$ sürede çözer. □

### 22.2.4 İki Boyutlu Prefix Sums

Prefix sums mantığı çok boyutlu dizilere de doğrudan genellenir. $H \times W$ boyutlarında bir matriste verilen herhangi bir dikdörtgensel bölgenin toplamını $O(1)$ sürede bulmak için benzer bir önek hesabı kurulabilir.

**Tanım 22.9** (2 Boyutlu Prefix Sums)**.** *$M$ matrisi için $P[r][c]$, sol üst köşesi $(0,0)$ ve sağ alt köşesi $(r-1, c-1)$ olan dikdörtgenin içindeki tüm sayıların toplamıdır.*

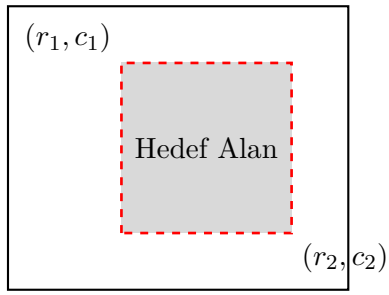
Bunu hesaplarken Bölüm 8'de ele aldığımız içerme-dışarma ilkesinden (*inclusion-exclusion*) faydalanırız:

$$P[r][c] = M[r-1][c-1] + P[r-1][c] + P[r][c-1] - P[r-1][c-1]$$

Solundaki ve üstündeki alanlar toplanır; her ikisinde de ortak bulunan sol-üst çapraz alan iki kez sayıldığı için bir kez çıkarılır.

Benzer şekilde, sol üst köşesi $(r_1, c_1)$ ve sağ alt köşesi $(r_2, c_2)$ olan bir alt matrisin toplamı şudur:

$$\text{Alan Toplamı} = P[r_2+1][c_2+1] - P[r_1][c_2+1] - P[r_2+1][c_1] + P[r_1][c_1]$$




Şekil 22.1: 2 Boyutlu Prefix Sums'ta Hedef Alanın İçerme-Dışarma ile Hesaplanması

### 22.2.5 Fark Dizisi Tekniği (Difference Array)

Prefix sums'ın ters işlemi de aralık güncellemelerinde oldukça etkilidir. Diyelim ki boş bir diziyle başladınız ve size şu tip güncellemeler veriliyor: "$[L, R]$ aralığındaki tüm sayılara $V$ değerini ekle."

$q$ adet aralık güncellemesini döngüyle yapmak $O(q \times n)$ sürer. Ancak **fark dizisi** tekniğiyle her güncellemeyi $O(1)$ işlemle kaydedebiliriz:

$D$ fark dizisi, $D[i] = A[i] - A[i-1]$ olarak tanımlansın ($A[-1] = 0$). $[L, R]$ aralığına $V$ eklemek, fark dizisinde yalnızca iki noktayı etkiler:

1. $D[L]$ değeri $V$ kadar artar ($A[L]$, $A[L-1]$'e kıyasla $V$ kadar büyümüştür).
2. $D[R+1]$ değeri $V$ kadar azalır ($A[R+1]$ değişmemiş, ancak solundaki $A[R]$ değeri $V$ kadar büyümüştür).

Tüm sorgular bittikten sonra, fark dizisinin prefix sumı alınarak orijinal dizinin son hali tek hamlede $O(n)$ sürede elde edilir.

#### En Sık Yapılan Hatalar

- **Tamsayı Taşması (Integer Overflow):** Dizi elemanları $10^9$ mertebesindeyse, bu elemanların toplamı 32-bit işaretli tamsayı sınırını ($2^{31} - 1 \approx 2 \times 10^9$) hızla aşar. Prefix sums dizisi tanımlanırken C/C++ dillerinde mutlaka `long long` türü tercih edilmelidir.
- **İndis Kaymaları:** $P[R] - P[L]$ mi yoksa $P[R+1] - P[L]$ mi yazılacağı sıkça karıştırılır. Sağlamasını yapmak için 1 elemanlı bir aralığı ($L = R$) test edebilirsiniz: Formülün $P[L+1] - P[L] = A[L]$ sonucunu vermesi gerekir.

## 22.3 Two Pointers

### 22.3.1 Motivasyon: Sıralı Yapıların Getirdiği Bilgi Gücü

Elimizde küçükten büyüğe sıralanmış bir $A$ dizisi olsun:

$$A = [1, 3, 4, 7, 10, 11, 15], \quad n = 7$$

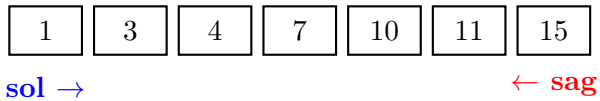
Bu dizide toplamları tam olarak $X = 17$ olan iki eleman bulmak istiyoruz.

Akla gelen ilk yöntem, iç içe iki döngü yazıp tüm olası $(A[i], A[j])$ çiftlerini denemektir. Olası çift sayısı $\binom{n}{2} = \frac{n(n-1)}{2} = O(n^2)$'dir. $n = 100\,000$ durumunda bu yaklaşım zaman aşımına uğrar.

Dizinin **sıralı** olması ise bize net bir yön bilgisi verir: Sağa gitmek sayıları büyütür, sola gitmek küçültür. Bu sayede arama uzayını körlemesine denemek yerine adım adım daraltabiliriz.

### 22.3.2 Uçlardan Merkeze Tarama (Opposite Direction)

Diziye iki uçtan yaklaşalım: Biri dizinin en küçük elemanını (en başı, $sol = 0$), diğeri ise en büyük elemanını (en sonu, $sag = n - 1$) göstersin. Konum belirten bu değişkenlere **işaretçi** (*pointer*) diyoruz.




Şekil 22.2: Two Pointers Yaklaşımında İşaretçilerin İki Uçtan Başlatılması

Şimdi mevcut toplama bakalım: $T = A[sol] + A[sag] = 1 + 15 = 16$.

- Aradığımız toplam 17'ydi; mevcut toplam (16) küçük kaldı.
- Toplamı büyütmek için *sag* işaretçisini sola kaydırırsak sayılar küçülür ve toplam daha da azalır. Dolayısıyla $A[sag]$ ile eşleşebilecek başka hiçbir eleman kalmamıştır. Toplamı büyütmenin tek yolu *sol* işaretçisini bir adım sağa ilerletmektir ($sol = 1$).

Yeni durum: $A[sol] = 3$, $A[sag] = 15$. Toplam $3 + 15 = 18$.

- Bu sefer toplam hedefimizden büyük çıktı ($18 > 17$).
- Toplamı küçültmek için *sag* işaretçisini bir adım sola kaydırmalıyız ($sag = 5$).

Yeni durum: $A[sol] = 3$, $A[sag] = 11$. Toplam $3 + 11 = 14 < 17$. *sol* bir sağa ($sol = 2$).
Yeni durum: $A[sol] = 4$, $A[sag] = 11$. Toplam $4 + 11 = 15 < 17$. *sol* bir sağa ($sol = 3$).
Yeni durum: $A[sol] = 7$, $A[sag] = 11$. Toplam $7 + 11 = 18 > 17$. *sag* bir sola ($sag = 4$).
Yeni durum: $A[sol] = 7$, $A[sag] = 10$. Toplam $7 + 10 = 17 = X$. Aradığımız çifti bulduk: $(7, 10)$.

**Teorem 22.10** (Two Pointers ile İki Toplam (Two Sum))**.** *Küçükten büyüğe sıralı $n$ elemanlı bir dizide toplamı $X$ olan bir çiftin varlığı, two pointers tekniği ile $O(n)$ zamanda ve $O(1)$ ek bellek ile tespit edilebilir.*

*İspat ve Değişmez (Invariant).* Algoritmanın doğruluğunu göstermek için aradığımız çözümün adımlar sırasında elenmediğini kanıtlamalıyız.

Aranan çözümün $(i^*, j^*)$ indislerinde bulunduğunu varsayalım ($i^* < j^*$). Bölüm 18'de ele aldığımız değişmezlik yaklaşımıyla, ispatlamak istediğimiz *değişmez* şudur: "Herhangi bir anda $sol \leq i^*$ ve $sag \geq j^*$ eşitsizlikleri daima korunur."

Başlangıçta $sol = 0 \leq i^*$ ve $sag = n - 1 \geq j^*$ olduğundan ifade geçerlidir. Algoritmanın adımlarında iki durum oluşabilir:

1. $A[sol] + A[sag] < X$: Eğer $sol < i^*$ ise, $sol$'un bir artması aradığımız $i^*$'a doğru yaklaşmadır; değişmez bozulmaz. $sol = i^*$ iken bu durum gerçekleşebilir mi? Gerçekleşemez; çünkü $sol = i^*$ olsaydı, $sag \geq j^*$ koşulundan ötürü $A[sol]+A[sag] = A[i^*]+A[sag] \geq A[i^*]+A[j^*] = X$ olurdu. Bu ise toplamın $X$'ten küçük olmasıyla çelişir. Dolayısıyla $sol = i^*$ iken işaretçi artırılmaz.

2. $A[sol] + A[sag] > X$: Benzer şekilde $sag = j^*$ iken toplam asla $X$'ten büyük olamaz. Dolayısıyla $sag$, hiçbir zaman $j^*$'ın soluna düşecek şekilde eksiltilmez.

İşaretçilerden en az biri her adımda diğerine bir birim yaklaşır ($sag - sol$ farkı kesin olarak azalır). Dolayısıyla algoritma en fazla $n$ adımda ya hedef çifte ulaşır ya da $sol \geq sag$ koşuluyla sonlanır. Toplam işlem sayısı $O(n)$'dir. □

```
int sol = 0, sag = n - 1;
bool bulundu = false;

while (sol < sag) {
    long long toplam = (long long)A[sol] + A[sag];
    if (toplam == X) {
        bulundu = true;
        break;
    } else if (toplam < X) {
        sol++; // Toplami buyut
    } else {
        sag--; // Toplami kucult
    }
}
```

### 22.3.3 Aynı Yöne Kayan Pencere (Sliding Window)

İki işaretçinin her zaman dizinin iki zıt ucundan başlaması gerekmez. Kimi problemlerde two pointers de dizinin başından (0'dan) başlar ve aynı yöne doğru ilerler. Bu yaklaşıma **kayan pencere** (*sliding window*) denir. Pencerenin sınırlarını $[sol, sag]$ aralığı belirler.

**Örnek 22.11** (Toplamı En Fazla $K$ Olan En Uzun Alt Dizi)**.** *Tüm elemanları **pozitif** tam sayılar olan bir $A$ dizisinde, elemanları toplamı $\leq K$ olan en uzun ardışık alt dizinin uzunluğunu bulunuz.*

*Çözüm ve Düşünce Yolu.* Tüm elemanlar pozitif olduğu için penceremize sağdan yeni bir eleman eklediğimizde ($sag$ arttığında) toplam kesinlikle *artar*; soldan bir eleman çıkardığımızda ($sol$ arttığında) toplam kesinlikle *azalır*. Bu durum arama için elverişli bir monotonluk sunar.

*sag* işaretçisini 0'dan $n-1$'e kadar birer birer ilerletiriz. Her adımda $A[sag]$'ı pencere toplamına ekleriz. Eğer toplam $K$'yı aşarsa, toplam tekrar $K$'nın altına inene kadar *sol* işaretçisini sağa kaydırıp $A[sol]$ değerlerini pencereden çıkarırız:

```
FUNCTION longestSubarray(A, K)
    left = 0
    currentSum = 0
    maxLength = 0

    FOR right FROM 0 TO length(A) - 1 DO
        currentSum = currentSum + A[right]

        WHILE currentSum > K AND left <= right DO
            currentSum = currentSum - A[left]
            left = left + 1
        END WHILE

        maxLength = MAX(maxLength, right - left + 1)
    END FOR

    RETURN maxLength
END FUNCTION
```

**Karmaşıklık Analizi (Amorti Edilmiş Analiz):** Kodda iç içe bir `for` ve bir `while` döngüsü yer alması ilk bakışta $O(n^2)$ izlenimi verebilir. Ancak işaretçilerin hareketine dikkat edilmelidir:

- *sag* işaretçisi 0'dan $n-1$'e kadar tam $n$ defa artırılır.
- *sol* işaretçisi hiçbir zaman sola dönmez; o da dizi boyunca en fazla $n$ defa artırılabilir.

Dolayısıyla içteki `while` döngüsünün toplam adım sayısı, programın tüm çalışma süresi boyunca en fazla $n$'dir. Amorti edilmiş analizle her eleman pencereye bir kez girer ve en fazla bir kez çıkar. Toplam zaman karmaşıklığı $O(n)$'dir. □

**Örnek 22.12** (Sıralı İki Diziyi Birleştirme / Merge)**.** *Boyutları sırasıyla $n$ ve $m$ olan sıralı iki dizi $A$ ve $B$ verildiğinde; bu iki diziyi tek bir sıralı dizi halinde $O(n+m)$ zamanda birleştiriniz.*

*Çözüm.* Bu problem, Bölüm 23'te göreceğimiz Birleştirmeli Sıralama (*Merge Sort*) algoritmasının temel adımıdır. $A$ için bir $i$ işaretçisi, $B$ için bir $j$ işaretçisi tutarız. Her adımda $A[i]$ ve $B[j]$'den hangisi daha küçükse onu yeni dizimize ekler ve ilgili işaretçiyi bir ilerletiriz:

```
int i = 0, j = 0, k = 0;
// C dizisi (n + m) boyutundadir
while (i < n && j < m) {
    if (A[i] <= B[j]) {
```

```
        C[k++] = A[i++];
    } else {
        C[k++] = B[j++];
    }
}
// Kalan elemanlari aktar
while (i < n) C[k++] = A[i++];
while (j < m) C[k++] = B[j++];
```

Her adımda bir işaretçi ilerlediğinden toplam zaman $O(n+m)$ olur. □

### En Sık Yapılan Hatalar: Negatif Sayılar ve Monotonluk İhlali

Kayan pencere (veya two pointers) tekniğinin çalışabilmesi için sistemde bir **monotonluk** (tekdüzelik) bulunması şarttır. Örneğin "Toplamı $\leq K$ olan alt dizi" probleminde dizide **negatif sayılar da yer alsaydı**, pencereye sağdan eleman eklemek toplamı küçültebilir, soldan eleman çıkarmak toplamı büyütebilirdi. Bu durumda işaretçilerin hangi yönde ilerletilmesi gerektiği kestirilemez ve two pointers tekniği geçerliliğini yitirir.

**Altın Kural:** Negatif sayıların bulunduğu toplam problemlerinde two pointers yerine yukarıda incelediğimiz *prefix sum + frekans tablosu* yöntemi tercih edilmelidir.

## Bölüm Özeti ve Hangi Tekniği Seçmeli?

TÜBİTAK 1. Aşama sınavında bir dizi problemiyle karşılaştığınızda şu karar ağacı yol gösterici olabilir:

1. **Tekrarlı Aralık Sorguları (Range Queries):** Dizi sabit kalıyor ve bize defalarca $[L, R]$ aralığındaki toplam, çift sayı adedi vb. soruluyorsa → **Prefix Sums ($O(1)$ sorgu)**.
2. **Toplu Aralık Güncellemeleri (Range Updates):** Birden fazla aralığa sabit bir değer eklenip en sonunda dizinin nihai hali isteniyorsa → **Fark Dizisi ($O(1)$ güncelleme)**.
3. **Sıralı Dizide Eşleşme / Çift Arama:** Dizi sıralıysa (veya sıralanabiliyorsa) ve belirli bir toplamı/farkı sağlayan eleman çifti aranıyorsa → **Zıt Yönlü İki İşaretçi ($O(n)$)**.
4. **Pozitif Değerli Dizilerde Alt Dizi Optimizasyonu:** Elemanların pozitif olduğu ve bir aralık koşulunun (örneğin toplam $\leq K$) arandığı sorularda → **Kayan Pencere ($O(n)$)**.

# Bölüm 23

# Sıralama

## 23.1 Neden Sıralarız: Sıralı Verinin Gücü

Elimizde $n$ adet tamsayıdan oluşan bir liste olduğunu düşünelim:

$$A = [42, 17, 89, 5, 23, 71, 12, 60]$$

Bu listede 23 sayısının bulunup bulunmadığını kontrol etmek istersek, en kötü durumda listenin bütün elemanlarını tek tek incelememiz gerekir. Eğer liste $n$ elemanlıysa, her bir arama sorgusu için $O(n)$ işlem yaparız. $q$ adet sorgumuz varsa toplam süre $O(q \cdot n)$ olur. $n = 10^5$ ve $q = 10^5$ olduğunda yapılacak işlem sayısı $10^{10}$ mertebesine çıkar ve standart bir işlemci için bu süre saniyeler değil, dakikalar sürer.

Şimdi aynı elemanları küçükten büyüğe sıralı bir biçimde tuttuğumuzu varsayalım:

$$A_{\text{sıralı}} = [5, 12, 17, 23, 42, 60, 71, 89]$$

Artık ortadaki elemana bakıp aradığımız sayının sağda mı yoksa solda mı kalması gerektiğini tek bir karşılaştırmayla anlarız. Her adımda arama uzayımız yarıya iner. Bölüm 24'te ayrıntılarına gireceğimiz ikili arama (binary search) sayesinde tek bir sorgu en fazla $\lceil \log_2 n \rceil$ adımda biter. $10^5$ sorgu için $10^5 \cdot 17 \approx 1.7 \times 10^6$ işlem yeterlidir; yani saniyenin küçük bir kesrinde işlem tamamlanır.

Sıralama (sorting), yalnızca arama hızını artırmakla kalmaz; algoritmik düşüncede birçok karmaşık problemi doğrudan gözlemlere indirger.

**Tanım 23.1** (Sıralama Problemi)**.** *Üzerinde tam sıralama bağıntısı $\leq$ tanımlı bir $S$ kümesinden alınan $n$ elemanlı bir $A = [a_1, a_2, \ldots, a_n]$ dizisi verilsin. Sıralama problemi, Bölüm 4'te ele aldığımız $\{1, 2, \ldots, n\}$ kümesinin öyle bir $\pi$ permütasyonunu bulma işlemidir ki,*

$$a_{\pi(1)} \leq a_{\pi(2)} \leq \cdots \leq a_{\pi(n)}$$

*koşulu sağlansın.*

Sıralı bir dizinin sunduğu başlıca yapısal avantajları şöyle özetleyebiliriz:

1. **Uç Değerlere Erişim:** Minimum eleman daima $A[1]$, maksimum eleman ise daima $A[n]$ konumundadır. Dolayısıyla uç değerlere $O(1)$ sürede erişilir.

2. **Tekrarları ve Benzersiz Elemanları Bulma:** Rastgele bir dizide aynı elemandan birden fazla olup olmadığını anlamak doğrudan bir bakışla zordur. Oysa sıralı bir dizide birbirine eşit elemanlar zorunlu olarak yan yana gelir ($A[i] = A[i+1]$). Tek bir doğrusal geçişle ($O(n)$ sürede) tüm tekrarlar tespit edilir veya ayıklanır.

3. **Aralık ve Yakınlık Problemleri:** Dizi içindeki en yakın iki elemanın farkını bulmak için sırasız dizide tüm çiftleri denemek $\binom{n}{2} = O(n^2)$ karşılaştırma gerektirir. Sıralı dizide ise en yakın iki eleman kesinlikle art arda durmak zorundadır; $\min_{1 \le i < n}(A[i+1] - A[i])$ değeri $O(n)$ sürede hesaplanır.

4. **Greedy Stratejiler:** Bölüm 27'de göreceğimiz üzere, bir problemi en küçük ya da en büyük değerden başlayarak adım adım çözme stratejileri daima sıralanmış veri üzerinde anlam kazanır.

**Örnek 23.2.** *Bir tam sayı dizisinde farkları birbirine en yakın iki elemanın mutlak farkını bulunuz.*

*Örnek girdi:* $A = [28, 4, 19, 45, 12, 73]$

*Çözüm.* Tüm olası çiftleri incelersek: $(28, 4), (28, 19), \ldots$ toplam $\binom{6}{2} = 15$ karşılaştırma gerekir. Genel durumda $n$ eleman için $\binom{n}{2} = \frac{n(n-1)}{2}$ çift vardır ve bu yaklaşım $O(n^2)$ zaman alır.

Oysa reel sayı doğrusunu düşündüğümüzde, birbirine en yakın iki nokta bu doğru üzerine dizildiklerinde zorunlu olarak komşu iki nokta olmak zorundadır. Araya başka bir nokta giriyorsa, komşulardan birine olan mesafe kesinlikle daha küçüktür.

O halde diziyi küçükten büyüğe sıralayalım:

$$A_{\text{sıralı}} = [4, 12, 19, 28, 45, 73]$$

Şimdi ardışık farkları hesaplayalım:

$$\begin{aligned} 12 - 4 &= 8 \\ 19 - 12 &= 7 \\ 28 - 19 &= 9 \\ 45 - 28 &= 17 \\ 73 - 45 &= 28 \end{aligned}$$

En küçük fark $\min(8, 7, 9, 17, 28) = 7$'dir. Bu kontrol yalnızca $n - 1$ adım sürdü. Sıralama maliyeti eklenince işlem toplamda $O(n \log n)$ sürede tamamlanır. □

**Örnek 23.3.** *Elimizde n kişinin katıldığı bir yarışmanın puanları bulunmaktadır. Puanlar bir tamsayı dizisi olarak verilmiştir. Bir puanın dizide "çoğunluk" (strictly majority) oluşturup oluşturmadığını, yani $n/2$'den fazla kez tekrar edip etmediğini belirlemek istiyoruz.*

*Örnek girdi:* $A = [3, 7, 3, 2, 3, 5, 3, 3, 1]$ *(*$n = 9$*)*

*Çözüm.* Eğer dizide bir eleman $\lfloor n/2 \rfloor + 1$ veya daha fazla kez bulunuyorsa, dizi sıralandığında bu elemanın kapladığı aralık o kadar geniş olmalıdır ki, dizinin tam orta noktası olan $\lfloor n/2 \rfloor$. indekse (0-tabanlı indekste $\lfloor n/2 \rfloor$) denk gelmek *zorundadır.*

Verilen diziyi sıralayalım:

$$A_{\text{sıralı}} = [1, 2, 3, 3, 3, 3, 3, 5, 7]$$

Dizinin ortasındaki eleman: indeksi $\lfloor 9/2 \rfloor = 4$ olan eleman $A[4] = 3$'tür. Eğer dizide çoğunluk elemanı varsa, bu eleman kesinlikle 3 olmak zorundadır. Şimdi tek bir geçişle 3'ün kaç kez geçtiğini sayarız: tam 5 defa geçmektedir ve $5 > 9/2$ olduğu için çoğunluk elemanı 3'tür.

Sıralama yapmasaydık her elemanın frekansını ayrı ayrı saymak veya ek veri yapıları kurmak zorunda kalacaktık. Sıralı yapının sunduğu geometrik yerleşim problemi tek bir noktaya indirgemiştir. □

**Örnek 23.4.** *n adet kapalı aralık veriliyor:* $[s_1, e_1], [s_2, e_2], \ldots, [s_n, e_n]$*. Bu aralıkların ortak bir kesişim kümesi olup olmadığını nasıl anlarız?*

*Çözüm.* İki aralığın kesişmesi için birinin başlangıcının diğerinin bitişinden küçük veya eşit olması gerekir. $n$ aralık için ortak bir nokta $x$'in bulunması, her $i$ için $s_i \leq x \leq e_i$ olması demektir. Bu ise şu anlama gelir:

$$\max_{1 \leq i \leq n} s_i \leq \min_{1 \leq i \leq n} e_i$$

Başlangıç ve bitiş noktalarını sıralı düşünelim. En büyük başlangıç noktası $S_{\max}$, en küçük bitiş noktası $E_{\min}$ olsun. Eğer $S_{\max} \leq E_{\min}$ ise $[S_{\max}, E_{\min}]$ aralığındaki her nokta tüm aralıkların ortak kesişimidir; aksi halde kesişim boştur. Uç değerlerin sıralı düşünülmesi çözümü doğrudan verir. □

Bu kazanımlar bizi temel bir soruya götürür: Sıralama işlemini en düşük maliyetle ve en verimli şekilde nasıl gerçekleştirebiliriz?

## 23.2 Bubble Sort ve Insertion Sort: Basit Yaklaşımlar ve $O(n^2)$ Sezgisi

Sıralama algoritmalarını anlamanın en doğal yolu, küçük veri kümeleri üzerinde elle yapılan işlemlerin adımlarını incelemektir. Bu doğrultuda önce $O(n^2)$ karma-

şıklığına sahip iki temel algoritmayla başlayalım: Kabarcık Sıralaması (Bubble Sort) ve Araya Ekleme Sıralaması (Insertion Sort).

### 23.2.1 Ters Düz Olma Sayısı (Inversion)

Bir dizinin ne kadar "karışık" olduğunu nasıl ölçeriz? Bu soru, sıralama algoritmalarının alt sınırlarını ve çalışma mantığını anlamak için temel bir kavramı doğurur.

**Tanım 23.5** (Ters Düz Çifti (Inversion))**.** *Bir $A$ dizisinde, $i < j$ indisleri için $A[i] > A[j]$ koşulu sağlanıyorsa, $(i, j)$ çiftine bir* ***ters düz çifti*** *(inversion) denir.*

Örneğin $A = [3, 1, 2]$ dizisinde:

- $i = 1, j = 2 \implies A[1] = 3 > A[2] = 1$ (ters düz)
- $i = 1, j = 3 \implies A[1] = 3 > A[3] = 2$ (ters düz)
- $i = 2, j = 3 \implies A[2] = 1 < A[3] = 2$ (sıralı)

Bu dizide toplam 2 adet ters düz çifti vardır. Tamamen sıralı bir dizide ($[1, 2, 3]$) ters düz sayısı 0'dır. Tamamen tersten sıralı bir dizide ($[3, 2, 1]$) ise olası her çift bir ters düzdür, yani $\binom{n}{2} = \frac{n(n-1)}{2}$ adet ters düz vardır.

**Teorem 23.6.** *Yalnızca komşu iki elemanın yerini değiştiren herhangi bir sıralama algoritması, diziyi sıralamak için en az dizideki ters düz sayısı kadar takas (swap) işlemi yapmak zorundadır.*

*Kanıt.* Dizideki iki komşu eleman $A[k]$ ve $A[k+1]$ olsun. Bu iki eleman yer değiştirdiğinde:

- $k$ ve $k+1$ dışındaki hiçbir indisin birbirine göre sırası değişmez.
- Diğer elemanların $A[k]$ ve $A[k+1]$ ile olan bağıl konumları değişmez (çünkü elemanlar bir bütün olarak blok halinde yer değiştirir).
- Yalnızca $A[k]$ ile $A[k+1]$ arasındaki sıra tersine döner.

Dolayısıyla tek bir komşu takası, toplam ters düz sayısını en fazla 1 azaltabilir (eğer $A[k] > A[k+1]$ ise 1 azalır, $A[k] < A[k+1]$ ise 1 artar). Başlangıçta $I$ adet ters düz varsa ve hedef 0 ters düze ulaşmaksa, en az $I$ adet takas zorunludur. □

### 23.2.2 Kabarcık Sıralaması (Bubble Sort)

**Sezgi:** Su dolu bir kabın dibindeki büyük hava kabarcıklarının yüzeye doğru yükselmesi gibi, dizideki büyük elemanların da adım adım dizinin sonuna doğru "yüzdürülmesi" fikrine dayanır.

Dizinin başından başlarız; her komşu çifte bakarız. Eğer soldaki sağdakinden büyükse yer değiştiririz. Bir tam geçişin sonunda en büyük eleman kesinlikle en sağdaki yerine oturmuş olur. İkinci geçişte ikinci en büyük yerine oturur.

**Örnek 23.7.** $A = [5, 1, 4, 2, 8]$ *dizisini Bubble Sort adımlarıyla sıralayalım.*

*Çözüm.* **1. Geçiş:**

- $[\mathbf{5}, \mathbf{1}, 4, 2, 8] \to 5 > 1$, takas et $\to [1, 5, 4, 2, 8]$
- $[1, \mathbf{5}, \mathbf{4}, 2, 8] \to 5 > 4$, takas et $\to [1, 4, 5, 2, 8]$
- $[1, 4, \mathbf{5}, \mathbf{2}, 8] \to 5 > 2$, takas et $\to [1, 4, 2, 5, 8]$
- $[1, 4, 2, \mathbf{5}, \mathbf{8}] \to 5 < 8$, dokunma $\to [1, 4, 2, 5, \mathbf{8}]$

En büyük eleman (8) ait olduğu son pozisyona ulaştı.

**2. Geçiş:**

- $[\mathbf{1}, \mathbf{4}, 2, 5, 8] \to 1 < 4$, dokunma
- $[1, \mathbf{4}, \mathbf{2}, 5, 8] \to 4 > 2$, takas et $\to [1, 2, 4, 5, 8]$
- $[1, 2, \mathbf{4}, \mathbf{5}, 8] \to 4 < 5$, dokunma

Bu geçişte ikinci en büyük eleman (5) yerine yerleşti.

**3. Geçiş:** Dizi taranır, hiçbir takas gerçekleşmez. Eğer bir geçiş boyunca hiç takas yapılmamışsa dizi zaten sıralıdır ve algoritma erkenden sonlandırılabilir. □

C dilinde gerçekleme:

```c
void bubble_sort(int A[], int n) {
    for (int i = 0; i < n - 1; i++) {
        int takas_oldu = 0;
        for (int j = 0; j < n - 1 - i; j++) {
            if (A[j] > A[j + 1]) {
                int gecici = A[j];
                A[j] = A[j + 1];
                A[j + 1] = gecici;
                takas_oldu = 1;
            }
        }
        if (!takas_oldu) break; // Erken bitirme optimizasyonu
    }
}
```

**Karmaşıklık Analizi:**

- En kötü durumda (dizi tersten sıralıysa): $\sum_{i=0}^{n-2}(n-1-i) = \frac{n(n-1)}{2}$ karşılaştırma ve aynı sayıda takas yapılır. Karmaşıklık $O(n^2)$'dir.
- En iyi durumda (dizi zaten sıralıysa): İlk geçişte hiç takas olmaz, $n-1$ karşılaştırma ile biter; $O(n)$ zaman alır.

### 23.2.3 Araya Ekleme Sıralaması (Insertion Sort)

**Sezgi:** Elimize aldığımız oyun kartlarını tek tek sıralamaya benzer. Sol elimizde her zaman sıralı bir deste tutarız. Masadan yeni bir kart çektiğimizde, bu kartı sol elimizdeki destenin sağından başlayarak sola doğru kaydırır ve ait olduğu uygun konuma yerleştiririz.

**Tanım 23.8** (Değişmez (Invariant) İlkesi)**.** *Bölüm 18'de incelediğimiz değişmez (invaryant) kavramı uyarınca, Insertion Sort algoritmasının $i$. adımının başında dizinin ilk $i$ elemanı olan $A[0 \dots i-1]$ kendi içinde sıralıdır. Algoritmanın amacı, $A[i]$ elemanını bu sıralı alt dizinin doğru yerine yerleştirerek değişmezi $A[0 \dots i]$ için korumaktır.*

**Örnek 23.9.** $A = [6, 3, 7, 1, 4]$ *dizisini Insertion Sort ile adım adım sıralayalım.*

*Çözüm.* Başlangıçta ilk eleman tek başına sıralıdır: [6].

**Adım 1 ($i = 1$):** Eleman 3. 3, 6'dan küçüktür; 6 sağa kaydırılır ve 3 başa yazılır:

$$[\mathbf{3}, \mathbf{6}, 7, 1, 4]$$

**Adım 2 ($i = 2$):** Eleman 7. 7, 6'dan büyüktür; yeri zaten doğrudur, kaydırma yapılmaz:

$$[\mathbf{3}, \mathbf{6}, \mathbf{7}, 1, 4]$$

**Adım 3 ($i = 3$):** Eleman 1. 1; sırasıyla 7, 6 ve 3'ten küçüktür. Tümü birer sağa kayar:

$$[\mathbf{1}, \mathbf{3}, \mathbf{6}, \mathbf{7}, 4]$$

**Adım 4 ($i = 4$):** Eleman 4. 4; 7 ve 6'dan küçüktür, sağa kayarlar. 3'ten büyüktür, 3'ün sağına yerleşir:

$$[\mathbf{1}, \mathbf{3}, \mathbf{4}, \mathbf{6}, \mathbf{7}]$$

Dizi tamamen sıralandı. □

Sözde kod ile gerçekleme:

```
Algorithm InsertionSort(A)
    n = length(A)
    for i = 1 to n - 1 do
        key = A[i]
        j = i - 1
        while j >= 0 and A[j] > key do
            A[j + 1] = A[j]
            j = j - 1
        end while
        A[j + 1] = key
    end for
End Algorithm
```

**Teorem 23.10** (Insertion Sort Karmaşıklığı)**.** *$n$ elemanlı ve içinde $I$ adet ters düz çifti barındıran bir dizide, Insertion Sort tam olarak $I$ adet kaydırma (eleman atama) işlemi yapar. Algoritmanın çalışma zamanı $\Theta(n + I)$'dır.*

*Kanıt.* $A[i]$ elemanı sola doğru kaydırılırken, kendisinden önce gelen ve kendisinden kesinlikle büyük olan her elemanın üzerinden tam bir adım atlar. Bu durum, $A[i]$'nin sağda yer aldığı her bir ters düz çifti için tam olarak bir kaydırmaya karşılık gelir. Dizi üzerinde $n - 1$ dış döngü adımı atıldığından, yapılan toplam iş $\Theta(n + I)$ mertebesindedir. □

Bu teorem önemli bir pratik sonucu doğurur: **Neredeyse sıralı dizilerde** (yani her elemanın nihai yerinden en fazla $k$ uzaklıkta olduğu durumlarda, $I \leq k \cdot n$), Insertion Sort $O(k \cdot n)$ gibi doğrusal bir sürede çalışır. Bu nedenle günümüz gelişmiş kütüphane algoritmaları (örneğin Timsort), diziler küçüldüğünde Insertion Sort'a geçiş yapar.

**Örnek 23.11.** *Bir dizide her elemanın ait olması gereken sıralı konumundan en fazla $k$ pozisyon uzakta olduğu bilinmektedir ($k \ll n$). Bu dizi için neden Insertion Sort $O(nk)$ zaman alır?*

*Çözüm.* İlk hamle olarak döngünün her adımındaki hareketi inceleyelim. Eklenen eleman $A[i]$, gerçek sıralı yerinden en fazla $k$ kadar uzaktadır. Dolayısıyla `while` döngüsü her bir $i$ için en fazla $k$ kez dönebilir. Dış döngü $n$ adım sürdüğüne göre, içteki kaydırmaların toplamı en fazla $n \cdot k$ olabilir. Toplam karşılaştırma ve işlem sayısı $O(n \cdot k)$ ile sınırlıdır. $k$ küçük bir sabitse bu süre $O(n)$'dir. □

### 23.2.4 En Sık Yapılan Hatalar

- **Bubble Sort'ta Döngü Sınırları:** İç döngüyü her zaman $n - 1$'e kadar koşturmak gereksiz karşılaştırmalara yol açar. $i$. geçişten sonra son $i$ eleman zaten yerleşmiştir, iç döngü $n - 1 - i$'de bitmelidir.
- **Insertion Sort'ta Dizi Dışına Çıkma:** `while` döngüsünde `j >= 0` kontrolünü `A[j] > anahtar` koşulundan *önce* yazmamak, C gibi dillerde negatif indeks erişimine (`A[-1]`) ve tanımsız davranışa (segmentation fault) yol açar.

## 23.3 Merge Sort ve Quick Sort: Böl-ve-Fethet ve $O(n \log n)$ Sınırı

$O(n^2)$ algoritmaları $n = 10^5$ gibi olimpiyat boyutlarında kesinlikle zaman aşımına (TLE) uğrar. $n^2 = 10^{10}$ işlem saniyeleri aşarken, $n \log_2 n \approx 1.7 \times 10^6$ işlem hızla tamamlanır. Bu sıçramayı sağlayan temel yaklaşım **Böl ve Fethet** (Divide and Conquer) paradigmasıdır.

### 23.3.1 Karşılaştırma Tabanlı Sıralamaların Alt Sınırı

Acaba daha da hızlı sıralayabilir miyiz? Örneğin $O(n)$ sürede ikişer ikişer karşılaştırarak genel bir diziyi sıralamak mümkün müdür?

**Teorem 23.12** (Bilgi-Kuramsal Alt Sınır)**.** *Yalnızca elemanların ikili karşılaştırmalarına dayanan herhangi bir sıralama algoritması, en kötü durumda en az* $\lceil \log_2(n!) \rceil = \Omega(n \log n)$ *karşılaştırma yapmak zorundadır.*

*Kanıt.* $n$ farklı elemandan oluşan bir dizinin $n!$ farklı olası sıralanışı (permütasyonu) vardır. Algoritma başlangıçta bu $n!$ permütasyondan hangisinin doğru olduğunu bilmez.

Yapılan her $A[i] \leq A[j]$ karşılaştırması "evet" veya "hayır" olmak üzere en fazla 2 yanıt üretebilir. Bu durum, olası durumlar kümesini bir karar ağacında (decision tree) iki dala ayırır. $k$ adet karşılaştırma yapıldığında ağacın en fazla $2^k$ yaprağı olabilir. Tüm permütasyonları ayırt edebilmek için yaprak sayısı en az permütasyon sayısı kadar olmalıdır:

$$2^k \geq n! \implies k \geq \log_2(n!)$$

Stirling yaklaşımına göre ($n! \approx \sqrt{2\pi n}(n/e)^n$):

$$\log_2(n!) = \sum_{i=1}^{n} \log_2 i \geq \sum_{i=\lfloor n/2 \rfloor}^{n} \log_2 i \geq \frac{n}{2} \log_2 \left(\frac{n}{2}\right) = \Omega(n \log n)$$

Demek ki karşılaştırma tabanlı hiçbir algoritma genel durumda $O(n \log n)$'den daha hızlı olamaz. Merge Sort ve Quick Sort bu teorik sınıra ulaşan optimal algoritmalardır. □

### 23.3.2 Birleştirmeli Sıralama (Merge Sort)

Merge Sort dengeli bir stratejiye dayanır:

1. **Böl:** Diziyi tam ortasından iki eşit (veya neredeyse eşit) parçaya ayır.
2. **Fethet:** Her iki yarıyı rekürsif olarak kendi içinde sırala.
3. **Birleştir (Merge):** Sıralı iki alt diziyi tek bir sıralı dizi halinde birleştir.

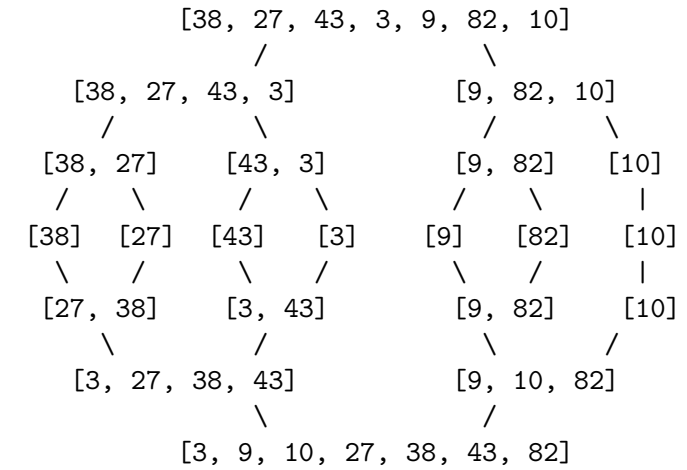
Algoritmanın temel adımı "Birleştirme" (Merge) aşamasıdır. Sıralı iki diziyi birleştirmek oldukça basittir: her iki dizinin de en küçük elemanı en baştadır. Bölüm 22'deki two pointers mantığıyla, iki baştaki elemanı karşılaştırır, küçük olanı yeni diziye ekler ve o dizideki işaretçimizi bir adım ilerletiriz.

**Örnek 23.13.** $L = [2, 5, 9]$ *ve* $R = [1, 4, 8]$ *sıralı listelerini tek bir sıralı dizide birleştirelim.*

*Çözüm.* İki işaretçi kullanalım: $i$, $L$'nin başında; $j$, $R$'nin başında olsun.

1. $L[i] = 2$, $R[j] = 1 \implies 1 < 2$, 1'i al, $j$ ilerler. Sonuç: $[1]$
2. $L[i] = 2$, $R[j] = 4 \implies 2 < 4$, 2'yi al, $i$ ilerler. Sonuç: $[1, 2]$
3. $L[i] = 5$, $R[j] = 4 \implies 4 < 5$, 4'ü al, $j$ ilerler. Sonuç: $[1, 2, 4]$
4. $L[i] = 5$, $R[j] = 8 \implies 5 < 8$, 5'i al, $i$ ilerler. Sonuç: $[1, 2, 4, 5]$
5. $L[i] = 9$, $R[j] = 8 \implies 8 < 9$, 8'i al, $j$ ilerler. Sonuç: $[1, 2, 4, 5, 8]$
6. $R$ bitti. $L$'de kalan elemanları (9) sona ekle. Sonuç: $[1, 2, 4, 5, 8, 9]$

$n$ elemanlı birleştirme işlemi sadece $O(n)$ zamanda bitti. □




Şekil 23.1: Merge Sort'un rekürsif ayrışma ve birleşme ağacı.

Merge Sort'un zaman karmaşıklığı şu rekürsif bağıntı ile ifade edilir:

$$T(n) = 2T(n/2) + O(n)$$

Ağacın derinliği $\log_2 n$ seviyedir ve her seviyedeki birleştirmelerin toplam maliyeti $O(n)$'dir. Dolayısıyla toplam süre her durumda (en iyi, ortalama, en kötü) $\Theta(n \log n)$'dir.

Merge Sort'un temel bedeli bellekten gelir: İki parçayı birleştirmek için orijinal dizi dışında $O(n)$ ek alana (yardımcı diziye) ihtiyaç duyar.

**Örnek 23.14.** *Bir dizideki toplam ters düz (inversion) sayısını $O(n \log n)$ sürede nasıl buluruz?*

*Çözüm.* Ters düz çiftlerinin konumuna odaklanalım. Tanım gereği $i < j$ iken $A[i] > A[j]$ aranır. Diziyi ikiye böldüğümüzde bir ters düz $(i, j)$ çifti üç yerde bulunabilir:

1. İkisi de sol yarıdadır ($i, j \leq$ orta).
2. İkisi de sağ yarıdadır ($i, j >$ orta).
3. Biri sol yarıda, diğeri sağ yarıdadır ($i \leq$ orta $< j$).

1 ve 2. durumlar rekürsif çağrılarla hesaplanır. Kritik nokta 3. durumdur: Sol ve sağ yarı sıralandıktan sonra birleştirme aşamasında, eğer $R[j] < L[i]$ ise, $L$ sıralı olduğu için $L$'de $i$'den sonra gelen *tüm* elemanlar da $R[j]$'den büyük olmak zorundadır. Dolayısıyla $R[j]$'yi birleşik diziye alırken, tek adımda sol dizide kalan eleman sayısı kadar ($|L| - i$) ters düzü sayaç değişkenimize ekleriz.

Böylece diziyi sıralarken aynı zamanda ters düz sayısını ek bir maliyet olmadan $O(n \log n)$ zamanda saymış oluruz. □

### 23.3.3 Hızlı Sıralama (Quick Sort)

Quick Sort, Merge Sort'un tersi bir felsefeyle çalışır:

- Merge Sort: Bölme işi önemsizdir (ortadan ikiye kır), asıl iş birleştirmededir.
- Quick Sort: Asıl iş bölmededir (bölümleme - partitioning), birleştirme işi ise ek işlem gerektirmez.

**Algoritmanın Mantığı:**

1. Diziden bir eleman seç: Buna **pivot** denir.
2. Diziyi öyle ikiye ayır ki (bölümleme): Pivotun solundakiler pivottan küçük veya eşit, sağındakiler ise pivottan büyük olsun. Pivot artık dizideki nihai ve doğru yerine oturmuştur.
3. Pivotun solunda ve sağında kalan alt dizileri aynı yöntemle sırala.

Bölümleme için yaygın iki yöntem vardır: Lomuto ve Hoare bölümleme algoritmaları. Burada anlaşılması en duru olan Hoare bölümleme yaklaşımına yakın two pointers yaklaşımını ele alacağız.

Sözde kod ile gösterimi:

```
FUNCTION quickSort(A)
    IF length(A) <= 1 THEN
        RETURN A
    END IF
```

```
    pivot = A[floor(length(A) / 2)]

    left = [x IN A WHERE x < pivot]
    middle = [x IN A WHERE x == pivot]
    right = [x IN A WHERE x > pivot]

    RETURN concatenate(quickSort(left), middle, quickSort(right))
END FUNCTION
```

Yukarıdaki kod ek bellek harcar; pratikte yarışmalarda Quick Sort *yerinde* (in-place, $O(1)$ ek bellek ile) yazılır:

```c
void swap(int *a, int *b) {
    int t = *a; *a = *b; *b = t;
}

int bolumle(int A[], int bas, int son) {
    int pivot = A[son]; // Son elemani pivot secelim
    int i = bas - 1;
    for (int j = bas; j < son; j++) {
        if (A[j] <= pivot) {
            i++;
            swap(&A[i], &A[j]);
        }
    }
    swap(&A[i + 1], &A[son]);
    return i + 1; // Pivotun nihai indeksi
}

void quick_sort_yerinde(int A[], int bas, int son) {
    if (bas < son) {
        int p = bolumle(A, bas, son);
        quick_sort_yerinde(A, bas, p - 1);
        quick_sort_yerinde(A, p + 1, son);
    }
}
```

### Quick Sort'un Zayıf Karnı: Kötü Pivot Seçimi

Pivot her adımda alt diziyi dengeli iki yarıya ($n/2$ ve $n/2$) bölerse:

$$T(n) = 2T(n/2) + O(n) \implies O(n \log n)$$

Ancak son elemanı pivot seçtiğimiz bir uygulamada, dizi zaten sıralıysa (örneğin $[1, 2, 3, 4, 5]$):

- Pivot 5 olur; sol taraf 4 elemanlı, sağ taraf 0 elemanlı kalır.
- Bir sonraki adımda pivot 4 olur; sol taraf 3 elemanlı kalır.

- Problem her adımda yalnızca 1 küçülür:

$$T(n) = T(n-1) + O(n) = O(n^2)$$

Bu durum yarışmalarda test senaryolarını hazırlayanların sıkça başvurduğu bir durumdur. Deterministik olarak hep ilk ya da hep son elemanı pivot seçen bir Quick Sort, özel olarak hazırlanmış sıralı veya tersten sıralı testlerde anında $O(n^2)$ karmaşıklığına geriler ve zaman sınırını aşar.

**Çözüm: Rastgele Pivot (Randomized Quick Sort)** Pivotu sabit bir indis yerine aralıktan rastgele seçilen bir indeks olarak belirlersek:

$$\text{pivot\_indeks} = \text{bas} + \text{rand}() \pmod{\text{son} - \text{bas} + 1}$$

ve bu elemanı son elemanla takas edip algoritmayı çalıştırırsak, hiçbir özel girdi algoritmayı doğrudan $O(n^2)$'ye zorlayamaz. Matematiksel beklenti değeri (expected value) her türlü girdide $O(n \log n)$ düzeyinde kalır.

## 23.4 Karmaşıklık ve Karakteristiklerin Karşılaştırması

Bir algoritmanın yalnızca asimptotik çalışma zamanını bilmek yarışmalarda doğru tercihi yapmak için tek başına yeterli olmayabilir. Algoritmanın bellek tüketimi, kararlılığı ve pratikteki sabit katsayıları belirleyici rol oynar.

### 23.4.1 Sıralamada Kararlılık (Stability)

**Tanım 23.15** (Kararlı Sıralama (Stable Sort))**.** *Eğer bir sıralama algoritması, sıralama anahtarları birbirine eşit olan elemanların dizideki göreceli giriş sırasını koruyorsa, bu algoritmaya* ***kararlı (stable)*** *sıralama denir.*

*Biçimsel olarak: $i < j$ ve $A[i] = A[j]$ iken, sıralama sonrasında da $A[i]$'den gelen eleman $A[j]$'den gelen elemanın solunda kalıyorsa algoritma kararlıdır.*

**Örnek 23.16.** *Elimizde öğrencilerin puanları ve isimleri olsun:*

$$[(Ali, 80), (Ayşe, 90), (Burak, 80), (Cem, 70)]$$

*Puana göre küçükten büyüğe sıralama yapıldığında:*

- ***Kararlı Sonuç:*** $[(Cem, 70), (Ali, 80), (Burak, 80), (Ayşe, 90)]$
- ***Kararsız Sonuç:*** $[(Cem, 70), (Burak, 80), (Ali, 80), (Ayşe, 90)]$

*Kararlı algoritmada Ali, Burak'tan önce geldiği için sıralama sonrasında da sırasını korumuştur. Çok kriterli sıralamalarda (örneğin önce sınıfa, sonra ada göre) kararlılık vazgeçilmezdir.*

- **Insertion Sort:** Kararlıdır (eşitlik durumunda eleman diğerinin soluna geçirilmez).
- **Bubble Sort:** Kararlıdır ($A[j] > A[j+1]$ katı büyüklük kontrolü yapıldığı sürece).
- **Merge Sort:** Kararlıdır (eşitlik durumunda sol dizideki eleman öncelikli seçilirse).
- **Quick Sort: Kararsızdır** (uzak mesafeli takaslar eşit elemanların sırasını bozar).

### 23.4.2 Büyük Tablo

| Algoritma | En İyi | Ortalama | En Kötü | Ek Bellek | Kararlı |
|---|---|---|---|---|---|
| Bubble Sort | $O(n)$ | $O(n^2)$ | $O(n^2)$ | $O(1)$ | Evet |
| Insertion Sort | $O(n)$ | $O(n^2)$ | $O(n^2)$ | $O(1)$ | Evet |
| Merge Sort | $O(n \log n)$ | $O(n \log n)$ | $O(n \log n)$ | $O(n)$ | Evet |
| Quick Sort | $O(n \log n)$ | $O(n \log n)$ | $O(n^2)$ | $O(\log n)$ | Hayır |

### 23.4.3 Hangi Sıralama Ne Zaman Kullanılır?

1. $n \leq 50$ **veya Veri Neredeyse Sıralıysa: Insertion Sort**. İlave bellek istemez, dallanma tahmin edicileri (branch predictor) ve önbellek (cache) dostudur, çok az komut koşturur.
2. **Garantili Süre ve Kararlılık Gerekiyorsa: Merge Sort**. Dış bellek sıralamalarında (verinin RAM'e sığmadığı durumlarda) ve bağlantılı listelerde (linked list) en uygun tercihtir.
3. **Genel Amaçlı ve Bellek Kısıtlıysa: Quick Sort**. Önbellek dostu ardışık erişimi ve küçük sabit katsayıları nedeniyle pratikte Merge Sort'tan 2-3 kat daha hızlı çalışır.
4. **Yarışmalarda Standart Kütüphane Kullanımı:** C++'taki `std::sort` fonksiyonu, Quick Sort ile başlar, rekürsiyon derinliği $\approx 2 \log n$'i aşarsa en kötü durum $O(n^2)$'den kaçınmak için Heap Sort'a geçer, alt diziler küçüldüğünde ise Insertion Sort'a döner (bu hibrit yaklaşıma **IntroSort** denir). C++'ta kararlı sıralama gerektiğinde ise `std::stable_sort` (temelde Merge Sort türevi) kullanılır.

**Örnek 23.17.** *Elimizde $n$ elemanlı bir dizi var. Bu dizideki en küçük $k$ elemanı sıralı olarak bulmak istiyoruz ($k \ll n$, örneğin $n = 10^6, k = 5$). Dizinin tamamını $O(n \log n)$ sürede sıralamak yerine nasıl bir yol izleyebiliriz?*

*Çözüm.* Tüm diziyi sıralamak gereksiz bir emektir; çünkü $k$'nın ötesindeki elemanların kendi arasındaki sırası bizi ilgilendirmemektedir.

Seçenek 1: Bubble Sort mantığıyla yalnızca $k$ geçiş yaparız. Her geçişte bir minimum eleman başa çekilir. Karmaşıklık $O(k \cdot n)$ olur. $k = 5$ için bu süre $5n$'dir; $n \log n$'den $(20n)$ belirgin biçimde daha hızlıdır.

Seçenek 2: Quick Sort'un bölümleme adımını tek taraflı çalıştırırız (QuickSelect). Pivot indeksi $p$ olsun:

- $p = k$ ise ilk $k$ eleman bulunmuştur, bu $k$ elemanı kendi içinde sıralarız $(O(k \log k))$.
- $p > k$ ise yalnızca sol tarafa dallanırız.
- $p < k$ ise yalnızca sağ tarafa dallanırız.

QuickSelect ortalama $n+n/2+n/4+\ldots = O(n)$ sürede ilk $k$ elemanı seçer. Üzerine $k$ elemanın sıralanması $O(k \log k)$ eklenirse toplam süre $O(n + k \log k)$ olur. □

**Örnek 23.18.** *Yalnızca* $0, 1$ *ve* $2$ *değerlerinden oluşan* $n$ *elemanlı bir diziyi tek bir geçişte, ek bellek kullanmadan* $(O(1))$ *nasıl sıralarsınız? (Hollanda Ulusal Bayrağı Problemi, Edsger W. Dijkstra)*

*Örnek girdi:* $A = [2, 0, 2, 1, 1, 0]$

*Çözüm.* Bölüm 22'de ele aldığımız two pointers tekniğini burada üç işaretçiye genişletiriz. Dizide yalnızca üç farklı değer vardır. Sıralı halde tüm 0'lar en solda, tüm 1'ler ortada, tüm 2'ler ise en sağda toplanmalıdır. Bu dağılım üç bölgeyi yöneten şu yapıyı gerektirir:

- $0 \ldots \text{sol} - 1$: kesinlikle 0'ların bölgesi.
- $\text{sol} \ldots \text{orta} - 1$: kesinlikle 1'lerin bölgesi.
- $\text{orta} \ldots \text{sag}$: henüz incelenmemiş elemanlar.
- $\text{sag} + 1 \ldots n - 1$: kesinlikle 2'lerin bölgesi.

Başlangıçta $\text{sol} = 0$, $\text{orta} = 0$, $\text{sag} = n - 1$. $\text{orta} \leq \text{sag}$ olduğu sürece $A[\text{orta}]$'ya bakarız:

1. $A[\text{orta}] == 0$: Bu eleman sol bölgeye aittir. $A[\text{orta}]$ ile $A[\text{sol}]$ takas edilir; $\text{sol}++$, $\text{orta}++$.
2. $A[\text{orta}] == 1$: Zaten ait olduğu orta bölgededir; sadece $\text{orta}++$.
3. $A[\text{orta}] == 2$: Bu eleman sağ bölgeye aittir. $A[\text{orta}]$ ile $A[\text{sag}]$ takas edilir; $\text{sag}--$. Dikkat: orta ilerlemez, çünkü sağdan takasla gelen yeni elemanın ne olduğunu henüz bilmiyoruz, bir sonraki adımda incelenmelidir.

Her adımda incelenmemiş bölgenin uzunluğu $(\text{sag} - \text{orta} + 1)$ kesinlikle 1 azalır. Dolayısıyla algoritma tam olarak $O(n)$ adımda biter, hiç karşılaştırma operatörü kullanmaz ve ek bellek harcamaz. □

Sıralama algoritmaları, yalnızca veriyi dizmekle kalmaz; ters düz sayıları, böl-ve-fethet tekniği, değişmez kavramı ve bilgi kuramı sınırları gibi bilgisayar biliminin temel analitik araçlarını bünyesinde barındırır. Sıradaki bölümlerde bu sıralı yapının üzerine inşa edeceğimiz arama ve veri yapısı tekniklerine zemin oluşturacaktır.

# Bölüm 24

# İkili Arama

Bilgisayar biliminde ve matematikte en sık karşılaşılan temel problem türlerinden biri, geniş bir aday kümesi içinden belirli bir koşulu sağlayan elemanı veya değeri bulmaktır. Doğal akla gelen ilk yöntem, tüm adayları tek tek inceleyen *doğrusal arama* (linear search) yaklaşımıdır. Ancak arama uzayının büyüklüğü $n$ olduğunda, $n = 10^9$ ya da $n = 10^{18}$ gibi yarışmalarda sıkça görülen boyutlara ulaşıldığında doğrusal tarama kabul edilemez bir zaman maliyetine yol açar.

İkili arama (*binary search*), yapılandırılmış (sıralı veya monoton) bir uzayda her bir adımla arama uzayını yarıya indiren, $O(n)$ maliyetli bir doğrusal aramayı $O(\log n)$ seviyesine düşüren oldukça etkili bir tekniktir. Bu bölümde ikili aramanın diziler üzerindeki temel işleyişini, Bölüm 18'de ele aldığımız değişmez (*invariant*) ilkesinin buradaki rolünü, sınır koşullarını ve ardından bu tekniği kapsamlı bir problem çözme aracına dönüştüren *cevap üzerinde ikili arama* yöntemini ele alacağız.

## 24.1 Sıralı Dizide Arama

### 24.1.1 Motivasyon ve Sezgi: Sayı Tahmin Oyunu

İkili aramanın arkasındaki temel mantığı kavramak için klasik bir oyunla başlayalım: Arkadaşınız 1 ile 100 arasında (sınırlar dahil) gizli bir $x$ tam sayısı tutmuş olsun. Her tahmininizde arkadaşınız size yalnızca üç cevaptan birini verebilir: "Daha büyük", "Daha küçük" veya "Doğru".

Bu oyunu en az hamlede garantili olarak bitirmek için nasıl bir strateji izlersiniz? Eğer $1, 2, 3, \ldots$ diye sırayla sorarsanız, arkadaşınız 100 tuttuğunda 100 soru sormanız gerekir. Fakat ilk tahmin olarak aralığın tam ortasındaki sayıyı, yani 50'yi seçerseniz:

- Arkadaşınız "Doğru" derse oyun 1 hamlede biter.

- Arkadaşınız "Daha büyük" derse, gizli sayının 1 ile 50 arasında olamayacağını kesin olarak anlarsınız. Aday kümeniz bir anda $\{51, 52, \ldots, 100\}$ aralığına iner. Aday sayısı 100'den 50'ye düşmüştür.
- Arkadaşınız "Daha küçük" derse, benzer şekilde aday kümeniz $\{1, 2, \ldots, 49\}$ olur.

Her tahminde aralığın ortasını seçerek geriye kalan arama uzayının boyutunu yarıya indirirsiniz. 100 aday ile başlayıp her seferinde ikiye böldüğümüzde:

$$100 \xrightarrow{1} 50 \xrightarrow{2} 25 \xrightarrow{3} 12 \xrightarrow{4} 6 \xrightarrow{5} 3 \xrightarrow{6} 1$$

En kötü durumda dahi 7 soruda gizli sayıyı kesinlikle bulursunuz. İkili aramanın özü tam olarak budur: Bilgi taşıyan tek bir karşılaştırma ile arama uzayının yarısını doğrudan elemek.

### 24.1.2 Sıralı Dizide Arama Mekanizması

Bu mantığı tamsayı dizilerine uygulayabilmemizin tek bir önkoşulu vardır: Dizinin *sıralı* olması (sıralama algoritmaları için Bölüm 23'e bakılabilir).

$n$ elemanlı artan sırada bir $A$ dizisi verilmiş olsun:

$$A[0] \leq A[1] \leq A[2] \leq \cdots \leq A[n-1]$$

Bu dizide belirli bir $hedef$ değerinin bulunup bulunmadığını kontrol etmek istiyoruz.

Arama yapacağımız geçerli indis aralığını $[sol, sag]$ olarak tanımlayalım. Başlangıçta hedef eleman dizinin herhangi bir yerinde olabileceğinden $sol = 0$ ve $sag = n - 1$ olarak kurulur.

Her adımda aralığın orta indisi hesaplanır:

$$orta = \left\lfloor \frac{sol + sag}{2} \right\rfloor$$

Bu noktada $A[orta]$ değeri ile $hedef$ karşılaştırılır:

1. **Eşitlik durumu ($A[orta] = hedef$):** Aranan eleman bulunmuştur; $orta$ indisi döndürülür.
2. **Hedef sağda ($A[orta] < hedef$):** Dizi küçükten büyüğe sıralı olduğundan, $orta$ indisindeki ve onun solundaki tüm elemanlar $hedef$'ten kesinlikle küçüktür:
$$A[i] \leq A[orta] < hedef \quad (\forall i \leq orta)$$
Dolayısıyla sol yarıda $hedef$'in bulunması imkânsızdır. Arama aralığı sağ yarıya daraltılır: $sol = orta + 1$.

3. **Hedef solda ($A[orta] > hedef$):** Benzer mantıkla, *orta* ve onun sağındaki tüm elemanlar $hedef$'ten kesinlikle büyüktür:

$$A[i] \geq A[orta] > hedef \quad (\forall i \geq orta)$$

Bu durumda sağ yarı elenir ve arama aralığı sol yarıya daraltılır: $sag = orta - 1$.

Bu işlem, eleman bulunana kadar veya arama aralığı boşalıncaya dek ($sol > sag$) devam eder.

**Tanım 24.1** (Arama Aralığı Değişmezi)**.** *İkili arama döngüsünün her adımının başında şu değişmez (invariant) korunur: Eğer $hedef$ değeri $A[0 \ldots n-1]$ dizisinde mevcutsa, kesinlikle $A[sol \ldots sag]$ alt dizisinin içindedir.*

**Teorem 24.2** (İkili Aramanın Zaman Karmaşıklığı)**.** *$n$ elemanlı sıralı bir dizide ikili arama algoritması en kötü durumda en fazla $\lfloor \log_2 n \rfloor + 1$ adet karşılaştırma yapar ve $O(\log n)$ zaman karmaşıklığı ile çalışır.*

*Kanıt.* Başlangıçtaki aralık boyutu $L_0 = n$'dir. Algoritmanın her adımında yeni aralık boyutu $L_{k+1} \leq \lfloor L_k/2 \rfloor$ olur. Aralık boyutu 1'e ulaştığında en fazla bir karşılaştırma daha yapılır. $k$ adım sonra aralık boyutu en fazla $\lfloor n/2^k \rfloor$ olur. Algoritmanın sonlanması için $\lfloor n/2^k \rfloor < 1$, yani $2^k > n$ olması yeterlidir. Buradan $k = \lfloor \log_2 n \rfloor + 1$ elde edilir. Her adımda yapılan işlemler (toplama, bölme, karşılaştırma) $O(1)$ sürede gerçekleştiğinden, toplam çalışma zamanı $O(\log n)$ olur. □

### 24.1.3 Somut Bir Yürütme Örneği

Algoritmanın adımlarını $n = 9$ elemanlı şu sıralı dizi üzerinde izleyelim:

$$A = [3,\ 7,\ 11,\ 15,\ 19,\ 24,\ 31,\ 42,\ 55]$$

Dizide $hedef = 24$ değerini arayalım.

- **1. Adım:** $sol = 0$, $sag = 8$.

$$orta = \lfloor (0 + 8)/2 \rfloor = 4 \implies A[4] = 19$$

$19 < 24$ olduğundan hedef sağ yarıdadır. Yeni sol sınır: $sol = orta + 1 = 5$.

- **2. Adım:** $sol = 5$, $sag = 8$.

$$orta = \lfloor (5 + 8)/2 \rfloor = 6 \implies A[6] = 31$$

$31 > 24$ olduğundan hedef sol yarıdadır. Yeni sağ sınır: $sag = orta - 1 = 5$.

- **3. Adım:** $sol = 5$, $sag = 5$.

$$orta = \lfloor (5+5)/2 \rfloor = 5 \implies A[5] = 24$$

$A[5] = 24 = hedef$ bulundu. Algoritma 3 karşılaştırmada 5 indisini döndürür.

Şimdi aynı dizide bulunmayan bir değeri, $hedef = 20$'yi arayalım:

- **1. Adım:** $sol = 0, sag = 8 \implies orta = 4$, $A[4] = 19 < 20 \implies sol = 5$.
- **2. Adım:** $sol = 5, sag = 8 \implies orta = 6$, $A[6] = 31 > 20 \implies sag = 5$.
- **3. Adım:** $sol = 5, sag = 5 \implies orta = 5$, $A[5] = 24 > 20 \implies sag = 4$.
- **4. Adım:** $sol = 5, sag = 4$. Artık $sol > sag$ şartı sağlandığından döngü biter. Elemanın dizide bulunmadığı $(-1)$ anlaşılır.

Dikkat edilirse, arama başarısız bittiğinde *sol* göstergesi (5), 20'den büyük en küçük elemanın indisini ($A[5] = 24$) işaret etmektedir. Bu gözlem bizi doğrudan bir sonraki kavrama götürür.

### 24.1.4 Alt ve Üst Sınır Aramaları (Lower Bound ve Upper Bound)

Yarışmalarda nadiren bir dizide yalnızca elemanın var olup olmadığı sorulur. Genellikle tekrarlı elemanlar içeren dizilerde şu iki sorguyla karşılaşırız:

1. **Alt sınır (*lower bound*):** Dizide değeri $\geq x$ olan ilk elemanın indisi.
2. **Üst sınır (*upper bound*):** Dizide değeri $> x$ olan ilk elemanın indisi.

Örneğin $A = [2, 4, 4, 4, 7, 9]$ dizisinde $x = 4$ için:

- $A[i] \geq 4$ koşulunu sağlayan ilk indis 1'dir ($A[1] = 4$).
- $A[i] > 4$ koşulunu sağlayan ilk indis 4'tür ($A[4] = 7$).

Bu iki indis arasındaki fark $(4 - 1 = 3)$, 4 değerinin dizide kaç kez tekrarlandığını doğrudan verir.

```
// Dizide eleman >= hedef olan ilk indisi dondurur.
// Eger boyle bir eleman yoksa n dondurur.
int alt_sinir(int A[], int n, int hedef) {
    int sol = 0, sag = n - 1;
    int sonuc = n;
    while (sol <= sag) {
        int orta = sol + (sag - sol) / 2;
        if (A[orta] >= hedef) {
```

```
            sonuc = orta;    // Aday bulundu, daha sola bak
            sag = orta - 1;
        } else {
            sol = orta + 1;  // Bu eleman cok kucuk, saga bak
        }
    }
    return sonuc;
}
```

```
FUNCTION lowerBound(A, target)
    left = 0
    right = length(A) - 1
    result = length(A)

    WHILE left <= right DO
        mid = left + floor((right - left) / 2)
        IF A[mid] >= target THEN
            result = mid
            right = mid - 1
        ELSE
            left = mid + 1
        END IF
    END WHILE

    RETURN result
END FUNCTION
```

### 24.1.5 Uygulamada En Sık Yapılan Hatalar

İkili arama algoritması kavramsal olarak çok basit görünse de, Donald Knuth'un belirttiği gibi doğru kodlanması şaşırtıcı derecede dikkat ister. Karşılaşılan tipik tuzaklar şunlardır:

1. **Tamsayı Taşması (Integer Overflow):** Orta indis hesaplanırken yazılan `(sol + sag) / 2` ifadesi, *sol* ve *sag* büyük değerler aldığında (örneğin $10^9$ mertebesinde) toplamlarının 32-bit işaretli tamsayı sınırını ($2^{31} - 1 \approx 2.14 \times 10^9$) aşmasına ve negatif bir sayıya dönüşmesine sebep olur. Güvenli yazım biçimi şudur:
$$orta = sol + \left\lfloor \frac{sag - sol}{2} \right\rfloor$$
2. **Sonsuz Döngü (Off-by-One Hataları):** Aralık daraltılırken `sol = orta` veya `sag = orta` yazıldığında, aralık boyutu 2 iken ($sol + 1 = sag$) tamsayı bölmesi tabana yuvarladığı için $orta = sol$ olur. Eğer koşul gereği $sol = orta$ dallanmasına girilirse, aralık hiç değişmez ve algoritma sonsuz döngüye girer. Aralık kapalı tutuluyorsa ($[sol, sag]$), adımlar kesinlikle $sol = orta + 1$ veya $sag = orta - 1$ olmalıdır.

3. **Aralık Kapanış Koşulu:** Döngü koşulu `while (sol <= sag)` iken aralık daraltması `sag = orta - 1` olmalıdır. Eğer döngü `while (sol < sag)` şeklinde yarı-açık aralık için yazılmışsa daraltma kuralları farklı tasarlanmalıdır. İki farklı yaklaşımı birbirine karıştırmak en yaygın hata kaynağıdır.

### 24.1.6 Çözümlü Örnekler

**Örnek 24.3.** *Sıralı bir dizi, bilinmeyen bir pivot indisi etrafında dairesel olarak döndürülmüştür (örneğin* $[0, 1, 2, 4, 5, 6, 7]$ *dizisi* $[4, 5, 6, 7, 0, 1, 2]$ *haline gelmiştir). Tüm elemanların birbirinden farklı olduğu bilindiğine göre, dizinin en küçük elemanını* $O(\log n)$ *zamanda bulunuz.*

*Çözüm.* Dizi bütünüyle sıralı olmasa da temel bir yapı korunmuştur: Dizi iki sıralı parçadan oluşur ve sağdaki parçanın tüm elemanları soldaki parçanın tüm elemanlarından küçüktür. Aranan en küçük eleman, bu iki parçanın kırılma noktasıdır.

Orta eleman $A[orta]$ ile sağ uç eleman $A[sag]$ karşılaştırıldığında:

- Eğer $A[orta] > A[sag]$ ise, minimum eleman kesinlikle $orta$'nın sağında kalmalıdır; çünkü standart sıralı bir dizide sol eleman sağdakinden büyük olamaz. Kırılma sağdadır: $sol = orta + 1$.
- Eğer $A[orta] < A[sag]$ ise, $orta$'dan $sag$'a kadar olan kısım düzenli artandır. Minimum eleman $A[orta]$ olabilir veya onun solunda yer alır: $sag = orta$.

Tüm elemanlar farklı olduğundan $A[orta] = A[sag]$ durumu sadece $sol = sag$ iken gerçekleşir.

Bu mantıkla çalışan algoritma her adımda aralığı yarıya indirir:

```
int minimum_bul(int A[], int n) {
    int sol = 0, sag = n - 1;
    while (sol < sag) {
        int orta = sol + (sag - sol) / 2;
        if (A[orta] > A[sag]) {
            sol = orta + 1;
        } else {
            sag = orta;
        }
    }
    return A[sol];
}
```

Her adımda aralık boyutu en az yarıya indiğinden karmaşıklık $O(\log n)$ olur. □

**Örnek 24.4.** *Bir* $A$ *dizisi "dağ dizisi" (mountain array) olarak adlandırılır eğer öyle bir* $k$ $(0 < k < n - 1)$ *indisi varsa ki:*

$$A[0] < A[1] < \cdots < A[k-1] < A[k] > A[k+1] > \cdots > A[n-1]$$

*Elemanları bu kurala uyan bir dizide zirve noktasının ($A[k]$) değerini $O(\log n)$ karşılaştırma ile bulunuz.*

*Çözüm.* Zirve noktasında dizinin komşu farkları işaret değiştirir. Herhangi bir *orta* indisi için $A[orta]$ ile $A[orta+1]$ elemanlarını karşılaştıralım:

- Eğer $A[orta] < A[orta+1]$ ise, henüz zirveye ulaşılmamıştır; dağın tırmanış yamacındayız. Zirve kesinlikle $orta+1$ veya daha sağdadır. Bu nedenle $sol = orta + 1$ yaparız.
- Eğer $A[orta] > A[orta+1]$ ise, iniş yamacındayız veya tam zirvedeyiz. Zirve *orta* olabilir veya solunda kalır. Dolayısıyla aralığı $sag = orta$ yaparak sınırlarız.

Aralık boyutu 1'e indiğinde ($sol = sag$), elimizde kalan tek indis kesinlikle zirvedir.

```
FUNCTION findPeak(A)
    left = 0
    right = length(A) - 1

    WHILE left < right DO
        mid = left + floor((right - left) / 2)
        IF A[mid] < A[mid + 1] THEN
            left = mid + 1
        ELSE
            right = mid
        END IF
    END WHILE

    RETURN A[left]
END FUNCTION
```

Arama uzayı her adımda küçüldüğünden çözüm $O(\log n)$ zamanda zirveyi bulur. □

## 24.2 Cevap Üzerinde İkili Arama

### 24.2.1 Motivasyon: Aramayı Genelleştirmek

İkili arama çoğunlukla bellekte doğrudan tanımlı bir dizide arama yapmak olarak ele alınır. Oysa yarışmalarda bu tekniğin asıl gücü **cevap üzerinde ikili arama** (*binary search on answer*) yaklaşımında ortaya çıkar.

Birçok optimizasyon problemi şu kalıpta karşımıza çıkar:

"Belirli kısıtları sağlayan **en küçük** (veya **en büyük**) $x$ değerini bulunuz."

Doğrudan en uygun $x$'i hesaplamak güç olabilir. Ancak problemi tersine çevirip şu karar sorusuna dönüştürürsek:

"Herhangi bir $x$ değeri verildiğinde, kısıtlar sağlanabilir mi?"

Bu kontrol sorusuna (genellikle Bölüm 27'deki greedy yaklaşımlarla veya bir simülasyonla) hızlı yanıt verilebiliyorsa ve kontrol fonksiyonu *monoton* bir davranış sergiliyorsa, aradığımız cevabı ikili arama ile bulabiliriz.

### 24.2.2 Monotonluk İlkesi

Bir $P(x)$ doğruluk fonksiyonu tanımlayalım: $P(x) \in \{\text{DOĞRU}, \text{YANLIŞ}\}$.

**Tanım 24.5** (Monoton Koşul)**.** *Bir $P(x)$ fonksiyonu, tüm $x \leq y$ değerleri için $P(x) \implies P(y)$ şartını sağlıyorsa monoton artandır. Benzer şekilde, tüm $x \leq y$ için $P(y) \implies P(x)$ şartı sağlanıyorsa fonksiyon monoton azalandır.*

Arama uzayındaki her $x$ değeri için $P(x)$ çıktısı sıralandığında şu iki desenden biri oluşmalıdır:

$$\text{YANLIŞ}, \text{YANLIŞ}, \ldots, \text{YANLIŞ}, \mathbf{\text{DOĞRU}}, \text{DOĞRU}, \ldots, \text{DOĞRU}$$

veya

$$\text{DOĞRU}, \text{DOĞRU}, \ldots, \mathbf{\text{DOĞRU}}, \text{YANLIŞ}, \text{YANLIŞ}, \ldots, \text{YANLIŞ}$$

Burada amaç, iki bölge arasındaki **sınır noktasını** (ilk DOĞRU veya son DOĞRU değerini) bulmaktır. Değerler dizisi bellekte peşinen yer almaz; her bir eleman, $P(x)$ fonksiyonu o anda çalıştırılarak üretilir.

**Teorem 24.6** (Cevap Üzerinde İkili Arama Doğruluğu)**.** *Eğer $P(x)$ fonksiyonu $[L, R]$ tamsayı aralığında monoton ise ve tek bir girdi için çalışma karmaşıklığı $O(f(n))$ ise, $P(x) = DOĞRU$ yapan en küçük (veya en büyük) $x$ değeri $O(f(n) \cdot \log(R - L))$ zamanda bulunabilir.*

*Kanıt.* Her adımda orta nokta $x_{orta} = \lfloor (L + R)/2 \rfloor$ hesaplanır ve $P(x_{orta})$ değerlendirilir. Monotonluk gereği:

- Eğer $P(x_{orta}) = \text{DOĞRU}$ ise, aranan ilk DOĞRU değeri $x_{orta}$ veya daha soldadır. Dolayısıyla sağ yarı elenir: $R = x_{orta} - 1$ (veya adayın saklanmasıyla).
- Eğer $P(x_{orta}) = \text{YANLIŞ}$ ise, aranan değer kesinlikle $x_{orta}$'dan büyüktür. Sol yarı elenir: $L = x_{orta} + 1$.

Aralık boyutu her adımda yarıya indiğinden en fazla $\lceil \log_2(R - L + 1) \rceil$ adet değerlendirme yapılır. Her değerlendirme $O(f(n))$ sürdüğünden toplam karmaşıklık $O(f(n) \log(R - L))$ olur. □

### 24.2.3 Çözümlü Örnekler

**Örnek 24.7** (Kablo Kesme Problemi)**.** *Elimizde uzunlukları sırasıyla $L_1, L_2, \ldots, L_n$ olan $n$ adet kablo bulunmaktadır. Bu kablolardan eşit uzunlukta en az $k$ adet parça elde etmek istiyoruz. Kablolar birbirine eklenemez, yalnızca kesilebilir. Elde edilebilecek bir parçanın tamsayı cinsinden **en büyük** uzunluğu ne kadar olabilir?*

*Veriler: $n = 4$, $k = 11$, kablo uzunlukları:* $[802, 743, 457, 539]$.

*Çözüm.* Doğrudan en büyük parça uzunluğu $x$'i cebirsel olarak kurmak yerine karar sorusuna odaklanalım: "Her bir parçanın uzunluğu $x$ seçilirse, toplamda en az $k$ parça elde edilebilir mi?"

Uzunluğu $L_i$ olan bir kablodan boyu $x$ olan en fazla $\lfloor L_i/x \rfloor$ adet parça kesilebilir. O halde $x$ uzunluğunda elde edilebilecek toplam parça sayısı:

$$C(x) = \sum_{i=1}^{n} \left\lfloor \frac{L_i}{x} \right\rfloor$$

olur. Kontrol fonksiyonumuz:

$$P(x) = (C(x) \geq k)$$

**Monotonluk Kontrolü:** Eğer parça uzunluğu $x$'ten daha büyük bir $y$ değerine çıkarılırsa ($y > x$), her kablodan elde edilen parça sayısı azalır veya sabit kalır ($\lfloor L_i/y \rfloor \leq \lfloor L_i/x \rfloor$). Dolayısıyla $C(y) \leq C(x)$ olur. Eğer $x$ uzunluğunda $k$ parça elde **edilemiyorsa**, daha uzun bir $y$ parçasında hiç elde edilemez. Yani doğruluk dizilimi:

$$\underbrace{\text{DOĞRU}, \ldots, \text{DOĞRU}}_{x \leq x^*}, \underbrace{\text{YANLIŞ}, \ldots, \text{YANLIŞ}}_{x > x^*}$$

biçimindedir; aranan monotonluk geçerlidir.

**Aralıkların Belirlenmesi:**

- Alt sınır: $sol = 1$ (parça uzunluğu en az 1 olmalıdır).
- Üst sınır: $sag = \max(L_i) = 802$ (hiçbir parça eldeki en uzun kablodan uzun olamaz).

Verilen örnek için ikili aramayı adım adım işletelim:

- $sol = 1, sag = 802 \implies orta = 401$:

  $C(401) = \lfloor 802/401 \rfloor + \lfloor 743/401 \rfloor + \lfloor 457/401 \rfloor + \lfloor 539/401 \rfloor = 2+1+1+1 = 5$

  $5 < 11$ olduğundan YANLIŞ. Demek ki 401 çok büyük, $sag = 400$.

- Birkaç adım sonra aralık daralır ve $orta = 200$ denenir:

$$C(200) = 4 + 3 + 2 + 2 = 11 \geq 11 \implies \text{DOĞRU}$$

  200 geçerli bir cevaptır. Ancak daha büyüğü olabilir mi? $cevap = 200$, $sol = 201$.

- Arama bu şekilde devam ettiğinde $x = 200$ için 11 parça, $x = 201$ için:

$$C(201) = 3 + 3 + 2 + 2 = 10 < 11 \implies \text{YANLIŞ}$$

  olduğu görülür.

Böylece ulaşılabilecek en büyük parça uzunluğu 200 olarak bulunur.

Karmaşıklık: Her adımda dizi bir kez taranır ($O(n)$) ve arama aralığı $[1, \max L]$ olduğundan toplam süre $O(n \log(\max L))$ olur. □

**Örnek 24.8** (Ressam Bölüştürme / Kitap Dağıtımı Problemi)**.** *$n$ adet kitap yan yana dizilmiştir. $i$. kitabın sayfa sayısı $A[i]$'dir. Bu kitaplar sırası bozulmadan $m$ öğrenciye paylaştırılacaktır. Her öğrenciye ardışık bir kitap bloğu verilmelidir ve her kitap tam olarak bir öğrenciye verilmelidir.*

*Bir öğrenciye düşen toplam sayfa sayısının **en fazla** olduğu değeri **en küçük** (minimax) yapacak dağıtımı bulunuz.*

*Dizi: $A = [12, 34, 67, 90]$, Öğrenci sayısı: $m = 2$.*

*Çözüm.* En çok sayfa alan öğrencinin yükünü olabildiğince azaltmak istiyoruz. Doğrudan tüm bölme noktalarını denemek yerine cevap üzerinde ikili arama uygulayalım: "Herhangi bir öğrenciye en fazla $S$ sayfa verilmesine izin verilirse, bu kitaplar en fazla $m$ öğrenciye dağıtılabilir mi?"

$P(S)$ kontrol fonksiyonunu Bölüm 27'de göreceğimiz greedy (*greedy*) yöntemle yazabiliriz: İlk öğrenciden başlayarak, $S$ limitini aşmadığı sürece kitapları sırayla veririz. Bir sonraki kitabı eklediğimizde toplam $S$'yi aşacaksa yeni bir öğrenciye geçeriz. Bir kitabın sayfa sayısı tek başına $S$'den büyükse hiçbir öğrenci bu kitabı alamaz ($P(S) = \text{YANLIŞ}$).

**Monotonluk:** Sayfa limiti $S$ artırılırsa öğrencilerin kapasitesi genişler ve gereken öğrenci sayısı azalır veya sabit kalır. Dolayısıyla $m$ öğrenciye yetme koşulu bozulmaz:

$$\text{YANLIŞ}, \ldots, \text{YANLIŞ}, \textbf{DOĞRU}, \ldots, \text{DOĞRU}$$

Hedefimiz ilk DOĞRU değerini veren en küçük $S$'yi bulmaktır.

**Aralıkların Belirlenmesi:**

- Alt sınır: En azından en kalın tekil kitabı bir öğrenci alabilmelidir: $sol = \max(A[i]) = 90$.

- Üst sınır: Tüm kitaplar tek bir öğrenciye verilseydi toplam sayfa sayısı gerekirdi: $sag = \sum A[i] = 12 + 34 + 67 + 90 = 203$.

Aramanın adımları:

- $sol = 90, sag = 203 \implies orta = 146$: Limit 146 olsun.
  - 1. Öğrenci: $12 + 34 + 67 = 113 \leq 146$. (90 eklenirse $203 > 146$, bitti).
  - 2. Öğrenci: $90 \leq 146$.

  Toplam 2 öğrenci yetti; $m = 2$ sınırı aşılmadı ($P(146) =$ DOĞRU). Daha küçük bir limit arayalım: $sag = 145$, $cevap = 146$.

- $sol = 90, sag = 145 \implies orta = 117$:
  - 1. Öğrenci: $12 + 34 + 67 = 113 \leq 117$.
  - 2. Öğrenci: $90 \leq 117$.

  Yine 2 öğrenci yetti ($P(117) =$ DOĞRU). $sag = 116$, $cevap = 117$.

- $sol = 90, sag = 116 \implies orta = 103$:
  - 1. Öğrenci: $12 + 34 = 46 \leq 103$ (67 eklenirse $113 > 103$).
  - 2. Öğrenci: $67 \leq 103$ (90 eklenirse $157 > 103$).
  - 3. Öğrenci: $90 \leq 103$.

  3 öğrenci gerekti; elimizde ise yalnızca 2 öğrenci var ($P(103) =$ YANLIŞ). Limit yetersiz kaldı, artırmalıyız: $sol = 104$.

- Kalan adımlar tamamlandığında sınırın 113 olduğu görülür: $S = 113$ için: $[12, 34, 67] \to 113$, $[90] \to 90$. Kitaplar tam 2 öğrenciye dağıtılır.

En küçük maksimum toplam 113'tür. □

### 24.2.4 Gerçel Sayılarda İkili Arama (Bisection Yöntemi)

İkili arama yalnızca tamsayılar üzerinde değil, sürekli fonksiyonların köklerini veya ekstremum noktalarını bulmada da kullanılır. Sayısal analizde bu yaklaşım *bisection* (ikiye bölme) yöntemi olarak adlandırılır.

Sürekli bir $f(x)$ fonksiyonu için $f(a)$ ve $f(b)$ zıt işaretli ise ($f(a) \cdot f(b) < 0$), Bolzano Teoremi gereği $[a, b]$ aralığında en az bir $x^*$ kökü ($f(x^*) = 0$) bulunmalıdır.

Gerçel sayılarda çalışırken iki önemli fark ortaya çıkar:

1. İndis artırımı ($+1$ veya $-1$) yapılmaz; doğrudan $sol = orta$ veya $sag = orta$ atanır.

2. Döngü sonlandırma koşulu iki türlü yazılabilir:

   - Hata payı (*tolerans*): $|sag - sol| > \epsilon$ (örneğin $\epsilon = 10^{-7}$).
   - Sabit adım sayısı: Doğrudan 100 veya 80 adımlık bir `for` döngüsü çalıştırmak. 80 adım sonra aralık boyutu $2^{-80} \approx 10^{-24}$ kat küçülür; bu da 64-bitlik `double` veri tipinin makine hassasiyetinden bile daha yüksek bir doğruluk sağlar. Yarışmalarda sabit adım sayısı kullanmak, kayan nokta hatalarından kaynaklanabilecek sonsuz döngüleri engellediği için daha güvenlidir.

**Örnek 24.9.** *$f(x) = x^3 + x - 5 = 0$ denkleminin gerçel kökünü $[1, 2]$ aralığında $10^{-6}$ hassasiyetle bulunuz.*

*Çözüm.* Fonksiyonun türevi $f'(x) = 3x^2 + 1 > 0$ olduğundan $f(x)$ tüm gerçel eksende kesin artandır (monotondur). Uç noktalarda:

$$f(1) = 1 + 1 - 5 = -3 < 0$$

$$f(2) = 8 + 2 - 5 = 5 > 0$$

Fonksiyon artan olduğundan ve işaret değiştirdiğinden $[1, 2]$ aralığında tam olarak bir kök vardır.

Algoritmayı C dilinde sabit döngü sayısı ile kuralım:

```c
#include <stdio.h>

double f(double x) {
    return x * x * x + x - 5.0;
}

double kok_bul() {
    double sol = 1.0, sag = 2.0;
    // 80 adim, araligi 2^{-80} oraninda daraltir.
    for (int adim = 0; adim < 80; adim++) {
        double orta = sol + (sag - sol) / 2.0;
        if (f(orta) <= 0.0) {
            sol = orta;  // Kok daha sagda veya tam orta noktada
        } else {
            sag = orta;  // Kok daha solda
        }
    }
    return sol;
}
```

Yaklaşık 1.515980 değeri aranan kök olarak bulunur. □

### 24.2.5 Kapsamlı Bir Problem: Agresif İnekler

Cevap üzerinde ikili arama yönteminin sık karşılaşılan klasik uygulamalarından biri "Agresif İnekler" (Aggressive Cows) problemidir.

**Örnek 24.10.** *Düz bir hat üzerinde bulunan $n$ adet ahırın koordinatları $x_1, x_2, \ldots, x_n$ tam sayılarıdır. Elimizde $c$ adet agresif inek bulunmaktadır. İnekleri bu ahırlara öyle yerleştirmek istiyoruz ki, birbirine en yakın iki inek arasındaki mesafe mümkün olan **en büyük** değere ulaşsın. Bu maksimum mesafeyi bulunuz.*

*Örnek girdi: $n = 5$ ahır, koordinatlar: $[1, 2, 8, 4, 9]$, inek sayısı: $c = 3$.*

*Çözüm.* Öncelikle Bölüm 23'teki sıralama yöntemlerinden biriyle koordinatları küçükten büyüğe dizelim:

$$x = [1, 2, 4, 8, 9]$$

Yerleştirilecek $c = 3$ ineğimiz var. Amacımız ardışık inekler arasındaki mesafelerin minimumu olan $d$'yi olabildiğince büyütmektir.

Soruyu karar biçimine dönüştürelim: "Herhangi iki inek arasındaki mesafe en az $d$ olacak şekilde $c$ adet ineği ahırlara yerleştirebilir miyiz?"

$P(d)$ kontrol fonksiyonunu greedy yaklaşımla kurabiliriz: İlk ineği daima en soldaki ahıra ($x[0]$) koymak en avantajlı seçimdir; bu tercih sonraki ineklere mümkün olan en geniş alanı bırakır. Sonraki her ineği, bir önceki ineğin konduğu ahırdan en az $d$ birim uzaktaki ilk uygun ahıra koyarız. Eğer tüm inekler yerleştirilebilirse $P(d) =$ DOĞRU, ahırlar yetersiz kalırsa $P(d) =$ YANLIŞ olur.

**Monotonluk:** Mesafe $d$ büyüdükçe inekler arasına bırakılması gereken boşluk genişler, bu da yerleşimi zorlaştırır. Belirli bir $d$ mesafesinde yerleşim yapılamıyorsa, daha büyük bir $d'$ mesafesinde hiç yapılamaz:

$$\text{DOĞRU}, \ldots, \textbf{DOĞRU}, \text{YANLIŞ}, \ldots, \text{YANLIŞ}$$

Monotonluk korunduğu için en büyük $d$ değeri ikili arama ile bulunabilir.

**Sınırlar:**

- $sol = 1$ (en küçük tamsayı mesafe).
- $sag = x[n-1] - x[0] = 9 - 1 = 8$ (en sol ile en sağ ahır arasındaki mesafe).

Adımları inceleyelim:

- $sol = 1, sag = 8 \implies orta = 4$: Mesafe en az 4 olabilir mi?
  - 1. İnek: $x[0] = 1$
  - 2. İnek: En az $1 + 4 = 5$ koordinatı aranır. İlk uygun ahır $x[3] = 8$.
  - 3. İnek: En az $8 + 4 = 12$ koordinatı aranır. Uygun ahır kalmadı.

3 inek yerleştirilemedi (yalnızca 2 inek yerleşti; $P(4) =$ YANLIŞ). Yeni sağ sınır: $sag = 3$.

- $sol = 1, sag = 3 \implies orta = 2$: Mesafe en az 2 olabilir mi?
  - 1. İnek: $x[0] = 1$
  - 2. İnek: $1 + 2 = 3 \leq x[2] = 4 \implies$ 4'e konur.
  - 3. İnek: $4 + 2 = 6 \leq x[3] = 8 \implies$ 8'e konur.

  3 inek başarıyla yerleşti ($P(2) =$ DOĞRU). Daha büyük bir değer aranabilir: $cevap = 2$, $sol = 3$.
- $sol = 3, sag = 3 \implies orta = 3$: Mesafe en az 3 olabilir mi?
  - 1. İnek: 1
  - 2. İnek: $1 + 3 = 4 \leq x[2] = 4 \implies$ 4'e konur.
  - 3. İnek: $4 + 3 = 7 \leq x[3] = 8 \implies$ 8'e konur.

  3 inek yerleşti ($P(3) =$ DOĞRU). $cevap = 3$, $sol = 4$.
- $sol = 4 > sag = 3 \implies$ Döngü tamamlandı.

Ulaşılabilecek en büyük minimum mesafe 3'tür (inekler $1, 4, 8$ konumlarına yerleşir).

Sıralama adımı $O(n \log n)$, ikili arama adımı $O(\log(x_{maks}))$, her kontrol ise $O(n)$ sürdüğünden toplam zaman karmaşıklığı $O(n \log n + n \log(x_{maks}))$ olur. □

### 24.2.6 Cevap Üzerinde İkili Arama Şablonu

Aşağıdaki şablon, koşulu sağlayan en büyük değeri bulma problemlerinde başvurulabilecek genel yapıyı özetlemektedir:

```
FUNCTION isValid(x, params)
    // Monotonic decision function
END FUNCTION

FUNCTION binarySearchOnAnswer(left, right, params)
    ans = left
    WHILE left <= right DO
        mid = left + FLOOR((right - left) / 2)
        IF isValid(mid, params) THEN
            ans = mid
            left = mid + 1
        ELSE
            right = mid - 1
        END IF
    END WHILE
    RETURN ans
END FUNCTION
```

Eğer problem koşulu sağlayan en küçük değeri arıyorsa, kontrol başarılı olduğunda `sag = orta - 1` yapılarak daha küçük değerlere bakılmalı, başarısız olduğunda `sol = orta + 1` ile değer büyütülmelidir.

### 24.2.7 Bölüm Özeti ve Problem Çözme Kontrol Listesi

Bir problemde ikili arama yönteminin uygulanabilirliğini değerlendirirken şu adımlar izlenebilir:

1. **Optimizasyon kalıbını tespit edin:** Soru bir büyüklüğün en büyük veya en küçük değerini mi arıyor?
2. **Karar problemine dönüştürün:** "Cevap $x$ olabilir mi?" sorusu üzerinden bir kontrol fonksiyonu tasarlayın.
3. **Monotonluğu doğrulayın:** $P(x)$ sağlandığında $P(x+1)$'in de zorunlu olarak sağlandığını (veya tam tersini) gösterin. Monotonluk kurulamazsa ikili arama uygulanamaz.
4. **Arama sınırlarını belirleyin:** *sol* ve *sag* değerlerini, aranan cevabı kesinlikle içerecek biçimde seçin.
5. **Taşma riskini kontrol edin:** Orta nokta hesabında $sol + (sag - sol)/2$ formülünü kullanın; büyük aralıklarda 64-bit tam sayılardan (`long long`) yararlanın.

# Bölüm 25

# Temel Veri Yapıları

Bölüm 22'de incelediğimiz diziler, bellekte ardışık yer kaplayan ve herhangi bir elemanına $O(1)$ zamanda doğrudan erişebildiğimiz temel yapılardır. Ancak problem çözerken veriyi yalnızca bir sıra halinde saklamak çoğu zaman yetmez; verilere hangi sırayla eriştiğimiz, hangisini ne zaman çıkarıp yerine yenisini eklediğimiz algoritmanın verimliliğini doğrudan belirler.

Bir kutu düşünelim: İçine üst üste kitaplar koyuyoruz. En son koyduğumuz kitabı en önce alabiliriz; en alttakini çekip almak içinse üsttekileri tek tek kaldırmak gerekir. Şimdi de bir fırın kuyruğunu göz önüne getirelim: İlk gelen ilk ekmeği alır, sıraya sonradan katılan en arkada bekler.

Burada, bilgisayar biliminin ve yarışma programlamasının temelini oluşturan üç veri yapısını ele alacağız: Son giren ilk çıkar mantığıyla çalışan **yığın** (stack), ilk giren ilk çıkar mantığıyla çalışan **kuyruk** (queue) ve eleman sorgulamalarını hızlandıran **küme** (set) ile **sözlük / eşlem** (map) yapıları.

## 25.1 Stack (Yığın)

### 25.1.1 Motivasyon: Tersine Çevirme ve Geri İzleme

Diyelim ki bir metin düzenleyicide yazı yazıyorsunuz. Klavyeden sırasıyla 'A', 'B', 'C' harflerine bastınız. Sonra bir hata yaptığınızı fark edip "Geri Al" (Undo) tuşuna bastınız. Sistem hangi işlemi geri almalıdır? Elbette en son yaptığınız işlemi, yani 'C' harfini yazmayı. İkinci kez geri alırsanız 'B' silinir.

İşte bu davranış biçimi—en son eklenen elemana ilk olarak ihtiyaç duyma durumu—günlük hayatta olduğu gibi algoritmalarda da sürekli karşımıza çıkar. Bu yapıyı düz bir diziyle yönetmeye kalktığımızda, her ekleme veya çıkarma işleminde dizinin kalan elemanlarını kaydırmak zorunda kalabiliriz; bu da işlem başına $O(n)$ maliyet getirir. Oysa bize yalnızca en üstteki elemanla ilgilenen ve tüm işlemlerini $O(1)$ zamanda tamamlayan bir yapı gerekir.

### 25.1.2 Kavramsal Tanım ve LIFO Prensibi

**Tanım 25.1** (Yığın / Stack)**.** *Yığın (stack), eleman ekleme ve çıkarma işlemlerinin yalnızca tek bir uçtan—**tepe** (top) noktasından—yapıldığı doğrusal bir veri yapısıdır. Bu işleyiş kuralına **LIFO** (Last-In, First-Out — Son Giren İlk Çıkar) prensibi denir.*

Bir yığının sunduğu temel işlemler şunlardır:

- push($x$): $x$ elemanını yığının tepesine koyar.
- pop(): Yığının tepesindeki elemanı kaldırır ve bu elemanı döndürür (yığın boşken yapılırsa taşma/hata oluşur).
- top() veya peek(): Yığının tepesindeki elemana bakar, fakat onu yığından çıkarmaz.
- empty(): Yığının boş olup olmadığını kontrol eder (boşsa doğru, değilse yanlış).
- size(): Yığındaki anlık eleman sayısını verir.

**Teorem 25.2** (Yığın İşlemlerinin Zaman Karmaşıklığı)**.** *Boyutu dinamik olarak ayarlanabilen bir dizi (veya bağlı liste) ile gerçeklenen bir yığında;* push*,* pop*,* top*,* empty *ve* size *işlemlerinin her biri en kötü durumda ya da amortize edilmiş olarak* $O(1)$ *zamanda çalışır.*

*Kanıt.* Bir $A$ dizisi ve yığının tepe noktasının indisini tutan bir $t$ işaretçisi (pointer) tanımlayalım. Başlangıçta yığın boştur, $t = -1$.

- push($x$) işleminde önce $t \leftarrow t + 1$ yapılır, ardından $A[t] \leftarrow x$ atanır. Dizi indisine doğrudan erişim $O(1)$ adımdır.
- pop() işleminde $A[t]$ değeri saklanır, ardından $t \leftarrow t - 1$ yapılır. Bu işlem de $O(1)$ adımdır.
- top() işlemi doğrudan $A[t]$ değerini okur ($O(1)$).
- Boyut $t + 1$ formülüyle $O(1)$ zamanda hesaplanır; boşluk kontrolü $t == -1$ ifadesidir.

Hiçbir işlemde dizinin diğer elemanlarını kaydırmak gerekmediğinden tüm işlemler $O(1)$ zamanda tamamlanır. □

### 25.1.3 Kullanım Senaryoları ve Çözümlü Örnekler

Yığın yapısı matematikte ve bilgisayar biliminde özellikle şu problemlerde sıkça kullanılır:

1. **Parantez Eşleme (Balancing Symbols):** İç içe açılan parantezlerin doğru sırada kapanıp kapanmadığını denetleme.
2. **Monoton Yığın (Monotonic Stack):** Bir dizideki her eleman için kendisinden sonraki "ilk büyük" veya "ilk küçük" elemanı bulma.
3. **İfade Değerlendirme (Expression Parsing):** Postfix (ters Leh notasyonu) ifadeleri hesaplama.
4. **Çağrı Yığını (Call Stack):** Rekürsif fonksiyonların çalışma mekanizması (Bölüm 30'da göreceğimiz fonksiyon çağrıları bellekte tam olarak bir yığın gibi saklanır).

Şimdi bu kullanım alanlarını somut örneklerle ele alalım.

**Örnek 25.3** (Dengeli Parantez Dizisi)**.** *Sadece '(', ')', '[', ']', '{', '}' karakterlerinden oluşan bir metin veriliyor. Bu parantez diziliminin geçerli (dengeli) olup olmadığını belirleyiniz. Örneğin:*

- *$S$ = "{[()]}" geçerlidir.*
- *$S$ = "[(])" geçersizdir (çünkü '[' kapatılmadan önce ')' gelmiştir).*
- *$S$ = "(()" geçersizdir (açık kalan parantez vardır).*

*Çözüm ve Akıl Yürütme.* Parantezlerin kapanma kuralı şudur: Bir kapama parantezi geldiğinde, en son açılmış olan ve henüz eşleşmemiş olan parantez ile aynı türde olmalıdır. "En son açılan" şartı doğrudan LIFO mantığını, yani yığını işaret eder.

Metni soldan sağa tarayalım:

1. Bir açma parantezi ('(', '[', '{') gördüğümüzde, ileride kapatılmak üzere yığına push ederiz.
2. Bir kapama parantezi (')', ']', '}') gördüğümüzde:
   - Yığın boşsa, kapatılacak hiçbir açma parantezi yoktur; dizi dengesizdir.
   - Yığın boş değilse, tepedeki elemana bakarız (top). Bu eleman gelen kapama parantezinin türdeşi ise yığından çıkarırız (pop). Değilse (örneğin tepede '[' varken ')' geldiyse), sıra bozulmuştur; dizi geçersizdir.
3. Metin bittiğinde yığında hâlâ kapatılmamış parantez kalmışsa (örneğin "(()" durumundaki ilk parantez), dizi yine geçersizdir. Yalnızca yığın tamamen boşaldıysa dizi dengelidir.

$n$ karakterlik bir metinde her karakter yığına en fazla 1 kez girer ve en fazla 1 kez çıkar. Dolayısıyla toplam zaman karmaşıklığı $O(n)$, alan karmaşıklığı ise en kötü durumda $O(n)$ olur. □

**Örnek 25.4** (Sonraki Büyük Eleman - Next Greater Element)**.** $A = [4, 5, 2, 25]$ *dizisi verilsin. Dizideki her eleman için, kendisinden sonra (sağında) gelen ve kendisinden kesinlikle büyük olan ilk elemanı bulunuz. Eğer sağında kendisinden büyük bir eleman yoksa* $-1$ *yazınız.*

*Çözüm ve Akıl Yürütme.* Kaba kuvvet yaklaşımı, her $i$ elemanı için sağdaki $j > i$ indislerini tek tek taramaktır; bu da en kötü durumda $O(n^2)$ karşılaştırma gerektirir. $n = 10^5$ gibi tipik yarışma sınırlarında bu yöntem zaman aşımına uğrar.

Sorunu tersten ele alalım (Bölüm 19'daki tersine çalışma yaklaşımıyla): Diziyi sağdan sola doğru tarayalım. Diyelim ki $A[i]$ elemanındayız. Sağımızdaki elemanlar arasından bize en yakın ve bizden büyük olanı arıyoruz. Eğer sağda kalan bazı elemanlar $A[i]$'den küçük ya da ona eşitse, $A[i]$'nin solundaki herhangi bir eleman için bu küçük elemanlar asla "ilk büyük eleman" olamaz; çünkü $A[i]$ hem daha soldadır hem de onlardan daha büyüktür. Küçük elemanlar artık $A[i]$'nin ardında gizlenmiştir.

O halde sağdan sola ilerlerken elemanları bir yığında tutalım; öyle ki yığın tepeden aşağıya doğru kesin artan sırada kalsın (buna **monoton yığın** denir):

1. $i = 3$, $A[3] = 25$: Yığın boştur. Sağında büyük eleman yoktur $\rightarrow -1$. Yığına 25 eklenir. (Yığın: $[25]$).
2. $i = 2$, $A[2] = 2$: Tepedeki eleman $25 > 2$. Demek ki 2'den sonraki ilk büyük eleman 25'tir. Yığına 2 eklenir. (Yığın: $[25, 2]$).
3. $i = 1$, $A[1] = 5$: Tepedeki eleman $2 \leq 5$'tir; 2 elenir (pop). Yeni tepe $25 > 5$'tir. Demek ki 5'ten sonraki ilk büyük eleman 25'tir. Yığına 5 eklenir. (Yığın: $[25, 5]$).
4. $i = 0$, $A[0] = 4$: Tepedeki eleman $5 > 4$'tür. Cevap 5. Yığına 4 eklenir. (Yığın: $[25, 5, 4]$).

Sonuç dizisi: $[5, 25, 25, -1]$ bulunur.

**Karmaşıklık Analizi:** İç içe bir döngü varmış gibi görünse de, $n$ elemanın her biri yığına tam 1 kez eklenir (push) ve en fazla 1 kez çıkarılır (pop). Toplam yığın işlemi sayısı $2n$'i aşamaz; dolayısıyla algoritma $O(n)$ zamanda çalışır. □

```c
// C dilinde dizi ile Monoton Yigin yaklasimi
#include <stdio.h>

void sonrakiBuyukEleman(int A[], int n, int sonuc[]) {
    int yigin[n];
    int top = -1; // Yigin bos
```

```
    for (int i = n - 1; i >= 0; i--) {
        // Guncel elemandan kucuk veya esit olanlari yigindan at
        while (top >= 0 && yigin[top] <= A[i]) {
            top--;
        }
        // Yigin bossa saginda buyuk eleman yoktur
        if (top == -1) {
            sonuc[i] = -1;
        } else {
            sonuc[i] = yigin[top];
        }
        // Guncel elemani yigina ekle
        yigin[++top] = A[i];
    }
}
```

**Örnek 25.5** (Aritmetik İfadeleri Hesaplama - Postfix Notasyonu)**.** *Günlük kullanımda tercih ettiğimiz* $3 + 4 \times 2$ *gösterimi* ***infix*** *(içten) notasyondur ve işlem önceliğini belirtmek için parantez gerektirebilir. Bilgisayarlar için parantez ihtiyacını ortadan kaldıran* ***postfix*** *(sondan) notasyon çok daha uygundur. Aynı işlem postfix biçiminde şöyle ifade edilir:*

*"3 4 2 * +"*

*Bu ifadenin değerini hesaplayınız.*

*Çözüm ve Akıl Yürütme.* Soldan sağa okuma yaparken bir sayı gördüğümüzde henüz hangi işleme gireceğini bilmediğimizden sayıları biriktiririz. Bir operatör ($+$, $*$) ile karşılaştığımızda ise bu işlem, en son gördüğümüz iki sayıya uygulanmalıdır. Son görülen iki sayıyı geri çağırma ihtiyacı doğrudan LIFO düzenini gerektirir:

1. '3' oku: yığına at. (Yığın: [3])
2. '4' oku: yığına at. (Yığın: [3, 4])
3. '2' oku: yığına at. (Yığın: [3, 4, 2])
4. '*' oku: İki eleman çek: $b = 2$, $a = 4$. Çarp: $a \times b = 4 \times 2 = 8$. Sonucu yığına at. (Yığın: [3, 8])
5. '+' oku: İki eleman çek: $b = 8$, $a = 3$. Topla: $a + b = 3 + 8 = 11$. Sonucu yığına at. (Yığın: [11])

İfade bittiğinde yığında kalan son eleman (11), işlemin sonucudur. $n$ terimli bir ifade $O(n)$ sürede hesaplanır. □

```
FUNCTION evaluatePostfix(expression)
    stack = EMPTY_STACK
    tokens = SPLIT expression BY whitespace

    FOR EACH token IN tokens DO
        IF token IS "+", "-", "*", OR "/" THEN
            b = POP(stack)
            a = POP(stack)
            IF token == "+" THEN
                PUSH(stack, a + b)
            ELSE IF token == "-" THEN
                PUSH(stack, a - b)
            ELSE IF token == "*" THEN
                PUSH(stack, a * b)
            ELSE IF token == "/" THEN
                PUSH(stack, TRUNCATE(a / b))
            END IF
        ELSE
            PUSH(stack, TO_INTEGER(token))
        END IF
    END FOR

    RETURN stack[0]
END FUNCTION
```

**En Sık Yapılan Hatalar (Stack):**

- **Boş Yığından Eleman Çekmek (Stack Underflow):** Yığının boş olup olmadığını (empty()) kontrol etmeden top() veya pop() çağırmak tanımsız davranışa ve programın çökmesine yol açar. Özellikle parantez eşleme problemlerinde kapama parantezi başta geldiğinde bu hataya sık rastlanır.
- **İşlem Sırasını Karıştırmak:** Postfix çıkarmada yığından ilk çekilen eleman ikinci terimdir ($b$), sonra çekilen birinci terimdir ($a$). $a - b$ yerine $b - a$ yazmak sonucu tamamen bozar.

## 25.2 Queue (Kuyruk)

### 25.2.1 Motivasyon: Sıralı İşleme ve Genişlik Öncelikli Arama

Bir bilet gişesi düşünelim: İnsanlar sıraya girer ve gişedeki görevli ilk kim geldiyse önce ona hizmet verir; araya girmek ya da son gelene öncelik tanımak sıranın düzenini bozar.

Bilgisayar biliminde bir görevi geliş sırasına göre işleme alma ihtiyacı doğduğunda dizi üzerinde eleman kaydırmak ciddi bir verim kaybına yol açar. Örneğin

sıradan bir dizide en öndeki elemanı çıkardığınızda, arkadaki tüm elemanları birer basamak sola kaydırmanız gerekir; bu da $O(n)$ zaman alır. Bize hem en arkaya eklemenin hem de en önden çıkarmanın $O(1)$ olduğu bir veri yapısı lazımdır.

### 25.2.2 Kavramsal Tanım ve FIFO Prensibi

**Tanım 25.6** (Kuyruk / Queue)**.** *Kuyruk (queue), elemanların yalnızca bir uçtan—**arka** (rear/back)—eklendiği ve diğer uçtan—**ön** (front)—çıkarıldığı doğrusal bir veri yapısıdır. Bu işleyişe **FIFO** (First-In, First-Out — İlk Giren İlk Çıkar) prensibi denir.*

Temel kuyruk işlemleri:

- push($x$) veya enqueue($x$): $x$ elemanını kuyruğun sonuna ekler.
- pop() veya dequeue(): Kuyruğun en önündeki elemanı çıkarır.
- front(): Kuyruğun başındaki elemanı okur.
- empty(): Kuyrukta eleman olup olmadığını bildirir.
- size(): Kuyruktaki toplam eleman sayısını verir.

### 25.2.3 Dairesel Dizi (Circular Array) ile Kuyruk Tasarımı

Düz bir dizide baştan eleman sildikçe önceki bellek alanları atıl kalır. Ön işaretçi ($front$) sürekli sağa kayar ve dizi sınırına ulaşıldığında yer kalmamış gibi görünür. Bu problemi çözmek için diziyi bir çember gibi düşünebiliriz: İndis dizinin sonuna $(C-1)$ ulaştığında, Bölüm 14'te ele aldığımız modüler aritmetik sayesinde sıradaki indis tekrar 0'a döner.

**Tanım 25.7** (Dairesel Kuyruk İndis Aritmetiği)**.** *Kapasitesi $C$ olan bir dizide, bir $i$ indisinin ardılı:*

$$\text{sonraki}(i) = (i + 1) \pmod{C}$$

*formülüyle elde edilir.*

**Teorem 25.8** (Dairesel Kuyruk İşlemleri Karmaşıklığı)**.** *Sabit kapasiteli $C$ boyutunda bir dairesel dizi üzerinde gerçeklenen kuyrukta; ekleme, silme ve öndeki elemanı okuma işlemleri en kötü durumda $O(1)$ zamanda çalışır.*

*Kanıt.* $front$ (başlangıç indisi), $rear$ (bitiş indisi) ve anlık eleman sayısını tutan $size$ değişkenlerini saklayalım:

- enqueue($x$): Eğer $size == C$ ise kuyruk doludur (taşma). Aksi halde $rear \leftarrow (rear + 1) \pmod{C}$ yapılır, $A[rear] \leftarrow x$ yazılır ve $size \leftarrow size + 1$ artırılır. Bu adımların her biri $O(1)$ aritmetik işlemidir.

- dequeue(): Eğer $size == 0$ ise kuyruk boştur. Aksi halde $front \leftarrow (front + 1) \pmod{C}$ yapılır ve $size \leftarrow size - 1$ azaltılır. Bellek kaydırma yapılmaz, işlem $O(1)$ sürer.

Dolayısıyla her iki temel işlem de $O(1)$ zamanda çalışır. □

```c
// Sabit boyutlu dairesel kuyruk yapisi
#define KAPASITE 100

typedef struct {
    int veri[KAPASITE];
    int bas;
    int son;
    int elemanSayisi;
} DaireselKuyruk;

void kuyrukBaslat(DaireselKuyruk *q) {
    q->bas = 0;
    q->son = -1;
    q->elemanSayisi = 0;
}

void enqueue(DaireselKuyruk *q, int x) {
    if (q->elemanSayisi == KAPASITE) return; // Dolu
    q->son = (q->son + 1) % KAPASITE;
    q->veri[q->son] = x;
    q->elemanSayisi++;
}

int dequeue(DaireselKuyruk *q) {
    if (q->elemanSayisi == 0) return -1; // Bos
    int deger = q->veri[q->bas];
    q->bas = (q->bas + 1) % KAPASITE;
    q->elemanSayisi--;
    return deger;
}
```

### 25.2.4 Kullanım Senaryoları ve Çözümlü Örnekler

Kuyruk veri yapısı özellikle şu senaryoların temel aracıdır:

1. **Genişlik Öncelikli Arama (Breadth-First Search - BFS):** Bir başlangıç noktasından itibaren tüm düğümleri en yakın olandan başlayarak katman katman ziyaret etme.

2. **Kayan Pencere Problemleri (Sliding Window):** Zaman içinde gelen ve belli bir pencere genişliğinde saklanması gereken verileri yönetme.

3. **Sıra Tabanlı Simülasyonlar:** İşletim sistemlerinde süreç (process) zamanlama, yazıcı kuyrukları vb.

**Örnek 25.9** (İkili Sayıları Sırayla Üretme)**.** *1'den $n$'e kadar olan tamsayıların ikili (binary) tabandaki karşılıklarını küçükten büyüğe sırayla ekrana yazdıracak bir algoritma tasarlayınız. Örneğin $n = 5$ için çıktı:* `1, 10, 11, 100, 101` *olmalıdır.*

*Çözüm ve Akıl Yürütme.* Her sayıyı tek tek alıp 2'ye bölerek ikiliye çevirmek $O(n \log n)$ sürer. Oysa üretim mantığına dikkat edersek:

- "1" sayısının arkasına '0' eklersek "10" (2), '1' eklersek "11" (3) oluşur.
- "10" sayısının arkasına '0' eklersek "100" (4), '1' eklersek "101" (5) oluşur.

Üretilen her kök sayıdan iki yeni sayı doğar ve önce üretilen sayı önce işlenmelidir. Bu tam olarak bir FIFO (kuyruk) modelidir.

1. Kuyruğa ilk olarak `"1"` metnini atarız.
2. $n$ defa şu işlemi tekrarlarız:
   - Kuyruğun başındaki elemanı çıkar: $s = \text{pop}()$.
   - $s$ değerini yazdır.
   - Kuyruğa $s +$ "0" ekle.
   - Kuyruğa $s +$ "1" ekle.

Her adımda yalnızca $O(1)$ kuyruk işlemi yapıldığından toplam süre $O(n)$ olur. □

**Örnek 25.10** (Josephus Problemi (Flavius Josephus, MS 1. yüzyıl))**.** *$n$ kişi $(1, 2, \ldots, n)$ bir çember oluşturacak şekilde dizilmiştir. 1 numaralı kişiden başlanarak sayılır ve her $k$'ıncı kişi çemberden çıkarılır. Sayma işlemi son bir kişi kalana kadar bir sonraki kişiden devam eder. Hayatta kalan son kişiyi bulunuz. ($n = 5, k = 2$ için: 2 elenir, 4 elenir, 1 elenir, 5 elenir, geriye 3 kalır).*

*Çözüm ve Akıl Yürütme.* Çember etrafındaki dönme hareketini bir kuyrukla doğrudan modelleyebiliriz: Sırası gelen kişi elenmeyecekse kuyruğun önünden çıkıp tekrar kuyruğun arkasına girer. Elenecek kişi ise kuyruktan çıkarılır ve bir daha eklenmez.

1. Kuyruğa $1, 2, \ldots, n$ sayılarını ekleriz.
2. Kuyrukta 1'den fazla eleman olduğu sürece:
   - $k - 1$ defa: Kuyruğun başındaki elemanı çıkar ve hemen arkasına ekle (push(pop())).
   - $k$'ıncı elemanı kuyruktan tamamen çıkar (pop()).

3. Kuyrukta tek bir kişi kaldığında o kişi kazanan kişidir.

Her eleme için $k$ adım atılır, toplam $n-1$ eleme yapıldığından bu simülasyon $O(n \cdot k)$ zamanda çalışır. □

```
FUNCTION josephus(n, k)
    CREATE queue q
    FOR i FROM 1 TO n DO
        ENQUEUE(q, i)
    END FOR

    WHILE LENGTH(q) > 1 DO
        FOR step FROM 1 TO k - 1 DO
            ENQUEUE(q, DEQUEUE(q))
        END FOR
        DEQUEUE(q)
    END WHILE

    RETURN PEEK(q)
END FUNCTION
```

**Örnek 25.11** (İki Yığın ile Bir Kuyruk Oluşturma)**.** *Elinizde yalnızca iki adet yığın ($Y_1$ ve $Y_2$) bulunmaktadır. Bu iki yığını kullanarak $O(1)$ amortize edilmiş zamanda çalışan bir kuyruk (FIFO) veri yapısı inşa ediniz.*

*Çözüm ve Akıl Yürütme.* Bir yığına eleman eklediğimizde sıralama tersine döner ($A, B \to B, A$). Tersine dönmüş bir yığındaki elemanları alıp ikinci bir yığına aktarırsak sıra bir kez daha tersine döner; yani baştaki haline $(A, B)$ geri gelir. İki LIFO yapısının peş peşe işletilmesi böylece bir FIFO oluşturur.

Yapı şöyle çalışır:

- **Ekleme (**enqueue($x$)**):** Daima $Y_1$'e push($x$) yapılır. Bu işlem kesinlikle $O(1)$ sürer.
- **Çıkarma (**dequeue()**):** En eski elemana ihtiyacımız vardır, oysa o eleman $Y_1$'in tabanındadır.
  - Eğer $Y_2$ boş değilse, en eski eleman zaten $Y_2$'nin tepesindedir. Doğrudan $Y_2.\,\text{pop}()$ yaparız ($O(1)$).
  - Eğer $Y_2$ boşsa, $Y_1$'deki tüm elemanları teker teker çekip $Y_2$'ye aktarırız. Böylece $Y_1$'in tabanındaki eleman $Y_2$'nin tepesine gelir. Ardından $Y_2.\,\text{pop}()$ yaparız.

**Amortize Analiz:** Bir eleman sisteme girerken $Y_1$'e 1 kez eklenir. Yapı içinde kaldığı süre boyunca en fazla 1 kez $Y_1$'den $Y_2$'ye aktarılır ve 1 kez $Y_2$'den çıkarılır. Her eleman toplamda en fazla 3 yığın işlemine tabi tutulur. $n$ işlem için toplam maliyet $O(n)$ olduğundan işlem başına amortize maliyet $O(1)$'dir. □

**En Sık Yapılan Hatalar (Queue):**

- **Düz Dizide Kaydırma Yaparak Kuyruk Yazmak:** Kuyruktan eleman çıkarmayı dizinin 0'ıncı elemanını silip döngüyle tüm diziyi sola kaydırarak kodlamak, her silme için $O(n)$ maliyet doğurur. $n$ eleman için toplam süre $O(n^2)$'ye çıkar ve zaman aşımına yol açar. Çözüm, dairesel indisler veya çift uçlu kuyruk (`deque`) kullanmaktır.
- **Dairesel Dizide Doluluk-Boşluk Ayrımı:** $front == rear$ durumu hem kuyruğun tamamen boş hem de tamamen dolu olduğu ana denk gelebilir. Bu belirsizliği gidermek için ya bir *size* sayacı tutulmalı ya da kapasiteden bir hücre daima boş bırakılmalıdır.

## 25.3 Set ve Map (Küme ve Sözlük / Eşlem)

### 25.3.1 Motivasyon: Hızlı Arama ve Frekans Sayma

Diyelim ki $10^5$ kelimeden oluşan bir sözlüğünüz var. Kullanıcının girdiği bir kelimenin bu sözlükte yer alıp almadığını kontrol etmek istiyorsunuz. Kelimeleri gelişigüzel bir dizide tutarsanız, her sorgu için diziyi baştan sona taramanız gerekir ($O(n)$). $10^5$ sorgu yapıldığında işlem sayısı $10^{10}$ seviyesine ulaşır ve program gecikir.

Benzer şekilde, bir dizide en çok tekrar eden sayıyı (mod) bulmak istediğimizi düşünelim. Sayıların her birinin kaçar kez geçtiğini (frekansını) nasıl sayarız? Sayılar 0 ile 100 arasındaysa dizi indisi yeterlidir. Peki ya sayılar $10^9$ gibi büyük değerlerse veya tamsayı yerine metinlerden oluşuyorsa?

İşte "Bir eleman bu topluluğa ait mi?" ve "Bu anahtara karşılık gelen değer nedir?" soruları, bilgisayar biliminde **Set** (Küme) ve **Map** (Eşlem / Sözlük) veri modelleriyle çözülür.

### 25.3.2 Kavramsal Modeller

**Tanım 25.12** (Set / Küme)**.** *Set (küme), Bölüm 1'de ele aldığımız küme kavramının bilgisayar bilimindeki karşılığıdır. İçerisinde* ***aynı elemandan birden fazla bulunamaz*** *(her eleman tekildir) ve temel görevi bir elemanın kümede* ***var olup olmadığını*** *sorgulamaktır (∈).*

Bir kümenin temel işlemleri:

- insert($x$): $x$ elemanını kümeye ekler (eleman zaten varsa küme değişmez).
- erase($x$) veya remove($x$): $x$ elemanını kümeden çıkarır.
- contains($x$) veya count($x$): $x$ elemanının kümede bulunup bulunmadığını kontrol eder (1 veya 0).

**Tanım 25.13** (Map / Eşlem / Sözlük)**.** *Map (eşlem veya sözlük), her biri bir **anahtar** (key) ve bu anahtara bağlı bir **değer** (value) çiftinden oluşan $(k, v)$ ikililerini saklayan bir yapıdır. Bir eşlemde her anahtar **tekildir**.*

Bir eşlemin temel işlemleri:

- set$(k, v)$ veya $M[k] = v$: $k$ anahtarına $v$ değerini atar. $k$ zaten varsa eski değeri güncellenir.
- get$(k)$ veya $M[k]$: $k$ anahtarına karşılık gelen değeri getirir.
- erase$(k)$: $k$ anahtarını ve değerini yapıdan kaldırır.
- contains$(k)$: $k$ anahtarının tabloda tanımlı olup olmadığını söyler.

### 25.3.3 İki Farklı Gerçekleme Felsefesi: Sıralı ve Sırasız

Set ve Map soyut veri tipleridir; ne yapmaları gerektiği belirlenmiştir ancak arka plandaki çalışma düzeni dilden dile değişir. Pratikte iki ana mekanizma kullanılır:

| **Özellik** | **Sıralı (Ağaç Tabanlı)** | **Sırasız (Özetleme / Hash Tabanlı)** |
|---|---|---|
| C++ Karşılığı | `std::set`, `std::map` | `std::unordered_set`, `std::unordered_map` |
| Python Karşılığı | — | `set`, `dict` |
| Arka Plan Yapısı | Dengeli İkili Arama Ağacı (Red-Black) | Karma Tablosu (Hash Table) |
| Ekleme Karmaşıklığı | $O(\log n)$ | Ortalama $O(1)$, En Kötü $O(n)$ |
| Arama Karmaşıklığı | $O(\log n)$ | Ortalama $O(1)$, En Kötü $O(n)$ |
| Silme Karmaşıklığı | $O(\log n)$ | Ortalama $O(1)$, En Kötü $O(n)$ |
| Eleman Sıralaması | Küçükten büyüğe sıralıdır | Rastgele / Sırasızdır |

Tablo 25.1: Sıralı ve Sırasız Veri Yapılarının Karşılaştırması

**Teorem 25.14** (Frekans Sayımının Zaman Karmaşıklığı)**.** *$n$ elemanlı bir dizideki tüm elemanların frekanslarını bir Hash Map (Python `dict` veya C++ `unordered_map`) kullanarak saymak ortalama $O(n)$ zamanda, bir Dengeli Ağaç Map (C++ `map`) kullanarak saymak ise kesin olarak $O(n \log n)$ zamanda tamamlanır.*

*Kanıt.* Dizideki her eleman için haritaya bir ekleme veya güncelleme yapılır. Hash tablosunda her işlem ortalama $O(1)$ sürdüğünden $n$ eleman için toplam süre $n \times O(1) = O(n)$ olur. Dengeli ikili ağaçta ise $i$'inci eleman eklenirken ağaçta en fazla

$i$ eleman vardır ve arama/ekleme maliyeti $O(\log i) \leq O(\log n)$ adımdır. $n$ işlem için toplam maliyet:

$$\sum_{i=1}^{n} O(\log i) = O(\log(n!)) = O(n \log n)$$

olarak bulunur. □

### 25.3.4 Kullanım Senaryoları ve Çözümlü Örnekler

**Örnek 25.15** (İki Sayının Toplamı - Two Sum)**.** *Sıralı olmayan bir $A$ tamsayı dizisi ve bir $T$ hedef değeri veriliyor. Dizi içerisinde toplamları $T$ olan iki farklı elemanın bulunup bulunmadığını $O(n)$ zamanda belirleyiniz.*

*Çözüm ve Akıl Yürütme.* Kaba kuvvet yaklaşımı iki döngüyle tüm $(A[i], A[j])$ ikililerini kontrol eder ve $O(n^2)$ zaman alır.

Oysa $A[i]$ sayısını incelerken kendimize şu soruyu sorabiliriz: "Toplamın $T$ olması için diğer sayının ne olması gerekir?" Aradığımız tamamlayıcı değer $T - A[i]$'dir. Böylece soru şuna dönüşür: "Şu ana kadar gördüğüm sayılar arasında $T - A[i]$ var mı?" Bir elemanın daha önce görülüp görülmediğini en hızlı sorgulayan yapı bir kümedir (**Set**).

Algoritma adımları:

1. Boş bir *gorulenler* kümesi oluşturulur.
2. Dizideki her $x$ elemanı için:
   - Eğer $(T - x) \in$ *gorulenler* ise, aranan çift bulunmuştur: $(x, T - x)$. Arama tamamlanır.
   - Değilse, $x$ sayısı *gorulenler* kümesine eklenir.
3. Dizi bittiğinde çift bulunamadıysa sonuç olumsuzdur.

Ortalama küme arama ve ekleme süresi $O(1)$ olduğundan, tüm dizi $O(n)$ sürede taranmış olur. □

```
FUNCTION two_sum(A, T)
    seen = EMPTY_SET
    FOR EACH x IN A DO
        complement = T - x
        IF complement IN seen THEN
            RETURN True, (complement, x)
        END IF
        INSERT x INTO seen
    END FOR
    RETURN False, NULL
END FUNCTION
```

**Örnek 25.16** (Farklı Eleman Sayısı - Distinct Elements)**.** *Boyutu $n = 10^5$ olan bir tamsayı dizisi veriliyor. Dizide kaç farklı (tekrarsız) eleman olduğunu bulunuz.*

*Çözüm ve Akıl Yürütme.* Bir küme (Set) matematiksel tanımı gereği aynı elemandan yalnızca bir tane tutabilir. Dolayısıyla dizinin tüm elemanlarını bir kümenin içine aktarırsak kopyalar kendiliğinden elenir. Sonuçta kümenin eleman sayısı ($|S|$), dizideki tekil eleman sayısını verir.

Zaman karmaşıklığı: Hash kümesi kullanılırsa $n$ ekleme işlemi ortalama $O(n)$ sürer; sıralı küme tercih edilirse $O(n \log n)$ zamanda biter. □

**Örnek 25.17** (Kayan Pencerede Farklı Eleman Sayısı)**.** *$n$ elemanlı bir $A$ dizisi ve bir $k$ pencere genişliği veriliyor. Her $k$ uzunluğundaki ardışık alt dizide (pencerede) kaç farklı eleman olduğunu hesaplayınız.*

*Çözüm ve Akıl Yürütme.* Her pencere için sıfırdan küme oluşturmak $(n-k+1) \times k$ işlem gerektirir ve $O(n \cdot k)$ zaman alır; bu da büyük girdiler için yavaştır.

Bölüm 22'de gördüğümüz kayan pencere tekniğini hatırlayalım: Pencere sağa bir birim kaydığında durum şöyledir:

- Pencerenin en solundaki eleman pencereden **çıkar**.
- Pencerenin sağına yeni bir eleman **girer**.
- Ortadaki $k - 2$ eleman aynen kalır.

Sadece bir küme (Set) tutarsak, bir elemanı çıkardığımızda o elemandan pencerede başka kopyalar olup olmadığını anlayamayız. Bu yüzden her elemanın pencere içindeki sayısını (frekansını) takip etmek gerekir. Dolayısıyla uygun yapı bir **Map**'tir:

1. İlk $k$ elemanı bir haritaya ($frekans$) ekleriz. Haritanın boyutu ($|frekans|$), ilk penceredeki farklı eleman sayısını verir.
2. $i = k$'den $n - 1$'e kadar:
   - Pencereden çıkan eleman: $cikan = A[i - k]$. $frekans[cikan]$ değerini 1 azaltırız. Frekansı 0'a düşerse bu anahtarı haritadan tamamen sileriz (erase).
   - Pencereye giren eleman: $giren = A[i]$. $frekans[giren]$ değerini 1 artırırız.
   - Haritada kalan anahtar sayısı o anki pencerenin farklı eleman sayısıdır.

Her kaydırmada yalnızca 1 silme ve 1 ekleme yapıldığından, algoritma toplam $O(n)$ zamanda tamamlanır. □

```
FUNCTION slidingWindowDistinct(A, k)
    frequency = empty map with default value 0
    results = empty list

    FOR i FROM 0 TO k - 1 DO
        frequency[A[i]] = frequency[A[i]] + 1
    END FOR
    APPEND SIZE(frequency) TO results

    FOR i FROM k TO LENGTH(A) - 1 DO
        outgoing = A[i - k]
        frequency[outgoing] = frequency[outgoing] - 1
        IF frequency[outgoing] == 0 THEN
            REMOVE outgoing FROM frequency
        END IF

        incoming = A[i]
        frequency[incoming] = frequency[incoming] + 1

        APPEND SIZE(frequency) TO results
    END FOR

    RETURN results
END FUNCTION
```

**En Sık Yapılan Hatalar (Set ve Map):**

- **C++ `map` ile `unordered_map` Farkını Unutmak:** C++ dilindeki varsayılan `std::map` yapısının $O(1)$ olduğu yanılgısına sık rastlanır. `std::map` kırmızı-siyah ağaç tabanlıdır ve her işlem $O(\log n)$ sürer. Öte yandan `unordered_map` kötü niyetli test durumlarında (hash çakışması saldırıları) $O(n)$ düzeyine gerileyebilir.

- **Olmayan Anahtara Erişerek Boyutu Büyütmek:** C++ dilinde bir `map`'te bulunmayan bir anahtara `m[x]` şeklinde erişildiğinde, dil o anahtarı varsayılan değerle (0) haritaya otomatik olarak ekler. Bu durum arama sırasında haritanın boyutunu gereksizce artırır ve mantık hatalarına yol açar. Bir anahtarın varlığı denetlenirken `count(x)` veya `find(x)` kullanılmalıdır.

- **Multiset Yerine Set Kullanmak:** Soruda bir elemanın birden fazla kopyasının saklanması gerektiğinde standart `set` kullanılırsa kopyalar kaybolur. Tekrar eden elemanlar için *multiset* veya bir frekans eşlemi (*map*) tercih edilmelidir.

## Bölüm Özeti ve Doğru Veri Yapısını Seçme Rehberi

Bir yarışma probleminde hangi veri yapısını seçeceğinize karar verirken şu sorulardan yararlanabilirsiniz:

1. **"En son eklenen elemanla mı ilgileniyorum?"** → Cevap evet ise: **Stack (Yığın)**. (Parantez denetimi, monoton sıralamalar, geri alma işlemleri).
2. **"Geliş sırasına göre mi işlemeliyim / Katman katman mı yayılmalıyım?"** → Cevap evet ise: **Queue (Kuyruk)**. (BFS arama, simülasyonlar, adil sıra yönetimi).
3. **"Bu eleman daha önce geldi mi / Var mı?"** → Cevap evet ise: **Set (Küme)**. (Tekillik kontrolü, ziyaret edilen düğümler).
4. **"Bu elemandan kaç tane var / Bu nesneye ait ek bilgi nedir?"** → Cevap evet ise: **Map (Eşlem / Sözlük)**. (Frekans sayımı, anahtar-değer ilişkileri).

Her veri yapısının avantaj sağladığı ve yetersiz kaldığı durumlar vardır. Burada ele aldığımız temel yapılar; ilerleyen bölümlerde göreceğimiz graf algoritmalarının, dinamik programlama optimizasyonlarının ve gelişmiş veri yapılarının zeminini oluşturacaktır.

# Bölüm 26

# Rekürsiyon ve DP'ye Giriş

Büyük ve karmaşık görünen birçok problem, aslında kendi kopyalarının bir araya gelmesiyle oluşur. Bir kütüphanedeki binlerce kitabı alfabetik sıraya dizmek, iki yarıyı ayrı ayrı dizip birleştirmekten ibarettir; elli basamaklı bir merdivenin tepesine ulaşmak ise kırk dokuzuncu veya kırk sekizinci basamağa geldikten sonra atılacak son bir adıma bağlıdır.

Bu bölümde algoritma tasarımının temel taşlarından üç yaklaşımı inceleyeceğiz: Problemi aynı türden daha küçük alt problemlere indirgeyen **rekürsiyon (rekürsiyon)**, yinelenen hesaplamaları belleğe kaydederek zaman kazandıran **memoization (bellekleme)** ve çözümü en küçük yapı taşlarından yukarıya doğru adım adım kuran **dinamik programlama (DP)**.

## 26.1 Rekürsiyon

Gündelik hayatta bir işi tanımlarken farkında olmadan rekürsiyonu kullanırız. Örneğin bir kuyrukta önünüzde kaç kişi olduğunu öğrenmek istediğinizi düşünün. Kuyruğun en başına kadar yürüyüp herkesi tek tek saymak zorunda değilsiniz. Önünüzdeki kişiye dönüp "Önünde kaç kişi var?" diye sorabilirsiniz. O kişi de kendi önündekine sorar. Bu soru zinciri en öndeki kişiye kadar ilerler. En öndeki kişi "Önümde kimse yok, yani 0 kişi var" dediğinde geriye doğru cevaplar birer artarak döner ve nihayet size ulaşır. Rekürsiyonun özü tam olarak budur: Bir problemi, aynı problemin daha küçük boyutlu bir girdisine devretmek.

### 26.1.1 Kendi Kendini Çağırma ve Baz Durum

Bir fonksiyonun kendi gövdesi içinde kendisini doğrudan ya da dolaylı olarak çağırmasına **rekürsif fonksiyon** denir. Ancak bir fonksiyon sürekli kendisini çağırırsa bu süreç sonsuza kadar devam eder. Bu nedenle her rekürsif yapının duracağı bir sınıra ihtiyacı vardır.

**Tanım 26.1** (Baz Durum ve Rekürsif Adım)**.** *Bir rekürsif fonksiyon iki ana parçadan oluşur:*

1. ***Baz Durum (Base Case):*** *Problemin artık daha küçük parçalara bölünemeyecek kadar basit olduğu ve cevabın doğrudan bilindiği durdurma koşuludur.*
2. ***Rekürsif Adım (Recursive Step):*** *Asıl problemin, aynı problemin daha küçük boyutlu bir veya daha fazla örneğine dönüştürülüp fonksiyonun yeniden çağrıldığı kısımdır.*

Bölüm 4'te permütasyonları sayarken kullandığımız faktöriyel tanımını hatırlayalım. $n$ pozitif bir tam sayı olmak üzere $n!$ değerini hesaplamak isteyelim:

$$5! = 5 \times 4 \times 3 \times 2 \times 1$$

Burada $4 \times 3 \times 2 \times 1$ çarpımı $4!$ değerine eşittir. Dolayısıyla:

$$5! = 5 \times 4!$$

Genel olarak $n \geq 1$ için:

$$n! = n \times (n-1)!$$

yazabiliriz. Bu küçülme $n = 0$ olana kadar sürer; $0! = 1$ kabulü bizim durdurma noktamız, yani baz durumumuzdur.

```
long long faktoriyel(int n) {
    if (n == 0) {
        return 1; // Baz durum
    }
    return n * faktoriyel(n - 1); // Rekursif adim
}
```

### 26.1.2 Çağrı Yığını (Call Stack) Mekanizması

Programlama dillerinde her fonksiyon çağrısı, belleğin **yığın (call stack)** adı verilen bölgesinde bir çerçeve (stack frame) oluşturur. Bir fonksiyon bir başkasını (veya kendisini) çağırdığında, mevcut fonksiyonun yürütülmesi duraklatılır; yerel değişkenleri ve kaldığı yer yığında saklanır. Çağrılan fonksiyon işini bitirip bir değer döndürdüğünde, duraklatılan fonksiyon yığından geri alınarak kaldığı yerden devam eder.

faktoriyel(3) çağrıldığında yığında gerçekleşen adımları inceleyelim:

1. faktoriyel(3) çağrılır. $3 \times$ faktoriyel(2) hesaplanmalıdır; beklemeye geçer.
2. faktoriyel(2) çağrılır. $2 \times$ faktoriyel(1) hesaplanmalıdır; beklemeye geçer.
3. faktoriyel(1) çağrılır. $1 \times$ faktoriyel(0) hesaplanmalıdır; beklemeye geçer.

4. faktoriyel(0) çağrılır. $n = 0$ baz durumdur, doğrudan 1 döndürür.
5. Şimdi yığın geriye doğru çözülür (unwinding):
   - faktoriyel(1) $= 1 \times 1 = 1$ döner.
   - faktoriyel(2) $= 2 \times 1 = 2$ döner.
   - faktoriyel(3) $= 3 \times 2 = 6$ döner.

**Teorem 26.2** (Matematiksel Tümevarım ile Rekürsiyonun Eşdeğerliği)**.** *Bir rekürsif algoritmanın doğruluğu, matematiksel tümevarım ilkesiyle doğrudan ispatlanır:*

1. ***Temel Adım (Baz Durum):*** *Algoritma en küçük girdi ($n = 0$ veya $n = 1$) için doğru sonucu verir.*
2. ***Tümevarım Adımı:*** *Algoritmanın $k < n$ değerleri için doğru çalıştığı varsayıldığında (tümevarım hipotezi), $n$ girdisi için yapılan birleşim işlemi de doğru sonucu üretir.*

*Her rekürsif çağrıda parametre kesinlikle baz duruma doğru yaklaşıyorsa, algoritma sonlu adımda sonlanır ve doğrudur.*

**Örnek 26.3.** *Verilen pozitif bir $a$ tam sayısı ve negatif olmayan bir $n$ tam sayısı için $a^n$ değerini $O(\log n)$ zamanda hesaplayan rekürsif bir algoritma tasarlayınız.*

*Çözüm.* İlk akla gelen yaklaşım $a^n = a \times a^{n-1}$ yazmaktır. Bu rekürsiyon $n$ adım sürer ve karmaşıklığı $O(n)$ olur. Ancak $n$'i her adımda bir azaltmak yerine yarıya indirebiliriz.

$n$ çift bir sayı ise, örneğin $a^8$, bunu $(a^4)^2$ olarak düşünebiliriz. Genel olarak:

$$a^n = (a^{n/2})^2 = a^{n/2} \times a^{n/2} \quad (n \text{ çift ise})$$

$n$ tek bir sayı ise, örneğin $a^9 = a \times a^8$:

$$a^n = a \times a^{n-1} \quad (n \text{ tek ise})$$

Baz durum ise $n = 0$ için $a^0 = 1$'dir.

Bu stratejide $n$ her iki adımda en az yarıya düşer; dolayısıyla çağrı derinliği en fazla $2\lceil \log_2 n \rceil$ olur.

```
long long us_al(long long a, int n) {
    if (n == 0) return 1; // Baz durum
    if (n % 2 == 0) {
        long long yari = us_al(a, n / 2);
        return yari * yari;
    } else {
        return a * us_al(a, n - 1);
    }
}
```

Burada kritik nokta, `us_al(a, n / 2)` değerini bir değişkende tutup karesini almaktır. Eğer doğrudan `us_al(a, n/2) * us_al(a, n/2)` yazılsaydı aynı fonksiyon iki kez çağrılır ve algoritma tekrar $O(n)$ karmaşıklığa gerilerdi. □

**Örnek 26.4** (Hanoi Kuleleri (Édouard Lucas, 1883))**.** *Üç direk $(A, B, C)$ ve başlangıçta $A$ direğinde büyük olanlar altta, küçük olanlar üstte olacak şekilde dizilmiş farklı boyutlarda $n$ adet disk bulunmaktadır.*

1. *Her hamlede sadece tek bir disk taşınabilir.*
2. *Hiçbir disk, kendisinden daha küçük bir diskin üzerine konulamaz.*

*Tüm diskleri $A$ direğinden $C$ direğine taşımak için gereken hamle sayısını ve adımları bulunuz.*

*Çözüm.* Bu problemi Bölüm 9'da yineleme bağıntıları bağlamında modellemiş ve $H_n = 2H_{n-1}+1$ bağıntısını elde etmiştik; şimdi aynı bağıntının rekürsif fonksiyon karşılığını kuralım. Problemi doğrudan tüm diskler üzerinden kavramak güçtür; bu nedenle en alttaki en büyük diske ($n$. diske) odaklanalım. Bu diski $A$'dan $C$'ye taşıyabilmemiz için:

1. Üzerindeki $n-1$ diskin $A$'dan kalkmış ve yardımcı direk olan $B$'de toplanmış olması gerekir.
2. Bu yapıldıktan sonra en büyük diski doğrudan $A$'dan $C$'ye taşırız (1 hamle).
3. Ardından $B$'deki $n-1$ diski, bu kez $A$'yı yardımcı direk olarak kullanarak $C$'ye taşırız.

Böylece $n$ diskli problem, iki adet $(n-1)$ diskli alt probleme indirgenmiş olur.

Baz durum: $n = 1$ olduğunda, tek disk doğrudan hedefe taşınır (1 hamle).

Hamle sayısını $H(n)$ ile gösterelim:

$$H(n) = 2H(n-1) + 1, \quad H(1) = 1$$

Bu rekürsif bağıntıyı çözelim:

$$\begin{aligned} H(1) &= 1 = 2^1 - 1 \\ H(2) &= 2(1) + 1 = 3 = 2^2 - 1 \\ H(3) &= 2(3) + 1 = 7 = 2^3 - 1 \\ &\vdots \\ H(n) &= 2^n - 1 \end{aligned}$$

Sözde kod ile çözümü ifade edersek:

```
ALGORITHM Hanoi(n, source, target, auxiliary)
    IF n == 1 THEN
        OUTPUT "Disk 1: " + source + " -> " + target
        RETURN
    END IF

    Hanoi(n - 1, source, auxiliary, target)
    OUTPUT "Disk " + n + ": " + source + " -> " + target
    Hanoi(n - 1, auxiliary, target, source)
END ALGORITHM
```

$2^n - 1$ hamle gereklidir ve her hamle kaçınılmazdır; dolayısıyla zaman karmaşıklığı $O(2^n)$'dir. □

**Örnek 26.5.** *Elemanları $\{1, 2, \ldots, n\}$ olan bir kümenin tüm alt kümelerini rekürsif olarak ekrana yazdıran bir algoritma tasarlayınız.*

*Çözüm.* Bölüm 1 ve 5'te gördüğümüz gibi, bir kümenin alt kümesini oluştururken her eleman için önümüzde bağımsız iki seçenek bulunur: O eleman alt kümede *ya vardır* ya da *yoktur*.

$k$. elemana geldiğimizde iki dala ayrılırız:

1. $k$ elemanını mevcut kümeye ekle ve $(k+1)$. eleman için rekürsiyonu çağır.
2. $k$ elemanını kümeye ekleme ve $(k+1)$. eleman için rekürsiyonu çağır.

Baz durum: $k = n + 1$ olduğunda tüm elemanlar hakkında karar verilmiştir; o anki küme yazdırılır.

```c
#include <stdio.h>

int kume[100];

void alt_kumeler(int n, int k, int secilen_sayisi) {
    if (k > n) {
        // Baz durum: Tum elemanlar incelendi
        printf("{ ");
        for (int i = 0; i < secilen_sayisi; i++) {
            printf("%d ", kume[i]);
        }
        printf("}\n");
        return;
    }
    // 1. Secenek: k elemanini sec
    kume[secilen_sayisi] = k;
    alt_kumeler(n, k + 1, secilen_sayisi + 1);

    // 2. Secenek: k elemanini secme
```

```
    alt_kumeler(n, k + 1, secilen_sayisi);
}
```

Toplam $2^n$ adet alt küme olduğundan fonksiyon tam $2^{n+1}-1$ kez çağrılır ve toplam çalışma zamanı $O(n2^n)$ olur. □

### En Sık Yapılan Hatalar

1. **Baz Durumu Unutmak veya Yanlış Tanımlamak:** Rekürsif fonksiyonlarda baz durum yazılmazsa veya parametre baz duruma hiçbir zaman ulaşamayacak şekilde güncellenirse (örneğin $n-1$ yerine yanlışlıkla $n+1$ yazılırsa), fonksiyon belleğin çağrı yığınını tüketene kadar kendini çağırır ve program *Segmentation Fault (Stack Overflow)* hatası vererek çöker.

2. **Ağır Nesneleri Değer Olarak (Pass-by-Value) Kopyalamak:** C++ veya Python gibi dillerde her çağrıda büyük bir diziyi kopyalayarak parametre geçirmek, hem zaman karmaşıklığını artırır hem de belleği hızla tüketir. Dizi paylaşımlı (global veya işaretçi/referans ile) iletilmelidir.

## 26.2 Memoization

Rekürsiyon, bir problemi alt parçalara bölerek kurgulamada son derece doğal ve etkilidir. Ancak saf rekürsiyon bazen ciddi bir verimsizliğe yol açar: **Örtüşen Alt Problemler (Overlapping Subproblems)**. Başka bir deyişle algoritma, tamamen aynı parametrelerle çağrılan alt problemleri tekrar tekrar en baştan hesaplar.

### 26.2.1 Tekrar Eden Hesaplamaların Maliyeti

Bölüm 9'da yineleme bağıntılarını incelerken gördüğümüz Fibonacci dizisini ele alalım. Tanım gereği:

$$F_0 = 0, \quad F_1 = 1$$

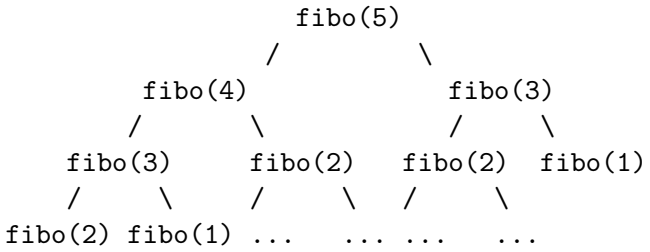
ve her $n \geq 2$ için:

$$F_n = F_{n-1} + F_{n-2}$$

Bu tanımı doğrudan koda dökelim:

```
long long fibo(int n) {
    if (n <= 0) return 0;
    if (n == 1) return 1;
    return fibo(n - 1) + fibo(n - 2);
}
```

$F_5$'i hesaplamak istediğimizde fonksiyonun oluşturduğu çağrı ağacını adım adım inceleyelim:




Ağaca bakıldığında gereksiz tekrarlar hemen göze çarpar:

- fibo(3) değeri **2 kez**,
- fibo(2) değeri **3 kez**,
- fibo(1) değeri **5 kez** sıfırdan hesaplanmaktadır.

$n$ biraz büyüdüğünde, örneğin $n = 45$ olduğunda toplam fonksiyon çağrısı sayısı milyarları bulur ve program saniyelerce sürer. Saf rekürsif Fibonacci fonksiyonunun çalışma zamanı kabaca altın oran tabanlıdır: $O\left(\left(\frac{1+\sqrt{5}}{2}\right)^n\right) \approx O(1.618^n)$, yani üsteldir.

### 26.2.2 Memoization İlkesi

**Tanım 26.6** (Memoization)**.** *Bir fonksiyonun daha önce hesapladığı girdilere ait sonuçları bir tabloda (dizi, hash tablosu vb.) saklaması ve aynı girdiyle tekrar çağrıldığında hesaplama yapmak yerine saklanan sonucu doğrudan döndürmesi tekniğine* ***memoization (bellekleme)*** *denir.*

Memoization, "yukarıdan aşağıya" (top-down) bir yaklaşımdır. Problemi yine en büyük girdiden başlayarak kurarız; ancak çözülen her alt problemin sonucunu hafızaya not ederiz.

**Teorem 26.7** (Memoization Karmaşıklık Analizi)**.** *Bir rekürsif fonksiyonda birbirinden farklı $S$ adet durum (farklı parametre kombinasyonu) varsa ve her durum için memoization tablosu kontrol edilip geçiş maliyeti $O(1)$ sürede hesaplanıyorsa; tüm algoritmanın zaman karmaşıklığı:*

$$T = O(S \times \textit{geçiş başına maliyet})$$

*ve bellek karmaşıklığı saklanan durum sayısı kadar, yani $O(S)$ olur.*

*İspat Taslağı.* Her farklı durum için fonksiyon gövdesi yalnızca *bir kez* baştan sona çalışır ve bulunan sonuç tabloya yazılır. Sonraki tüm çağrılarda tablo değeri $O(1)$ sürede doğrudan döndürülür. Dolayısıyla her durum en fazla bir kez çözülür. $S$ durumun her biri $O(1)$ zamanda işlendiğinde toplam süre $O(S)$ olur. □

Fibonacci problemine memoization uygulayalım:

```c
#include <stdio.h>

long long memo[100];

long long fibo_memo(int n) {
    if (n <= 0) return 0;
    if (n == 1) return 1;

    // Tabloda daha once hesaplanmis mi kontrol et
    if (memo[n] != -1) {
        return memo[n];
    }

    // Hesaplanmamis, hesapla ve kaydet
    memo[n] = fibo_memo(n - 1) + fibo_memo(n - 2);
    return memo[n];
}

int main() {
    for (int i = 0; i < 100; i++) memo[i] = -1; // -1 ile baslat
    printf("F(50) = %lld\n", fibo_memo(50));
    return 0;
}
```

Burada $S = n + 1$ farklı durum vardır. Her durum yalnızca bir kez hesaplandığından karmaşıklık $O(2^n)$ yerine doğrudan $O(n)$'e düşer. Böylece $F_{50}$ milisaniyeler içinde hesaplanabilir.

**Örnek 26.8** (Izgarada Yol Sayısı)**.** *$n \times m$ boyutlarında bir ızgaranın sol üst köşesinden $(0, 0)$ sağ alt köşesine $(n-1, m-1)$ gitmek istiyorsunuz. Her adımda yalnızca bir birim **sağa** veya bir birim **aşağı** hareket edebilirsiniz. $(0, 0)$ noktasından $(n-1, m-1)$ noktasına kaç farklı yoldan gidilebilir?*

*Çözüm.* Hedefe tersten bakalım: $(r, c)$ karesine nereden gelebiliriz? Sadece sağa ve aşağı hareket edilebildiğine göre, $(r, c)$ hücresine ya solundaki $(r, c-1)$ hücresinden sağa giderek ya da üstündeki $(r-1, c)$ hücresinden aşağı inerek ulaşılmış olabilir.

O halde $(r, c)$ hücresine ulaşma yollarının sayısı:

$$Yol(r, c) = Yol(r-1, c) + Yol(r, c-1)$$

**Baz durumlar:**

- Başlangıç noktası: $Yol(0, 0) = 1$.
- Izgara sınırları dışı: $r < 0$ veya $c < 0$ ise $Yol(r, c) = 0$.

Burada durum parametreleri $(r, c)$ ikilisidir. Toplam farklı durum sayısı $n \times m$'dir. Memoization olmadan dallanma sayısı $2^{n+m}$ mertebesinde olurken, $n \times m$'lik bir tablo ile karmaşıklık $O(n \times m)$'e iner.

Sözde kod ile memoization uygulaması:

```
ALGORITHM countPaths(row, col, memo)
    IF row == 0 AND col == 0 THEN
        RETURN 1
    END IF

    IF row < 0 OR col < 0 THEN
        RETURN 0
    END IF

    state = (row, col)
    IF state IS IN memo THEN
        RETURN memo[state]
    END IF

    memo[state] = countPaths(row - 1, col, memo) + countPaths(row, col -
    1, memo)
    RETURN memo[state]
END ALGORITHM

INITIALIZE memo AS EMPTY MAP
OUTPUT countPaths(14, 14, memo)
```

Bölüm 9'da kafes yollarını sayarken gördüğümüz gibi, bu problem kombinatorik olarak $\binom{(n-1)+(m-1)}{n-1}$ formülüyle de çözülebilir. DP ve memoization, bu kombinatorik toplamı formül ezberine gerek kalmadan adım adım keşfetmemizi sağlar. □

**Örnek 26.9** (Madeni Para Bozdurma)**.** *Elinizde $\{c_1, c_2, \ldots, c_k\}$ değerlerinde sınırsız sayıda madeni para bulunmaktadır. Toplam $V$ tutarını oluşturmak için kullanılması gereken **en az** madeni para sayısını bulunuz. Eğer bu tutar oluşturulamıyorsa $-1$ döndürünüz.*

*Çözüm.* Hedefimiz $V$ tutarını oluşturmak olduğuna göre ilk adımı düşünelim: İlk madeni para olarak listedeki herhangi bir $c_i$ değerini seçebiliriz.

Eğer $c_i$ değerli parayı seçersek, geriye $V - c_i$ tutarı kalır ve bu tutar için gereken en az para sayısını bulup 1 ekleriz. Tüm geçerli paralar arasından en küçük sonucu vereni seçmeliyiz:

$$f(V) = 1 + \min_{\{i \mid c_i \leq V\}} f(V - c_i)$$

**Baz durum:**

- $f(0) = 0$ (0 tutarı için 0 para gerekir).
- $V < 0$ ise imkânsızdır, sonsuz ($\infty$) döndürülür.

Farklı durum sayısı $V + 1$'dir. Memoization tablosu sayesinde her tutar sadece bir kez hesaplanır.

```
#include <stdio.h>
#include <limits.h>
#define INF 1000000000

int paralar[] = {1, 3, 4};
int k = 3;
int memo[10005];

int min_para(int v) {
    if (v == 0) return 0;
    if (v < 0) return INF;
    if (memo[v] != -1) return memo[v];

    int en_iyi = INF;
    for (int i = 0; i < k; i++) {
        if (v >= paralar[i]) {
            int sonuc = min_para(v - paralar[i]);
            if (sonuc != INF && sonuc + 1 < en_iyi) {
                en_iyi = sonuc + 1;
            }
        }
    }
    memo[v] = en_iyi;
    return memo[v];
}
```

Her durum için $k$ farklı para denendiğinden zaman karmaşıklığı $O(V \times k)$, bellek karmaşıklığı ise $O(V)$ olur. □

## En Sık Yapılan Hatalar

1. **Memo Tablosunu Yanlış İlklendirmek:** Bir durumun cevabı meşru olarak 0 olabiliyorsa (örneğin maliyetin 0 çıkabildiği durumlar), tabloyu 0 ile ilklendirmek fonksiyonun o durumu henüz hesaplanmamış sayıp tekrar tekrar çalıştırmasına yol açar. Tablo daima ulaşılamayacak bir değerle (genellikle $-1$ veya $\infty$) doldurulmalıdır.

2. **Çağrı Derinliği Aşımı:** Memoization yukarıdan aşağıya çalıştığı için çağrı derinliği çok büyüdüğünde (örneğin $N = 10^6$ için tek tek azalan durumlarda) çağrı yığını dolar ve program çöker. Bu tip durumlarda döngü tabanlı dinamik programlama tercih edilmelidir.

## 26.3 DP'ye Giriş (Dinamik Programlama)

Memoization yönteminde problemin zirvesinden başlayıp aşağıya doğru indik. Şimdi madalyonun diğer yüzünü çevirelim: Problemi en küçük parçalarından başlayarak bir duvar örer gibi yukarıya doğru adım adım inşa edebiliriz.

Bu yaklaşıma **Dinamik Programlama (Tabulation / Aşağıdan Yukarıya DP)** denir.

### 26.3.1 Dinamik Programlamanın İki Temel Özelliği

Bir problemin dinamik programlama ile çözülebilmesi için iki temel özelliğe sahip olması gerekir:

**Tanım 26.10** (Optimal Alt Yapı ve Örtüşen Alt Problemler)**.** *1. **Optimal Alt Yapı (Optimal Substructure):** Büyük bir problemin optimal (en iyi) çözümünün, alt problemlerinin optimal çözümlerinden oluşturulabilmesidir.*

*2. **Örtüşen Alt Problemler (Overlapping Subproblems):** Aynı alt problemlerin çözüm sürecinde defalarca karşımıza çıkmasıdır.*

Eğer alt problemler örtüşmüyorsa (yani her alt problem tamamen bağımsız ve yeniyse), dinamik programlama yerine **Böl ve Yönet (Divide and Conquer)** yaklaşımı (örneğin Hızlı Sıralama veya Birleştirmeli Sıralama) kullanılır.

### 26.3.2 Yukarıdan Aşağı (Memoization) ile Aşağıdan Yukarı (Tabulation) Karşılaştırması

| **Kriter** | **Yukarıdan Aşağı (Memoization)** | **Aşağıdan Yukarı (Tabulation)** |
|---|---|---|
| **Yapı** | Rekürsiyon + Önbellek | Döngüler (`for` / `while`) + Tablo |
| **Yürütme Sırası** | İhtiyaç duyuldukça alt problemlere dallanır. | En küçük alt problemden başlanarak sırayla tablo doldurulur. |
| **Avantajı** | Sadece gereken alt problemler çözülür; düşünmesi doğaldır. | Çağrı yığını maliyeti yoktur; hızlıdır; bellek optimizasyonuna açıktır. |
| **Dezavantajı** | Çağrı yığını taşması riski taşır; sabit çarpanı daha büyüktür. | İhtiyaç duyulmasa bile tablodaki tüm durumlar hesaplanabilir. |

### 26.3.3 DP'nin Çekirdeği: Durum ve Geçiş

Dinamik programlama ile problem çözerken iki temel soruya cevap aranır:

1. **Durum (State) nedir?** Alt problemi eksiksiz ve benzersiz şekilde tarif eden değişkenler kümesi nedir? ($dp[i]$, $dp[i][j]$ vb.)

2. **Geçiş Bağıntısı (Transition) nedir?** Bir durumu, daha önce hesaplanmış daha küçük durumlardan nasıl elde ederiz?

**Örnek 26.11** (Merdiven Tırmanma Problemi)**.** *$n$ basamaklı bir merdivenin tepesine çıkmak istiyorsunuz. Her adımda **1 basamak** veya **2 basamak** atlayabilirsiniz. Zirveye kaç farklı şekilde ulaşabilirsiniz?*

*Çözüm.* Bölüm 9'da yineleme bağıntıları çerçevesinde ele aldığımız bu problemi, şimdi dinamik programlamanın durum ve geçiş diliyle kuralım. Son adımı düşünelim. $n$. basamağa basmadan hemen önce nerede olabilirdik?

- $(n-1)$. basamakta olup 1 basamak atlamış olabiliriz.
- $(n-2)$. basamakta olup 2 basamak atlamış olabiliriz.

Başka bir ihtimal yoktur. $(n-1)$. ve $(n-2)$. basamağa kaç farklı yolla çıkabildiğimizi bilirsek, bu iki sayıyı toplayarak $n$. basamağa ulaşma sayısını buluruz.

**Durum Tanımı:** $dp[i]$: $i$. basamağa ulaşmanın farklı yollarının sayısı.

**Geçiş Bağıntısı:**

$$dp[i] = dp[i-1] + dp[i-2] \quad (i \geq 2)$$

**Baz Durumlar:** 0. basamakta durmak 1 yoldur: $dp[0] = 1$. 1. basamağa sadece 1 adımla gidilebilir: $dp[1] = 1$.

Bu bildiğimiz Fibonacci bağıntısıdır. Şimdi bunu döngü ile aşağıdan yukarıya çözelim:

```
long long merdiven(int n) {
    if (n <= 1) return 1;
    long long dp[n + 1];
    dp[0] = 1;
    dp[1] = 1;
    for (int i = 2; i <= n; i++) {
        dp[i] = dp[i - 1] + dp[i - 2];
    }
    return dp[n];
}
```

Zaman karmaşıklığı $O(n)$, bellek karmaşıklığı $O(n)$'dir.

**Bellek Optimizasyonu:** $dp[i]$'yi hesaplamak için tablonun tamamına ihtiyacımız yoktur; yalnızca bir önceki ($dp[i-1]$) ve iki önceki ($dp[i-2]$) değer yeterlidir. Dolayısıyla bütün bir diziyi tutmak yerine yalnızca iki değişken saklayarak bellek karmaşıklığını $O(1)$'e düşürebiliriz:

```
long long merdiven_optimizasyon(int n) {
    if (n <= 1) return 1;
    long long onceki2 = 1; // dp[0]
    long long onceki1 = 1; // dp[1]
    long long suanki = 0;
    for (int i = 2; i <= n; i++) {
        suanki = onceki1 + onceki2;
        onceki2 = onceki1;
        onceki1 = suanki;
    }
    return suanki;
}
```

Bu sadeleştirme, yarışmalarda bellek limitini aşmamak için önemli bir tekniktir. □

**Örnek 26.12** (Minimum Maliyetli Merdiven Tırmanışı)**.** *$n$ basamaklı bir merdivende her $i$. basamağa basmanın bir maliyet$[i]$ bedeli vardır. 0. veya 1. basamaktan tırmanmaya başlayabilirsiniz. Her adımda 1 veya 2 basamak yukarı çıkabilirsiniz. Merdivenin tepesine (yani $n$. indekse) ulaşmak için katlanılması gereken* ***minimum*** *toplam maliyet nedir?*

*Çözüm.* Merdivenin $i$. basamağına varmak için atılan son adımı düşünelim. O basamağa ya $(i-1)$. basamaktan ya da $(i-2)$. basamaktan gelinmiştir. $i$. basamağa gelene kadar ödenen toplam minimum maliyet, bu iki basamağın hangisinden gelmek daha ucuzsa onun maliyetine $i$. basamağın maliyetinin eklenmesiyle bulunur.

**Durum:** $dp[i]$: $i$. basamağa basmanın minimum toplam maliyeti.

**Geçiş:**

$$dp[i] = maliyet[i] + \min(dp[i-1], dp[i-2])$$

**Baz Durum:** Başlangıç 0 veya 1 olabildiğinden: $dp[0] = maliyet[0]$ $dp[1] = maliyet[1]$

Merdivenin tepesi $n$. basamaktır (üzerinde basamak maliyeti yoktur). Oraya varmak $\min(dp[n-1], dp[n-2])$ maliyetindedir.

```
#define MIN(a,b) (((a)<(b))?(a):(b))

int min_maliyet(int maliyet[], int n) {
    if (n == 1) return maliyet[0];
    int dp[n];
    dp[0] = maliyet[0];
    dp[1] = maliyet[1];

    for (int i = 2; i < n; i++) {
        dp[i] = maliyet[i] + MIN(dp[i - 1], dp[i - 2]);
    }
    return MIN(dp[n - 1], dp[n - 2]);
```

```
}
```

Zaman karmaşıklığı $O(n)$, bellek karmaşıklığı $O(n)$'dir (benzer şekilde $O(1)$ belleğe optimize edilebilir). □

**Örnek 26.13** (Maksimum Alt Dizi Toplamı — Kadane Algoritması)**.** *$A = [a_0, a_1, \ldots, a_{n-1}]$ tam sayılardan oluşan bir dizi veriliyor (negatif sayılar içerebilir). Dizinin ardışık (bitişik) elemanlarından oluşan alt dizileri arasından toplamı **en büyük** olan alt dizinin toplamını bulunuz.*

*Çözüm.* Tüm $[i, j]$ aralıklarını tek tek denemek $O(n^2)$ süre alır. Bunun yerine diziyi soldan sağa tararken dinamik programlamadan yararlanabiliriz.

$i$. elemanı incelerken, bu elemanda biten en büyük toplamlı alt diziyi düşünelim. Önümüzde iki seçenek vardır:

1. $i$. elemanı, $(i-1)$. elemanda biten en iyi alt dizinin ucuna ekleriz.
2. Önceki alt diziyi bırakıp $a_i$ elemanından yeni bir alt dizi başlatırız (önceki toplam negatif ise bu hamle açıkça daha kazançlıdır).

**Durum:** $dp[i]$: $i$. elemanda $(a_i)$ **kesinlikle biten** en büyük alt dizi toplamı.

**Geçiş:**

$$dp[i] = \max(a_i, dp[i-1] + a_i)$$

Tüm dizideki genel en büyük toplam ise, herhangi bir indekste biten en büyüklerin maksimumudur:

$$\text{Cevap} = \max_{0 \le i < n} dp[i]$$

**Baz Durum:** $dp[0] = a_0$.

```
#define MAX(a,b) (((a)>(b))?(a):(b))

long long max_alt_dizi(int a[], int n) {
    long long suanki_max = a[0];
    long long genel_max = a[0];

    for (int i = 1; i < n; i++) {
        suanki_max = MAX((long long)a[i], suanki_max + a[i]);
        genel_max = MAX(genel_max, suanki_max);
    }
    return genel_max;
}
```

Bu etkili DP çözümü, literatürde **Kadane Algoritması** olarak geçer. Çalışma zamanı sadece $O(n)$ ve bellek ihtiyacı $O(1)$'dir. □

**Örnek 26.14** (Aşağıdan Yukarıya Madeni Para Bozdurma)**.** $\{1, 3, 4\}$ *değerindeki paralarla* $V$ *tutarını oluşturmak için gereken minimum para sayısını aşağıdan yukarıya (tabulation) yöntemiyle bulunuz.*

*Çözüm.* Hesaplamaya en küçük tutardan, yani 0'dan başlayarak $V$'ye kadar her bir tutar için gereken minimum para sayısını bir tabloda saklayalım.

**Durum:** $dp[i]$: $i$ tutarını elde etmek için gereken minimum para sayısı.

**Geçiş:** $i$ tutarına ulaşmak için son eklediğimiz para $c \in \{1, 3, 4\}$ olabilir:

$$dp[i] = 1 + \min_{\{c | c \le i\}} dp[i - c]$$

**Baz Durum:** $dp[0] = 0$. Diğer tüm değerleri başlangıçta $\infty$ yaparız.

Tabloyu elle $V = 6$ için adım adım dolduralım:

- $dp[0] = 0$
- $i = 1$: $dp[1] = 1 + dp[0] = 1$ (1'lik para kullanıldı)
- $i = 2$: $dp[2] = 1 + dp[1] = 2$ (1'lik)
- $i = 3$: Paralar 1 veya 3 olabilir: $\min(1 + dp[2], 1 + dp[0]) = \min(3, 1) = 1$ (3'lük)
- $i = 4$: Paralar 1, 3 veya 4: $\min(1+dp[3], 1+dp[1], 1+dp[0]) = \min(2, 2, 1) = 1$ (4'lük)
- $i = 5$: $\min(1 + dp[4], 1 + dp[2], 1 + dp[1]) = \min(2, 3, 2) = 2$ (4+1 veya 1+4)
- $i = 6$: $\min(1 + dp[5], 1 + dp[3], 1 + dp[2]) = \min(3, 2, 3) = 2$ (3+3)

Sonuç $dp[6] = 2$'dir ($3 + 3$ madeni paralarıyla).

```
int min_para_tabulation(int paralar[], int k, int V) {
    int dp[V + 1];
    dp[0] = 0;
    for (int i = 1; i <= V; i++) {
        dp[i] = 1e9; // Sonsuz ile baslat
        for (int j = 0; j < k; j++) {
            if (i >= paralar[j]) {
                if (dp[i - paralar[j]] + 1 < dp[i]) {
                    dp[i] = dp[i - paralar[j]] + 1;
                }
            }
        }
    }
    return (dp[V] >= 1e9) ? -1 : dp[V];
}
```

Tablo düzenli doldurulduğundan hiçbir rekürsif çağrı maliyeti oluşmaz; döngü $V \times k$ kez döner ve süre $O(V \cdot k)$ olur. □

### En Sık Yapılan Hatalar

1. **Döngü Sıralamasını Yanlış Kurmak:** DP tablosu doldurulurken temel kural şudur: *Bir durumu hesaplarken başvurduğunuz tüm alt durumlar daha önceden hesaplanmış olmalıdır.* Örneğin $dp[i]$ değeri $dp[i-1]$'e bağlıysa döngü 1'den $n$'e doğru gitmelidir; aksi yönde bir döngü henüz hesaplanmamış anlamsız değerleri kullanır.

2. **Durumu Eksik Tanımlamak:** Problemin sonucunu etkileyen bir parametreyi (örneğin son yapılan hamle, kullanılan para adedi vb.) durum tanımına katmamak geçiş bağıntısını kurmayı imkânsız hale getirir.

3. **Dizi Sınırlarını Aşmak (Out of Bounds):** $dp[i-2]$ veya $dp[i-c]$ yazarken $i-2<0$ durumlarını kontrol etmemek negatif indeks hatalarına yol açar.

## 26.4 Bölüm Özeti ve Problem Çözme Rehberi

TÜBİTAK 1. Aşama sınavında bir soruyla karşılaştığınızda izlemeniz gereken zihinsel yol haritası şudur:

1. **İndirgeme Düşüncesi:** Problemin en son adımını hayal edin. "Son hamle ne olabilirdi?" veya "Bu elemanı kümeye alsam geriye ne kalır, almasam ne kalır?" diye sorun.

2. **Saf Rekürsiyonu Kurun:** İlk olarak kafanızda veya karalama kâğıdında baz durumları ve rekürsif bağıntıyı netleştirin.

3. **Alt Problemler Çakışıyor mu?** Fonksiyon aynı parametrelerle tekrar çağrılıyor mu? Cevabınız evet ise bir memoization tablosu tasarlayın.

4. **Aşağıdan Yukarıya Çevirin (Tabulation):** Bellek sınırları darsa veya çağrı yığını riski bulunuyorsa bağıntıyı bir döngü yapısına dönüştürün.

5. **Bellek Tasarrufu Yapılabilir mi?** Gelecekteki durumlar için tablonun sadece son birkaç hücresi yetiyorsa diziyi silip durumu birkaç değişkende tutun.

Rekürsiyon ve dinamik programlama belirli bir algoritma değil, genel bir problem çözme biçimidir. İlerleyen bölümlerde bu temellerin üzerine 2-boyutlu DP, sırt çantası (knapsack) ve ağaçlar üzerinde DP gibi daha ileri teknikleri kuracağız.

# Bölüm 27

# Greedy Yaklaşımı

Günlük hayatta bir karar alırken çoğu zaman uzun vadeli, karmaşık olasılık ağaçlarını hesaplamak yerine o an için en avantajlı görünen seçeneğe yöneliriz. Markette en kısa kuyruğu seçmek, trafikte anlık olarak daha hızlı ilerleyen şeride geçmek veya bir açık büfede tabağa ilk olarak en lezzetli görünen yemekleri doldurmak bu yaklaşımın tipik örnekleridir. Bilgisayar biliminde ve matematikte bu felsefeye **greedy yaklaşım** (*greedy approach*) adı verilir.

Greedy yaklaşımı, çok adımlı bir optimizasyon probleminde her aşamada genel durumu hesaba katmaksızın, o anki koşullarda en karlı ya da "yerel olarak en iyi" (*locally optimal*) görünen adımı atma ilkesine dayanır. Ancak bu refleks her zaman doğru sonuca ulaştırmaz. Şerit değiştirdikten hemen sonra o şeridin tıkanması gibi, anlık yerel kararlar kimi zaman genel çözümün başarısız olmasına yol açabilir.

Bu bölümde, greedy yaklaşımının ne zaman kesin çözüme ulaştırdığını, ne zaman yanıltıcı olabileceğini inceleyecek; algoritmanın doğruluğunu kanıtlama yöntemlerini ve olimpiyatlarda sıkça kullanılan temel problem kalıplarını ele alacağız.

## 27.1 Greedy Kavramı

Greedy algoritmalarının temel vaadi basitlik ve hızdır. Bölüm 26'da ele aldığımız dinamik programlama veya tam arama (*brute force*) gibi yöntemler tüm durum uzayını adım adım değerlendirirken, greedy algoritması geriye dönüp bakmaz; verdiği kararları sonradan değiştirmez (*irrevocable decisions*).

### 27.1.1 Yerel En İyi Her Zaman Doğru mu?

Bir dağcının yoğun sis altında zirveye tırmanmak istediğini varsayalım. Dağcı etrafını göremediği için şöyle bir kural uygular: "Her adımda ayağımın altındaki eğimin en dik yukarı gittiği yöne doğru bir adım atacağım." Bu strateji, dağcının hızla yükselmesini sağlar. Ancak dağ tek bir zirveden ibaret değilse, dağcı kendini ana zirve yerine küçük bir tepenin (yerel maksimum) doruğunda bulabilir; oradan

ana zirveye ulaşmak için önce vadiye inmesi gerekir, fakat izlediği kural inişe izin vermez.

**Örnek 27.1.** *Elimizde* 1*,* 3 *ve* 4 *TL değerinde madeni paralar bulunmaktadır. Toplam* 6 *TL tutarındaki bir ödemeyi* ***en az sayıda*** *madeni para kullanarak yapmak istiyoruz.*

*Çözüm ve Analiz.* **Greedy Stratejisi:** Her adımda ödenmesi gereken tutarı aşmayan en büyük değerli parayı seçelim.

1. 6 TL için seçilebilecek en büyük para 4 TL'dir. Kalan borç: $6 - 4 = 2$ TL.
2. 2 TL için 4 veya 3 seçilemez, mecburen 1 TL seçilir. Kalan: $2 - 1 = 1$ TL.
3. 1 TL için yine 1 TL seçilir. Kalan: 0 TL.

Greedy algoritması $\{4, 1, 1\}$ olmak üzere **3 adet** para kullandı.

**Küresel En İyi (Optimal) Çözüm:** Bunun yerine iki adet 3 TL seçilirse, $3 + 3 = 6$ TL elde edilir ve yalnızca **2 adet** para yeterli olur!

Görüldüğü üzere greedy yaklaşım bu problemde başarısız olmuştur. İlk adımda en büyük parayı seçmek yerel olarak borcu en çok azaltan hamleydi; fakat sistemi öyle bir alt probleme (2 TL) kilitledi ki, o alt problemin çözümü genel verimliliği düşürdü. □

Bir başka klasik başarısızlık örneği ise 0-1 Sırt Çantası (*0-1 Knapsack*) problemidir.

**Örnek 27.2.** *Taşıma kapasitesi* $W = 50$ *kg olan bir çantamız ve aşağıdaki üç eşyamız olsun:*

- *Eşya 1:* 10 *kg, Değeri:* 60 *TL (Birim değer:* 6 *TL/kg)*
- *Eşya 2:* 20 *kg, Değeri:* 100 *TL (Birim değer:* 5 *TL/kg)*
- *Eşya 3:* 30 *kg, Değeri:* 120 *TL (Birim değer:* 4 *TL/kg)*

*Her eşyadan en fazla bir adet alabiliriz (bölünemez). Çantaya sığacak eşyaların toplam değerini maksimize etmek istiyoruz.*

*Çözüm ve Analiz.* Eğer "birim ağırlık başına değeri en yüksek olan eşyayı önce al" şeklinde sezgisel bir greedy kuralı uygularsak:

1. İlk olarak Eşya 1 seçilir (10 kg, 60 TL). Kalan kapasite: 40 kg.
2. Sonraki en değerli birim Eşya 2 seçilir (20 kg, 100 TL). Kalan kapasite: 20 kg.
3. Kalan 20 kg kapasiteye Eşya 3 (30 kg) sığmaz.

Toplam elde edilen değer: $60 + 100 = 160$ TL (toplam ağırlık 30 kg, 20 kg boş kaldı).

Oysa Eşya 2 ve Eşya 3'ü seçseydik: Ağırlık: $20 + 30 = 50$ kg (çanta tam doldu), toplam değer: $100 + 120 = 220$ TL olacaktı. Birim değeri en yüksek olan Eşya 1'i almak, geride kalan kapasiteyi verimsiz parçalara bölerek en iyi sonuca ulaşmamızı engelledi. □

### 27.1.2 Bir Greedy Algoritmasının Anatomisi

Bir problemin greedy yöntemle kesin ve optimal biçimde çözülebilmesi için iki temel matematiksel özelliğe sahip olması gerekir:

**Tanım 27.3** (Greedy Seçim Özelliği / Greedy Choice Property)**.** *Bir problemin küresel optimal çözümüne, yerel olarak en iyi görünen seçimler adım adım yapılarak ulaşılabilmesidir. Yani geçmişteki durumları yeniden değerlendirmeye veya gelecekteki adımların tüm kombinasyonlarını dallandırmaya gerek yoktur.*

**Tanım 27.4** (Optimal Alt Yapı / Optimal Substructure)**.** *Problemin optimal çözümü, kendi içinde barındırdığı alt problemlerin de optimal çözümlerini içerir. Yani ilk greedy seçimi yaptıktan sonra geriye kalan alt problem çözüldüğünde, bu iki çözümün birleşimi orijinal problemin kesin çözümünü verir.*

### 27.1.3 Greedy Doğruluğunu İspatlama Teknikleri

Olimpiyatlarda bir probleme greedy bir sezgiyle yaklaşmak işin sadece başlangıcıdır. Esas kritik aşama, algoritmanın doğruluğundan emin olmaktır. "Örnek girdilerde çalışıyor" düşüncesi en yaygın yanılgılardan biridir. Bir greedy algoritmanın kesinliğini kanıtlamak için genellikle iki temel yöntemden yararlanılır:

#### 1. Değiş-Tokuş Argümanı (*Exchange Argument*)

Bu yöntem, greedy ispatlarının en temel ve etkili aracıdır. Mantık şöyledir:

1. Algoritmamızın ürettiği çözüme $G$ diyelim.
2. Algoritmamızın doğru olmadığını varsayalım. O halde $G$'den farklı, gerçek bir optimal çözüm $O$ var olsun.
3. $O$ çözümü ile $G$ arasındaki ilk farkı (ayrışılan ilk kararı) buluruz.
4. $O$ çözümündeki bu farklı elemanı çıkarıp yerine $G$'deki ilgili elemanı koyarız (değiş-tokuş yaparız).
5. Gösteririz ki yeni elde edilen çözüm $O'$, $O$'dan daha kötü değildir (değer azalmıyor veya maliyet artmıyor) ve geçerliliğini korumaktadır.

6. Bu değiş-tokuş adımı sonlu kez tekrarlanarak $O$ çözümü adım adım $G$'ye dönüştürülebilir. Hiçbir adımda çözüm kötüleşmediğine göre $G$ de en az $O$ kadar iyidir, yani $G$ bir optimal çözümdür.

#### 2. Greedy Önde Gider (*Greedy Stays Ahead*)

Bu yöntemde, çözümün her adımında ölçülebilen bir parametre (örneğin tamamlanan iş sayısı, harcanan minimum süre vb.) belirlenir. Bölüm 2'de gördüğümüz tümevarım (*induction*) ilkesi kullanılarak, her $k$. adımda greedy algoritmanın ulaştığı durumun, varsayımsal herhangi bir optimal çözümün $k$. adımındaki durumundan "daha geride olmadığı" ispatlanır.

> **Uyarı:** Bir greedy kural tasarlayıp doğruluğunu ispatlamadan kodlamaya geçmek ciddi bir risk taşır. Karşılaşılan problemlerin birçoğunda ilk akla gelen greedy sezgisi yanıltıcıdır ve gizli bir karşı örnek (*counterexample*) barındırır. İspatı net biçimde kurulmamış bir greedy kuralına güvenilmemelidir.

## 27.2 Tipik Örnekler

Greedy yaklaşımının doğru sonuç verdiği temel problem sınıflarını ve bu problemlere ait ispatları adım adım inceleyelim.

### 27.2.1 Aralık Seçimi Problemi (*Interval Scheduling*)

Bu problem, zamanlama ve kaynak yönetimi algoritmalarının temel taşlarından biridir.

**Tanım 27.5** (Aralık Seçimi Problemi)**.** *Bize tek bir salon ve bu salonda yapılmak istenen $n$ adet etkinlik veriliyor. Her etkinliğin bir başlangıç zamanı $s_i$ ve bir bitiş zamanı $e_i$ vardır; yani etkinlik $[s_i, e_i)$ yarı açık aralığında salonu kullanacaktır. İki etkinlik zaman bakımından çakışmıyorsa (biri bittiğinde diğeri başlıyor veya daha sonra başlıyorsa) aynı salonda peş peşe gerçekleştirilebilir. Amaç, salonda gerçekleştirilebilecek **maksimum sayıda** etkinliği seçmektir.*

Doğru bir greedy kuralı oluşturmak için hangi ölçüte göre sıralama yapılması gerektiğine bakalım:

- **Fikir 1: En erken başlayan etkinliği seçmek.** Basit bir karşı örnek: Çok erken başlayan ama tüm günü kaplayan bir etkinlik $[0, 100)$ olsun. Diğer etkinlikler $[1, 2), [2, 3), [3, 4)$ olsun. En erken başlayanı seçersek sadece 1 etkinlik yapabiliriz, oysa diğerlerini seçerek 3 etkinlik yapmak mümkündür. Bu fikir **yanlış**.

- **Fikir 2: En kısa süren ($e_i - s_i$ en küçük) etkinliği seçmek.** Karşı örnek: Etkinlikler $[0,5), [4,6), [5,10)$ olsun. En kısa süren $[4,6)$ aralığıdır (2 birim). Onu seçersek diğer iki etkinlikle de çakışır ve toplam 1 etkinlik yapılmış olur. Oysa $[0,5)$ ve $[5,10)$ seçilerek 2 etkinlik yapılabilirdi. Bu fikir de **yanlış**.

- **Fikir 3: Diğerleriyle en az çakışan etkinliği seçmek.** Makul görünse de karmaşık çizelgelerde bu kuralı da bozan karşı örnekler kurulabilir.

- **Fikir 4: En erken biten etkinliği seçmek.** Gerekçe: Bir etkinliği ne kadar erken bitirirsek, salonda geriye kalan etkinlikler için gelecekte o kadar çok serbest zaman kalır. Sonraki adımlara en geniş hareket alanını bırakan seçim budur.

Şimdi Fikir 4'ün doğruluğunu değiş-tokuş argümanıyla kanıtlayalım.

**Teorem 27.6.** *Aralık Seçimi Problemi'nde, mevcut uyumlu etkinlikler arasından daima bitiş zamanı $e_i$ en küçük olan etkinliği seçen greedy strateji optimal sonucu verir.*

*Kanıt.* Greedy algoritmamızın seçtiği etkinlik kümesi $G = \{g_1, g_2, \ldots, g_k\}$ olsun. Bu etkinlikler bitiş zamanlarına göre sıralıdır; dolayısıyla $e(g_1) \leq e(g_2) \leq \cdots \leq e(g_k)$ geçerlidir.

Varsayalım ki $G$ optimal olmasın. O halde $G$'den kesinlikle daha fazla sayıda etkinlik içeren bir optimal çözüm $O = \{o_1, o_2, \ldots, o_m\}$ vardır ($m > k$). $O$'daki etkinlikler de başlangıç/bitiş sıralarına göre dizilmiş olsun.

$G$ ve $O$ kümelerini baştan itibaren karşılaştıralım. Diyelim ki ilk $r-1$ etkinlik aynı olsun, yani $g_1 = o_1, \ldots, g_{r-1} = o_{r-1}$, fakat $r$. etkinlikte ayrılsınlar ($g_r \neq o_r$).

Algoritmamızın tanımı gereği, $g_{r-1}$ bittikten sonra başlayabilecek tüm geçerli etkinlikler arasında bitiş zamanı en küçük olan $g_r$ olarak seçilmiştir. Dolayısıyla:

$$e(g_r) \leq e(o_r)$$

olmak zorundadır.

Şimdi $O$ kümesinden $o_r$'yi çıkarıp yerine $g_r$'yi yerleştirelim. Yeni kümemiz $O' = (O \setminus \{o_r\}) \cup \{g_r\}$ olsun.

- $g_r$, $g_{r-1} = o_{r-1}$ bittikten sonra başladığı için kendinden öncekilerle çakışmaz.

- Kendinden sonraki etkinlik olan $o_{r+1}$, $o_r$ bittikten sonra başlıyordu ($s(o_{r+1}) \geq e(o_r)$). Eşitsizlikten biliyoruz ki $e(g_r) \leq e(o_r)$, dolayısıyla $s(o_{r+1}) \geq e(g_r)$ de sağlanır. Yani $g_r$, $o_{r+1}$ ile de çakışmaz.

Böylece $O'$ kümesi de geçerli ve çakışmasız bir çözümdür. Üstelik eleman sayısı $|O'| = |O| = m$'dir; hiçbir kayıp yaşanmamıştır.

Bu değiş-tokuş işlemini her farklılık için tekrarlarsak, $O$ çözümünü adım adım $G$'ye dönüştürebiliriz. $k$ adımdan sonra $O$ kümesinin ilk $k$ elemanı $G$ ile tamamen aynı olur. Eğer $m > k$ olsaydı, $O$ kümesinde $g_k$'den sonra başlayan bir $o_{k+1}$ etkinliği kalmış olurdu. Fakat algoritmamız $g_k$'den sonra seçilebilecek hiçbir etkinlik kalmadığında durmuştu; bu bir çelişkidir. Dolayısıyla $m > k$ olamaz, $k = m$ olmalıdır. $G$ kümesi optimaldir. □

```
function intervalScheduling(activities)
    sort activities in ascending order by end time

    selectedActivities = empty list
    lastEndTime = -infinity

    for each (start, end) in activities do
        if start >= lastEndTime then
            append (start, end) to selectedActivities
            lastEndTime = end
        end if
    end for

    return selectedActivities
end function

activities = [(1, 4), (3, 5), (0, 6), (5, 7), (3, 8), (5, 9), (6, 10), (8,
     11), (8, 12), (2, 13), (12, 14)]
result = intervalScheduling(activities)
print "Maximum number of selected activities: " + length of result
```

Bu algoritmanın zaman karmaşıklığı diziyi sıralamaktan dolayı $O(n \log n)$, aralıkları tek geçişte taramaktan dolayı $O(n)$'dir; toplamda $O(n \log n)$ zaman alır.

### 27.2.2 Bozuk Para Değişimi (*Coin Change*) ve Kanonik Sistemler

Bölümün başında $1, 3, 4$ para sisteminde greedy yaklaşımın hatalı sonuç verdiğini gördük. Peki günlük hayatta kullandığımız Türk Lirası ($1, 5, 10, 25, 50$ Kuruş ve 1 TL) veya ABD Doları ($1, 5, 10, 25$ Cent) sistemlerinde para üstü verirken greedy yaklaşım neden her zaman en az sayıda para kullanır?

**Tanım 27.7** (Kanonik Para Sistemi)**.** *Bir madeni para kümesinde, her $V$ tamsayı değeri için greedy algoritmanın ürettiği madeni para sayısı, teorik olarak mümkün olan minimum madeni para sayısına eşit oluyorsa, bu sisteme* ***kanonik para sistemi*** *(canonical coin system) denir.*

Bir para sisteminin kanonik olmasını sağlayan temel yeter koşullardan biri, Bölüm 11'de gördüğümüz bölünebilme ilişkisinin ardışık elemanlar arasında sağlanması, yani paraların birbirinin tam katı olmasıdır.

**Teorem 27.8.** *Para değerleri $c_1 < c_2 < \cdots < c_k$ olsun; her $1 \le i < k$ için $c_i \mid c_{i+1}$ (her para birimi bir öncekinin tam katı) ve $c_1 = 1$ sağlansın. Bu durumda greedy algoritması her $V$ değeri için daima optimaldir.*

*Kanıt.* İspatı bir çelişki ve Bölüm 18'de ele aldığımız değişmez (invaryant) mantığıyla kuralım. Herhangi bir optimal çözümde, $c_i$ değerindeki madeni paradan en fazla kaç adet bulunabilir? $c_{i+1}$, $c_i$'nin tam katı olduğuna göre $q_i = c_{i+1}/c_i \ge 2$ bir tamsayıdır. Eğer optimal çözümde $q_i$ veya daha fazla sayıda $c_i$ parası kullanılmış olsaydı, bu $q_i$ adet $c_i$ parasını sistemden çıkarıp yerine sadece **1 adet** $c_{i+1}$ koyabilirdik ($q_i \cdot c_i = c_{i+1}$). Bu hamle toplam değeri değiştirmez, fakat kullanılan para sayısını $q_i - 1 \ge 1$ adet azaltarak çözümü daha iyi hale getirirdi. Bu durum çözümün optimal olmasıyla çelişir.

Dolayısıyla herhangi bir optimal çözümde $c_i$ parasının adedi $a_i$ olmak üzere:

$$a_i \le q_i - 1 = \frac{c_{i+1}}{c_i} - 1$$

olmalıdır. Bu koşul altında paraların toplam değeri:

$$\sum_{j=1}^{i} a_j c_j \le \sum_{j=1}^{i} \left( \frac{c_{j+1}}{c_j} - 1 \right) c_j$$

Bu toplam sadeleştirildiğinde:

$$(c_2 - c_1) + (c_3 - c_2) + \cdots + (c_{i+1} - c_i) = c_{i+1} - c_1 < c_{i+1}$$

elde edilir.

Bu eşitsizlik şunu gösterir: $c_{i+1}$'den küçük olan tüm paraların mümkün olan en fazlası bir araya getirilse bile, toplamları bir sonraki para birimi olan $c_{i+1}$ değerine ulaşamaz. Dolayısıyla ödenmesi gereken tutar $V \ge c_{i+1}$ olduğunda, $c_{i+1}$ kullanılmadan sadece daha küçük paralarla ödeme yapılırsa kesinlikle daha fazla sayıda madeni para harcanır. Bu nedenle $V$'yi aşmayan en büyük parayı ($c_{i+1}$) seçmek zorunludur; bu da tam olarak greedy algoritmanın izlediği kuraldır. □

**Örnek 27.9.** *Bir bilgisayar sisteminde dosya boyutları için $\{1, 2, 4, 8, 16, 32\}$ baytlık bloklar kullanılmaktadır.* 59 *baytlık bir veri en az kaç blokta depolanabilir?*

*Çözüm.* Blok boyutları 2'nin kuvvetleridir ve her biri bir sonrakini böler ($2^k \mid 2^{k+1}$). Yukarıdaki teorem gereği greedy yaklaşım optimaldir. En büyükten başlayarak:

$$\begin{aligned} 59 &= 1 \times 32 + 27 \\ 27 &= 1 \times 16 + 11 \\ 11 &= 1 \times 8 + 3 \\ 3 &= 1 \times 2 + 1 \\ 1 &= 1 \times 1 + 0 \end{aligned}$$

Toplamda 5 blok kullanılır ($32 + 16 + 8 + 2 + 1 = 59$). Bu işlem, Bölüm 15'te gördüğümüz taban dönüşümüyle 59 sayısının ikili tabandaki karşılığı olan $(111011)_2$ gösterimindeki 1'lerin sayısına (Hamming ağırlığı) doğrudan karşılık gelir. □

### 27.2.3 Kesirli Sırt Çantası (*Fractional Knapsack*)

0-1 Sırt Çantası probleminde eşyalar bölünemediği için greedy yaklaşımın yetersiz kaldığını gördük. Eşyaların istenilen oranda bölünebildiği (*fractional*) durumda ise greedy strateji doğrudan optimal çözümü verir.

**Teorem 27.10.** *Kesirli Sırt Çantası probleminde, birim değer oranı $v_i/w_i$ en yüksek olan eşyadan başlanarak çantanın alabildiği kadarını alma kuralı daima optimal çözümü verir.*

*Kanıt.* Değiş-tokuş argümanı ile ispatlayalım. Eşyaları birim değerlerine göre azalan sırada dizelim:

$$\frac{v_1}{w_1} \geq \frac{v_2}{w_2} \geq \cdots \geq \frac{v_n}{w_n}$$

Greedy algoritmamız $G$, çantayı sırasıyla $1, 2, \ldots$ numaralı eşyalarla doldurur. Kalan son boşluğa gerekirse bir eşyanın kesirli bir kısmını yerleştirir.

Farklı bir optimal çözüm $O$ olduğunu varsayalım. Bu durumda $O$ çözümü, $G$'nin çantaya koyduğu yüksek birim değerli bir eşyadan (örneğin $i$ numaralı eşya) daha az miktarda almış ve onun yerine daha düşük birim değerli bir başka eşyadan (örneğin $j$ numaralı eşya, $j > i$) $\Delta w$ ağırlığında fazladan koymuştur.

O halde $O$ çantasından $j$ eşyasına ait $\Delta w$ kadar ağırlığı çıkarıp, yerine henüz tamamen tükenmemiş olan $i$ eşyasından $\Delta w$ kadar ekleyelim. Çantanın toplam ağırlığı değişmez. Toplam değerdeki değişim ise:

$$\Delta V = \Delta w \cdot \left( \frac{v_i}{w_i} - \frac{v_j}{w_j} \right)$$

olur. Sıralamadan dolayı $v_i/w_i \geq v_j/w_j$ olduğundan $\Delta V \geq 0$'dır. Yani çantadaki toplam değer azalmaz, bilakis artabilir. Bu değiş-tokuş adımıyla $O$ çözümü $G$'ye dönüştürülebilir. Buradan, greedy yaklaşımın kesirli sırt çantası için kesinlikle optimal olduğu görülür. □

### 27.2.4 MST (Minimum Spanning Tree) ve Kesit Özelliği

Greedy algoritmalarının graf teorisindeki en önemli uygulamalarından biri, Bölüm 28'de ayrıntılı biçimde işlenecek olan graflar üzerindeki **MST (minimum spanning tree)** problemidir.

Ağırlıklı, yönsüz ve bağlı bir $G = (V, E)$ grafında, tüm düğümleri birbirine bağlayan ve toplam kenar ağırlığı en küçük olan ağaca MST denir. Kruskal ve Prim algoritmaları bu problemi çözen iki temel greedy yaklaşımdır:

- **Kruskal Algoritması:** Grafın tüm kenarlarını ağırlıklarına göre küçükten büyüğe sıralar. Çevrim (*cycle*) oluşturmayan en hafif kenarı her adımda kümeye ekler.
- **Prim Algoritması:** Tek bir düğümle başlar. Her adımda mevcut ağaç düğümlerini ağaçta olmayan düğümlere bağlayan kenarların en hafif olanını seçerek ağacı büyütür.

Her iki algoritmanın doğruluğu, graf teorisindeki **Kesit Özelliği**'ne (*Cut Property*) dayanır.

**Tanım 27.11** (Graf Kesiti)**.** *Bir $G = (V, E)$ grafının düğüm kümesi $V$'nin, $S$ ve $V \setminus S$ şeklinde boş olmayan iki ayrık alt kümeye bölünmesine bir **kesit** (cut) denir. Bir ucu $S$'de, diğer ucu $V \setminus S$'de olan kenarlara bu kesiti **kesen kenarlar** denir.*

**Teorem 27.12** (Kesit Özelliği / Cut Property)**.** *Bir graftaki herhangi bir $(S, V \setminus S)$ kesitini ele alalım. Bu kesiti kesen kenarlar arasında ağırlığı kesin olarak en küçük olan $e = (u, v)$ kenarı, grafın **tüm** minimum kapsayan ağaçlarında yer almak zorundadır.*

*Kanıt.* Değiş-tokuş argümanını burada da uygulayabiliriz. Varsayalım ki $T$, bu en hafif $e$ kenarını içermeyen bir MST olsun.

1. $T$ bir kapsayan ağaç olduğu için $u$ ile $v$ arasında $T$ üzerinde tek bir basit yol vardır.
2. $u \in S$ ve $v \in V \setminus S$ olduğundan, bu yol $S$ kümesinden çıkıp $V \setminus S$ kümesine geçmek zorundadır. Dolayısıyla bu yol üzerinde kesiti kesen en az bir başka $e' = (u', v')$ kenarı bulunmalıdır ($e' \neq e$).
3. Şimdi $T$ ağacına $e$ kenarını ekleyelim. Ağaçta bir çevrim oluşur. Bu çevrim hem $e$ hem de $e'$ kenarlarını içerir.
4. Çevrimden $e'$ kenarını çıkarıp yeni bir $T' = (T \setminus \{e'\}) \cup \{e\}$ grafı elde edelim.
5. $T'$ hâlâ tüm düğümleri birbirine bağlar ve $|V|-1$ kenara sahiptir, yani geçerli bir kapsayan ağaçtır.
6. Toplam ağırlıktaki değişim:

   $$w(T') = w(T) - w(e') + w(e)$$

   Teoremde $e$, kesiti kesen kenarlar arasında ağırlığı **kesin olarak** en küçük kenar olarak seçilmişti. $e'$ de kesiti kesen bir kenar ve $e' \neq e$ olduğundan $w(e) < w(e')$'dir. Dolayısıyla

   $$w(T') = w(T) - w(e') + w(e) < w(T)$$

   olur.

7. Bu ise $T$'nin minimum ağırlıklı olmasıyla **çelişir**; çünkü $T$'den daha küçük ağırlıklı bir kapsayan ağaç ($T'$) elde ettik. Demek ki varsayım yanlıştır: $e$ kenarını içermeyen bir minimum kapsayan ağaç yoktur, yani $e$ **her** minimum kapsayan ağaçta yer alır.

Bu teorem, Kruskal ve Prim algoritmalarının her adımda yaptığı yerel seçimlerin (kesiti kesen en hafif kenarı seçmenin) küresel bir MST inşa edeceğini garanti eder. □

### 27.2.5 Kapsamlı Bir Problem: Huffman Kodlaması Sezgisi

Veri sıkıştırmada kullanılan Huffman algoritması, greedy mantığının bir diğer başarılı uygulama alanıdır. Metinde sık geçen karakterlere kısa kodlar, seyrek geçen karakterlere ise daha uzun kodlar atanarak toplam bit maliyeti azaltılır.

**Örnek 27.13.** *Bir metinde 4 farklı karakter şu frekanslarla geçmektedir:* $A : 40$*,* $B : 25$*,* $C : 20$*,* $D : 15$*. Bu karakterleri ikili ağaç yapısında kodlayarak ağırlıklı ortalama bit uzunluğunu minimize ediniz.*

*Çözüm.* **Greedy Kuralı:** Her adımda frekansı en küçük olan iki düğümü birleştirip tek bir yeni düğüm oluşturulur (bu iki düğüm ağacın en derininde, yani en uzun koda sahip kardeş yapraklar olacaktır).

1. En küçük iki frekans $C(20)$ ve $D(15)$'tir. Bunlar birleştirilir: Yeni düğüm $CD(35)$. Kalan frekanslar: $A(40), B(25), CD(35)$.

2. Şimdi en küçük iki frekans $B(25)$ ve $CD(35)$'tir. Birleştirildiğinde: $BCD(60)$. Kalanlar: $A(40), BCD(60)$.

3. Son iki düğüm birleştirilir: $ABCD(100)$ (Kök düğüm).

Kod uzunlukları:

- $A$ kökün doğrudan çocuğu: derinlik 1 bit.
- $B$, $BCD$'nin çocuğu: derinlik 2 bit.
- $C$ ve $D$, $CD$'nin çocukları: derinlik 3 bit.

Toplam harcanan bit: $40 \times 1 + 25 \times 2 + 20 \times 3 + 15 \times 3 = 40 + 50 + 60 + 45 = 195$ bit. Eğer her karaktere sabit 2 bit verilseydi toplam $100 \times 2 = 200$ bit harcanacaktı. Greedy seçim, frekansı düşük olanları ağacın daha derin dallarına yerleştirerek toplam maliyeti en aza indirmiştir. □

## Bölüm Özeti

- **Greedy Yaklaşımı:** Her adımda o an en avantajlı görünen seçimi yapar; geri dönüş (backtracking) yapmaz. Hızlıdır fakat her problemde optimal sonuca ulaştırmaz.
- **Doğruluk Şartları:** Bir problemin greedy ile çözülebilmesi için *greedy seçim özelliği* ve *optimal alt yapı* şartlarının sağlanması gerekir.
- **İspat Yöntemi:** Bir greedy fikri uygulamadan önce **Değiş-Tokuş Argümanı** ile test edilmelidir. Varsayımsal optimal bir çözümün değer kaybetmeksizin adım adım greedy çözüme dönüştürülebildiği gösterilmelidir.
- **Tipik Alanlar:** Aralık seçimi (bitiş zamanına göre sıralama), kanonik para sistemleri, kesirli sırt çantası ve minimum kapsayan ağaçlar (Kruskal/Prim) greedy yaklaşımının kesin sonuç verdiği başlıca alanlardır.

# Bölüm 28

# Graflar ve Arama

## 28.1 Graflar ve Temsiller

### 28.1.1 Motivasyon: Nesneler Arasındaki İlişkileri Nasıl Modelleriz?

Algoritma problemlerinde ve modelleme süreçlerinde ikili ilişkiler belirleyici bir rol oynar: şehirler arasındaki kara yolları, bilgisayarlar arasındaki ağ bağlantıları, bir projedeki işlerin öncelik sıraları veya sosyal ağlardaki arkadaşlıklar buna örnektir. Bu tür sistemlerin ortak paydası, birbirinden bağımsız bir grup *nesne* ile bu nesneler arasındaki *bağlantılardır*.

Matematikte ve bilgisayar biliminde bu ilişki ağını soyutlamanın temel aracı **graf** (graph) yapısıdır.

Dört öğrenciden oluşan bir topluluk düşünelim: Ali ($A$), Burak ($B$), Cansu ($C$) ve Deniz ($D$). Aralarındaki arkadaşlık ilişkileri şöyle verilsin:

- $A$ ile $B$ arkadaştır.
- $A$ ile $C$ arkadaştır.
- $B$ ile $C$ arkadaştır.
- $C$ ile $D$ arkadaştır.

Bu yapıyı bir kâğıda çizerken her öğrenciyi bir nokta, her arkadaşlığı ise iki nokta arasına çizilen bir çizgi olarak gösteririz. $D$'nin doğrudan yalnızca $C$ ile bağlı olduğunu; $A$ ve $B$'ye ise $C$ üzerinden dolaylı olarak ulaştığını hemen görebiliriz. Bu çizim, graf kavramının en somut karşılığıdır.

**Tanım 28.1** (Graf)**.** *Bir $G$ grafı, iki kümenin sıralı ikilisidir: $G = (V, E)$.*

- *$V$: Boş olmayan **düğüm** (node veya köşe - vertex) kümesidir. Düğüm sayısı genellikle $n = |V|$ ile gösterilir.*

- *E: Düğümler arasındaki bağlantıları belirten **kenar** (edge) kümesidir. Kenar sayısı genellikle $m = |E|$ ile gösterilir.*

Grafları kenar yapılarına göre üç ana grupta toplarız:

1. **Yönsüz Graf (Undirected Graph):** Kenarların yönü yoktur. $u$ ile $v$ arasındaki kenar $\{u, v\}$ kümesi olarak ifade edilir. $u$'dan $v$'ye gitmek ile $v$'den $u$'ya gitmek farksızdır (karşılıklı arkadaşlık ilişkisi gibi).

2. **Yönlü Graf (Directed Graph / Digraph):** Kenarlar sıralı ikililerden oluşur: $(u, v) \in E$. Bu kenar $u$'dan başlar ve $v$'de biter; $u \to v$ olarak gösterilir. $v$'den $u$'ya bir geçiş bulunmak zorunda değildir (tek yönlü yollar veya web sayfalarındaki köprüler gibi).

3. **Ağırlıklı Graf (Weighted Graph):** Her kenara $w : E \to \mathbb{R}$ fonksiyonu ile bir maliyet, uzunluk veya kapasite atanmıştır.

Bir düğüme bağlı kenar sayısına o düğümün **derecesi** (degree) denir ve $\deg(v)$ ile gösterilir. Yönlü graflarda ise içeri derece ($\deg^-(v)$, düğüme gelen kenarlar) ve dışarı derece ($\deg^+(v)$, düğümden çıkan kenarlar) ayrımı yapılır.

**Teorem 28.2** (El Sıkışma Lemması / Handshaking Lemma)**.** *Herhangi bir yönsüz $G = (V, E)$ grafında, tüm düğümlerin dereceleri toplamı kenar sayısının tam iki katına eşittir:*

$$\sum_{v \in V} \deg(v) = 2|E|$$

*Kanıt.* Bölüm 10'da ele aldığımız çifte sayım ilkesiyle yaklaşalım. Her $e = \{u, v\}$ kenarı iki uç noktaya sahiptir. Kenarları tek tek gezdiğimizde her kenar iki kez sayılır: bir kez $u$'nun derecesini hesaplarken, bir kez de $v$'nin derecesini hesaplarken. Dolayısıyla derece toplamı doğrudan $2|E|$ değerini verir. □

**Örnek 28.3.** *$n = 5$ düğümlü bir yönsüz grafta düğüm dereceleri sırasıyla $3, 3, 3, 2, 2$ olabilir mi?*

*Çözüm.* Derecelerin toplamını hesaplayalım: $3 + 3 + 3 + 2 + 2 = 13$. El Sıkışma Lemması gereği bu toplamın $2|E|$ olması, yani çift sayı çıkması zorunludur. 13 tek sayı olduğundan böyle bir graf çizilemez. □

### 28.1.2 Bilgisayarda Grafların Saklanması

Bir grafı kod içinde işleyebilmek adına bellekte uygun bir veri yapısıyla tutmalıyız. En sık başvurulan iki yöntem *adjacency matrix* ve *adjacency list*dir.

### 1. Adjacency Matrix

Düğümleri $0, 1, \ldots, n-1$ olarak numaralandıralım. Adjacency matrix $n \times n$ boyutunda iki boyutlu bir dizidir (genellikle $A$ veya $adj$ ile gösterilir):

$$A[u][v] = \begin{cases} 1, & \text{eğer } (u,v) \in E \text{ ise} \\ 0, & \text{aksi halde} \end{cases}$$

Ağırlıklı graflarda 1 yerine kenarın ağırlığı $w(u,v)$, kenar yoksa $\infty$ veya 0 değeri saklanır.

**Avantajı:** İki $u$ ve $v$ düğümü arasında doğrudan kenar olup olmadığını $O(1)$ sürede sorgularız.

**Dezavantajı:** Bellek kullanımı graf seyrek de olsa her zaman $O(n^2)$'dir. $n = 10^5$ gibi büyük değerlerde matris belleğe sığmaz (yaklaşık 40 GB alan gerekir). Ayrıca bir düğümün tüm komşularını gezmek her durumda $O(n)$ sürer.

### 2. Adjacency List

Her $u \in V$ düğümü için, yalnızca $u$'ya komşu olan düğümleri saklayan dinamik bir dizi (veya liste) tutulur.

**Bellek Kullanımı:** Yönsüz bir grafta her kenar iki listede yer alır, yönlü grafta ise bir listede. Toplam bellek gereksinimi $O(n+m)$ düzeyindedir.

**Avantajı:** Bellek açısından son derece tutumludur; bir düğümün tüm komşularını gereksiz düğümleri taramadan, tam olarak $\deg(u)$ adımda gezmeyi sağlar.

```cpp
// C++ ile Komsuluk Listesi Temsili
#include <stdio.h>
#include <vector>

const int MAXN = 100005;
std::vector<int> adj[MAXN]; // adj[u], u dugumunun komsularini tutar

void kenar_ekle(int u, int v) {
    adj[u].push_back(v);
    adj[v].push_back(u); // Yonsuz graf oldugu icin iki yonlu eklenir
}
```

```
n = 4
m = 4
Initialize adj as an array of n empty lists

function addEdge(u, v)
    append v to adj[u]
    append u to adj[v]
end function
```

```
addEdge(0, 1)
addEdge(0, 2)
addEdge(1, 2)
addEdge(2, 3)
```

**Örnek 28.4.** *$n = 4$ düğümlü ve kenarları $E = \{(0,1),(0,2),(1,2),(2,3)\}$ olan yönsüz bir grafın adjacency matrix'ini ve adjacency list'ini oluşturunuz.*

*Çözüm.* Adjacency matrix $4 \times 4$ boyutundadır:

$$A = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Matrisin köşegene göre simetrik olduğuna dikkat ediniz; bu durum tüm basit yönsüz graflar için geçerlidir.

Adjacency list ise her düğümün komşularını ayrı ayrı tutar:

$$\begin{aligned} \text{adj}[0] &= [1, 2] \\ \text{adj}[1] &= [0, 2] \\ \text{adj}[2] &= [0, 1, 3] \\ \text{adj}[3] &= [2] \end{aligned}$$

□

**Dikkat:** $n = 10^5$ ve $m = 2 \cdot 10^5$ olan tipik bir yarışma probleminde adjacency matrix tanımlamak bellek aşımına (Memory Limit Exceeded) yol açar. Kenar sayısının düğüm sayısının karesine kıyasla az kaldığı durumlarda (*seyrek graf* / sparse graph: $m \ll n^2$) adjacency list tercih edilmelidir.

## 28.2 Derinlik Öncelikli Arama (DFS)

### 28.2.1 Motivasyon: Labirentten Çıkış ve Geri İzleme

Bir labirentte yol aradığımızı varsayalım. Bir kavşağa geldiğimizde kollardan birini seçip gidebildiğimiz noktaya kadar ilerleriz. Bir çıkmaz sokağa vardığımızda ise geldiğimiz yoldan adım adım geri çekilir, henüz denemediğimiz ilk yol ayrımına döner ve oradan yeni bir patikayı keşfederiz.

Bu yaklaşım, **Derinlik Öncelikli Arama**nın (Depth-First Search - DFS) temel çalışma mantığıdır. DFS, bulunduğu düğümden başlayarak keşfedebileceği en derin noktaya kadar ilerler; ilerleyecek düğüm kalmadığında ise Bölüm 26'da gördüğümüz rekürsiyon yapısına uygun biçimde *geri izleme* (backtracking) yaparak henüz taranmamış dallara yönelir.

### 28.2.2 Algoritmanın İşleyişi ve Ziyaret Mantığı

Graflarda döngüler (cycle) bulunabildiğinden, aynı düğümü tekrar tekrar ziyaret etmemek için her düğüm adına bir işaretçi dizisi (`visited`) tutulur.

Algoritma şu adımlarla çalışır:

1. $u$ düğümünü ziyaret edildi olarak işaretle.
2. $u$'nun tüm komşularını ($v \in \text{adj}[u]$) sırayla tara.
3. Eğer $v$ daha önce ziyaret edilmemişse, $v$ için rekürsif olarak DFS fonksiyonunu çağır.
4. $u$'nun ziyaret edilmemiş komşusu kalmadığında fonksiyon tamamlanır ve bir önceki çağrıya geri döner.

```
bool vis[MAXN];

void dfs(int u) {
    vis[u] = true;
    for (int v : adj[u]) {
        if (!vis[v]) {
            dfs(v);
        }
    }
}
```

### 28.2.3 Zaman ve Bellek Karmaşıklığı

**Teorem 28.5.** *$G = (V, E)$ grafında adjacency list ile çalışan DFS algoritmasının zaman karmaşıklığı $O(|V| + |E|)$, bellek karmaşıklığı ise en kötü durumda $O(|V|)$'dir.*

*Kanıt.* Her düğüm için `vis[u] = true` ataması yapıldığından hiçbir düğüm çağrı yığınına birden fazla kez girmez. Dolayısıyla her $u$ düğümünün kenar listesi tam olarak bir kez taranır. Bütün düğümlerin komşuluk listeleri uzunlukları toplamı yönsüz grafta $2|E|$, yönlü grafta $|E|$ kadardır. Kenar taramaları $O(|E|)$, başlangıç hazırlıkları ise $O(|V|)$ sürer. Toplam süre $O(|V| + |E|)$ olur.

Grafın tek bir uzun zincir (yol grafı) olduğu en kötü senaryoda, rekürsiyon derinliği $|V|$'ye ulaşır ve çağrı yığını $O(|V|)$ bellek tüketir. □

### 28.2.4 DFS'in Temel Uygulamaları

#### 1. Bağlı Bileşenlerin (Connected Components) Sayısını Bulma

Yönsüz bir grafta, aralarında en az bir yol bulunan düğümler kümesine *bağlı bileşen* denir. Graf birbirinden kopuk parçalardan oluşabilir. DFS yardımıyla her

bileşeni ayrı ayrı tespit edebiliriz:

```
componentCount = 0
visited = array of size n initialized to false

FOR i = 0 TO n - 1 DO
    IF NOT visited[i] THEN
        DFS(i)
        componentCount = componentCount + 1
    END IF
END FOR
```

### 2. İki Renklendirilebilirlik (Bipartite / İki Kümeli Graf Kontrolü)

Bir grafın düğümleri, komşu olan hiçbir iki düğüm aynı renge sahip olmayacak biçimde iki renkle (örneğin 0 ve 1) boyanabiliyorsa bu yapıya **iki kümeli graf** (bipartite graph) denir.

**Teorem 28.6.** *Bir grafın iki kümeli olması için gerek ve yeter koşul, içinde* ***tek uzunlukta döngü barındırmamasıdır.***

**Örnek 28.7.** *DFS kullanarak bir grafın iki kümeli olup olmadığını belirleyen yaklaşımı kurunuz.*

*Çözüm.* Her düğüme $renk \in \{0, 1\}$ değeri atayalım. Başlangıçta tüm düğümler renksizdir ($-1$). Başlangıç düğümüne 0 rengi verilir. $u$ düğümünden komşusu $v$'ye geçerken:

- Eğer $v$ boyanmamışsa, ona $1 - renk[u]$ rengi atanır ve DFS $v$'den devam eder.
- Eğer $v$ boyanmışsa ve $renk[v] == renk[u]$ ise komşu iki düğüm aynı renge denk gelmiştir. Bu durum tek uzunlukta bir döngüye işaret eder; graf iki kümeli değildir.

Tüm düğümler çelişki üretmeden boyanabiliyorsa graf iki kümelidir. □

**Dikkat:** Graf birden fazla bağlı bileşenden oluşuyorsa yalnızca `dfs(0)` çağırmak diğer parçaları açıkta bırakır. Grafın tamamını kapsamak için ana döngüde 0'dan $n - 1$'e kadar tüm düğümler taranmalıdır.

## 28.3 Genişlik Öncelikli Arama (BFS)

### 28.3.1 Motivasyon: Dalgakıran ve En Az Adım Problemi

Durgun bir suya bir taş bırakıldığında dalgalar merkezden dışa doğru halkalar halinde yayılır: önce 1 birim uzaklıktaki noktalar ıslanır, ardından 2 birim, sonra 3 birim...

Ağırlıksız bir grafta "A düğümünden B düğümüne en az kaç kenar geçerek gidebilirim?" sorusunu çözerken bu dalga mantığı devreye girer. DFS tek bir patikayı sonuna kadar zorlarken, **Genişlik Öncelikli Arama** (Breadth-First Search - BFS) kaynak düğümden başlayarak eşit mesafedeki tüm düğümleri seviye seviye keşfeder.

### 28.3.2 Kuyruk (Queue) ile Seviye Seviye İlerleme

Arama katman katman ilerlediği için önce keşfedilen düğümlerin komşuları daha önce incelenmelidir. Bu kural, Bölüm 25'te ele aldığımız ve *İlk Giren İlk Çıkar* (FIFO) prensibiyle çalışan **kuyruk** (queue) veri yapısını gerektirir.

```
#include <queue>

int dist[MAXN]; // Baslangica olan en kisa mesafe (kenar sayisi)

void bfs(int baslangic, int n) {
    for (int i = 0; i < n; i++) dist[i] = -1;

    std::queue<int> q;
    q.push(baslangic);
    dist[baslangic] = 0;

    while (!q.empty()) {
        int u = q.front();
        q.pop();

        for (int v : adj[u]) {
            if (dist[v] == -1) { // Ilk kez karsilasilan dugum
                dist[v] = dist[u] + 1;
                q.push(v);
            }
        }
    }
}
```

### 28.3.3 BFS'in En Kısa Yol Özelliği

**Teorem 28.8.** *Tüm kenar ağırlıklarının 1 (veya birbirine eşit) olduğu bir grafta BFS, başlangıç düğümünden erişilebilen her $v$ düğümü için en az kenar içeren en kısa yolu (shortest path) bulmayı garanti eder.*

*İspat Taslağı.* Kuyruktaki düğümlerin mesafeleri monoton olarak artar. Başlangıçta kuyrukta yalnızca $dist = 0$ olan kaynak düğüm yer alır. $dist = k$ olan tüm düğümler tüketilmeden hiçbir $dist = k+1$ düğümünün komşularına geçilmez. Dolayısıyla bir $v$ düğümüne ilk ulaşıldığı an, ona en az sayıda kenar üzerinden varılan

andır. Daha sonra bulunabilecek hiçbir alternatif yol $dist[v]$ değerinden daha az kenar içeremez. □

**Örnek 28.9.** *$n = 6$ düğümlü bir grafta kenarlar şunlardır:*

$$E = \{(0,1), (0,2), (1,3), (2,3), (3,4), (2,5)\}$$

*Başlangıç düğümü 0 seçildiğinde her düğümün 0'a olan BFS mesafesini adım adım çıkarınız.*

*Çözüm.* Kuyruğun adımlarını izleyelim:

1. Başlangıç: $dist[0] = 0$. Kuyruk: [0].
2. 0 çıkarılır. Komşuları 1 ve 2: $dist[1] = 1, dist[2] = 1$. Kuyruk: $[1, 2]$.
3. 1 çıkarılır. Komşuları 0 (ziyaret edildi), 3: $dist[3] = dist[1] + 1 = 2$. Kuyruk: $[2, 3]$.
4. 2 çıkarılır. Komşuları 0 (ziyaret edildi), 3 (ziyaret edildi), 5: $dist[5] = dist[2] + 1 = 2$. Kuyruk: $[3, 5]$.
5. 3 çıkarılır. Komşusu 4: $dist[4] = dist[3] + 1 = 3$. Kuyruk: $[5, 4]$.
6. 5 ve 4 çıkarılır; ziyaret edilmemiş komşu kalmamıştır.

Nihai mesafeler $dist = [0, 1, 1, 2, 3, 2]$ olarak elde edilir. □

### 28.3.4 0-1 BFS Sezgisi

Kenar ağırlıkları yalnızca 0 veya 1 olabiliyorsa standart BFS doğrudan uygulanamaz; zira 0 ağırlıklı bir kenardan geçmek mesafeyi artırmaz. Ancak tam bir Dijkstra algoritması ($O(m \log n)$) çalıştırmak yerine deque (`deque`) kullanarak problemi $O(n + m)$ sürede çözebiliriz:

- 0 ağırlıklı bir kenardan $v$'ye geçiliyorsa: $dist[v] = dist[u]$ olur ve düğüm kuyruğun **önüne** (`push_front`) eklenir.
- 1 ağırlıklı bir kenardan geçiliyorsa: $dist[v] = dist[u] + 1$ olur ve düğüm kuyruğun **arkasına** (`push_back`) eklenir.

Böylelikle kuyruktaki düğümlerin mesafe farkı hiçbir zaman 1'i aşmaz ve monotonluk korunur.

**Dikkat:** Kenar ağırlıkları birbirinden farklı pozitif sayılar olan bir grafta "en az kenar sayısı = en kısa yol" kabulü geçerli değildir. BFS yalnızca ağırlıksız veya tüm kenar ağırlıkları birbirine eşit graflarda en kısa yolu verir.

## 28.4 Topolojik Sıralama (Topological Sort)

### 28.4.1 Motivasyon: Ders Önkoşulları ve Bağımlılık Zincirleri

Bir lisans programında alınması gereken dersleri düşünelim:

- "Algoritmalar" dersini alabilmek için önce "Programlamaya Giriş" tamamlanmalıdır.
- "Yapay Zeka" dersi için hem "Algoritmalar" hem de "Lineer Cebir" gereklidir.

Bu dersler kural ihlali yapmadan hangi sırayla tamamlanabilir?

Müfredatta "A için B gerekli, B için C gerekli, C için de A gerekli" şeklinde döngüsel bir bağımlılık bulunuyorsa süreci başlatmak imkânsızdır. Bağımlılıkların çözülebilir olması için sistemin bir **Yönlü Döngüsüz Graf** (Directed Acyclic Graph - DAG) oluşturması zorunludur.

**Tanım 28.10** (Topolojik Sıralama)**.** *Bir yönlü $G = (V, E)$ grafında, her $(u, v) \in E$ kenarı için $u$'nun $v$'den önce yer aldığı doğrusal düğüm dizilimine* ***topolojik sıralama*** *(topological sort) denir.*

**Teorem 28.11.** *Bir yönlü grafın en az bir topolojik sıralamaya sahip olması için gerek ve yeter koşul grafın* ***döngüsüz (DAG)*** *olmasıdır.*

*Kanıt.* Grafta bir döngü ($v_1 \to v_2 \to \cdots \to v_k \to v_1$) bulunuyorsa tanım uyarınca sıralamada $v_1$'in $v_2$'den, $v_2$'nin $v_3$'ten ... ve $v_k$'nin $v_1$'den önce gelmesi gerekirdi. Bir eleman kendisinden önce gelemeyeceğinden döngülü bir grafta topolojik sıralama yapılamaz.

Tersine, döngüsüz her sonlu grafta içeri derecesi 0 olan en az bir düğüm vardır (aksi durumda geriye doğru sonsuz bir yol veya döngü oluşurdu). Bu düğüm en başa yerleştirilip graftan çıkarılarak tümevarım yoluyla geçerli bir sıralama kurulabilir. □

### 28.4.2 Kahn Algoritması (İçeri Derece Sezgisi)

Kahn algoritması, Bölüm 27'de ele aldığımız greedy yaklaşımı içeri derecelere uygular:

1. Grafın her düğümü için içeri dereceyi (`in_degree`) belirle.
2. İçeri derecesi 0 olan (hiçbir önkoşulu bulunmayan) tüm düğümleri bir kuyruğa ekle.
3. Kuyruktan bir $u$ düğümü çıkar ve sıralama listesine yaz.

4. $u$'dan çıkan tüm kenarları kaldır; yani her komşu $v$ için in_degree[$v$] değerini 1 azalt.

5. Bir $v$ düğümünün içeri derecesi 0'a düşerse önkoşulları tamamlanmış demektir; bu düğümü de kuyruğa al.

6. Kuyruk boşalana kadar adımları tekrarla.

Sıralama listesindeki toplam eleman sayısı grafın düğüm sayısı $n$'den küçük kalırsa grafın döngü içerdiği anlaşılır.

```
std::vector<int> topolojik_sira;
std::queue<int> q;

for (int i = 0; i < n; i++) {
    if (in_degree[i] == 0) q.push(i);
}

while (!q.empty()) {
    int u = q.front();
    q.pop();
    topolojik_sira.push_back(u);

    for (int v : adj[u]) {
        in_degree[v]--;
        if (in_degree[v] == 0) {
            q.push(v);
        }
    }
}

if (topolojik_sira.size() < n) {
    // Graf dongu iceriyor, siralanis yok!
}
```

**Örnek 28.12.** *$V = \{0, 1, 2, 3, 4\}$ ders kümesi üzerinde bağımlılıklar ($u \to v$, $u$ bitmeden $v$ alınamaz) şöyle verilsin:*

$$E = \{(0, 2), (1, 2), (2, 3), (2, 4), (3, 4)\}$$

*Bu graf için geçerli bir topolojik sıralama bulunuz.*

*Çözüm.* Düğümlerin içeri derecelerini belirleyelim:

- in_degree[0] = 0
- in_degree[1] = 0
- in_degree[2] = 2 (0 ve 1'den)

- in_degree[3] = 1 (2'den)
- in_degree[4] = 2 (2 ve 3'ten)

Başlangıçta içeri derecesi 0 olan düğümler 0 ve 1'dir. Kuyruk: $[0, 1]$.

1. 0 çıkarılır. Liste: [0]. $(0, 2)$ kenarı silinir: in_degree[2] $= 2 - 1 = 1$.
2. 1 çıkarılır. Liste: $[0, 1]$. $(1, 2)$ kenarı silinir: in_degree[2] $= 1 - 1 = 0$. 2 kuyruğa girer. Kuyruk: [2].
3. 2 çıkarılır. Liste: $[0, 1, 2]$. $(2, 3)$ ve $(2, 4)$ silinir. in_degree[3] $= 0$, in_degree[4] $= 1$. 3 kuyruğa girer. Kuyruk: [3].
4. 3 çıkarılır. Liste: $[0, 1, 2, 3]$. $(3, 4)$ silinir. in_degree[4] $= 0$. 4 kuyruğa girer. Kuyruk: [4].
5. 4 çıkarılır. Liste: $[0, 1, 2, 3, 4]$.

Geçerli bir topolojik sıralama $0, 1, 2, 3, 4$ olarak bulunur. $(1, 0, 2, 3, 4$ dizilimi de bir diğer geçerli çözümdür; topolojik sıralama tek olmak zorunda değildir.) □

## 28.5 MST (Minimum Spanning Tree): Kruskal ve Prim Sezgisi

### 28.5.1 Motivasyon: Şehirleri En Düşük Maliyetle Birbirine Bağlamak

Bir ülkede $n$ şehir ve bu şehirler arasına döşenebilecek demir yolu hatları bulunmaktadır. Her hattın belirli bir inşa maliyeti vardır. Hedefimiz, tüm şehirlerin birbirine doğrudan veya dolaylı olarak bağlanabildiği, döngü barındırmayan ve **toplam maliyeti minimum** olan bir hat ağı kurmaktır.

Graf teorisinde bu yapı **MST (minimum spanning tree)** olarak adlandırılır.

**Tanım 28.13** (Ağaç ve Spanning Tree)**.** *$n$ düğümlü, bağlı ve döngüsüz bir grafa **ağaç** (tree) denir. Ağaçlar her zaman tam olarak $n - 1$ kenara sahiptir.*

*Bağlı ve yönsüz bir $G = (V, E)$ grafının tüm düğümlerini kapsayan döngüsüz alt grafa $G$'nin bir **spanning tree** denir. Kenar ağırlıkları toplamı en küçük olan spanning tree'ye ise **MST (minimum spanning tree)** denir.*

### 28.5.2 Kesit Özelliği (Cut Property)

MST algoritmalarının temel dayanağı kesit özelliğidir.

**Teorem 28.14** (Kesit Özelliği)**.** *$V$ düğüm kümesini boş olmayan iki ayrık $S$ ve $V \setminus S$ kümesine ayıralım. Bir ucu $S$'de, diğer ucu $V \setminus S$'de olan kenarlara "kesit kenarları" denir. Kesit kenarları arasında ağırlığı kesin biçimde en küçük olan kenar $e^*$, $G$'nin MST'sinde yer almak **zorundadır**. **zorundadır**.*

*Kanıt.* Aksini varsayalım: $e^*$ kenarını içermeyen bir $T$ MST (minimum spanning tree) bulunsun. $e^*$ kenarı $T$'ye eklendiğinde ağaçta bir döngü oluşur. Bu döngü $S$ kümesinden $V \setminus S$ kümesine geçmek zorunda olduğundan, döngü üzerinde kesiti aşan başka bir $e'$ kenarı yer almalıdır.

Şimdi $T$'den $e'$ çıkarılıp yerine $e^*$ eklensin: $T' = T \cup \{e^*\} \setminus \{e'\}$. Yeni yapı tüm düğümleri birbirine bağlayan geçerli bir spanning tree'dir ve toplam maliyeti:

$$w(T') = w(T) - w(e') + w(e^*)$$

olur. $e^*$ kesitteki en küçük ağırlıklı kenar olduğundan $w(e^*) \leq w(e')$ geçerlidir. Eğer $w(e^*) < w(e')$ ise $w(T') < w(T)$ elde edilir ki bu durum $T$'nin minimumluğuyla çelişir. Dolayısıyla en küçük ağırlıklı kenarın seçilmesi daima güvenlidir. □

### 28.5.3 Kruskal Algoritması (Kenar Odaklı Greedy)

Kruskal algoritması, kenarları en ucuzdan en pahalıya doğru sıralayarak Bölüm 27'deki greedy yöntemle ilerler:

1. Grafın tüm kenarlarını ağırlıklarına göre küçükten büyüğe sırala.
2. Sıradaki kenarı incele: Kenar mevcut seçilmiş kenarlarla bir **döngü oluşturmuyorsa** MST'ye dahil et.
3. Toplam $n - 1$ kenar seçildiğinde işlemi sonlandır.

Döngü kontrolünü verimli kılmak için Ayrık Küme Veri Yapısı (*Disjoint Set Union - DSU*) kullanılır. Kenarları sıralamak $O(m \log m)$, DSU işlemleri ise neredeyse doğrusal zaman aldığından Kruskal algoritmasının toplam karmaşıklığı $O(m \log n)$ olur.

### 28.5.4 Prim Algoritması (Düğüm Odaklı Greedy)

Prim algoritması tek bir düğümle başlar ve ağacı adım adım genişletir:

1. Ağaca başlangıç düğümünü ekle (küme $S$).
2. $S$ kümesindeki düğümleri dışarıdaki ($V \setminus S$) düğümlere bağlayan kenarlar arasından en küçük ağırlıklı olanı seç.
3. Bu kenarı MST'ye, ucundaki yeni düğümü de $S$'ye dahil et.
4. Tüm düğümler $S$'ye katılana kadar adımları sürdür.

En küçük ağırlıklı kenarı hızlı seçmek için Bölüm 25'te gördüğümüz priority queue (min-heap) yapısı kullanılır. Bu sayede Prim algoritması $O(m \log n)$ sürede çalışır.

**Örnek 28.15.** *Köşeleri $A, B, C, D$ olan bir grafın kenar ağırlıkları şöyledir:*

$$(A,B):1, \quad (B,D):2, \quad (A,C):3, \quad (B,C):4, \quad (C,D):5$$

*Kruskal algoritması adımlarıyla MST'yi bulunuz.*

*Çözüm.* Kenarları ağırlıklarına göre sıralayalım:

1. $(A,B)$ ağırlık 1
2. $(B,D)$ ağırlık 2
3. $(A,C)$ ağırlık 3
4. $(B,C)$ ağırlık 4
5. $(C,D)$ ağırlık 5

Kenarları sırayla inceleyelim:

- $(A,B)$ seçilir (Ağırlık 1). MST: $\{(A,B)\}$.
- $(B,D)$ seçilir (Ağırlık 2). Döngü oluşmaz. MST: $\{(A,B),(B,D)\}$.
- $(A,C)$ seçilir (Ağırlık 3). Döngü oluşmaz. MST: $\{(A,B),(B,D),(A,C)\}$.

$n-1=4-1=3$ kenar seçilmiş ve tüm düğümler tek bir bileşende birleşmiştir. Algoritma tamamlanır. MST toplam ağırlığı: $1+2+3=6$. □

## 28.6 En Kısa Yol: Dijkstra Algoritması Sezgisi

### 28.6.1 Motivasyon: Ağırlıklı Graflarda En Kısa Rota

Harita uygulamalarını ele alalım. Şehirler arası mesafeler veya yol süreleri birbirinden farklıdır. Bir ilden diğerine giderken en az sayıda kavşaktan geçmek değil, **toplam seyahat süresi minimum** olan güzergâhı bulmak hedeflenir.

BFS her kenarın maliyetini eşit kabul ettiğinden ağırlıklı graflarda doğrudan sonuç vermez. Kenar ağırlıklarının negatif olmadığı durumlarda **Dijkstra algoritması** devreye girer.

### 28.6.2 Gevşetme (Relaxation) İlkesi

Dijkstra algoritmasının temelinde, üçgen eşitsizliğine dayanan *gevşetme* işlemi yer alır.

$u$ düğümüne kadar bilinen en kısa mesafe $dist[u]$ olsun. $u$'dan $v$'ye uzanan kenarın ağırlığı $w(u, v)$ ise ve

$$dist[u] + w(u, v) < dist[v]$$

eşitsizliği sağlanıyorsa, $v$'ye daha kısa bir rota bulunmuş demektir. Mesafe değeri güncellenir:

$$dist[v] \leftarrow dist[u] + w(u, v)$$

### 28.6.3 Dijkstra Algoritmasının İşleyişi

Dijkstra, greedy bir yaklaşımla her adımda henüz mesafesi kesinleşmemiş düğümler arasından başlangıca en yakın olanını işleme alır:

1. Tüm düğümler için $dist[v] = \infty$ yap, kaynak düğüm için $dist[kaynak] = 0$.
2. Henüz ziyaret edilmemiş düğümler arasından $dist[u]$ değeri en küçük olan $u$ düğümünü seç.
3. $u$'yu kesinleşmiş olarak işaretle.
4. $u$'nun tüm komşuları $v$ için gevşetme adımını uygula.
5. Bütün düğümler kesinleşene kadar 2. adıma dön.

En küçük mesafeli düğüme hızla ulaşmak için Bölüm 25'te gördüğümüz priority queue (`std::priority_queue`) kullanılır.

```
#include <queue>
#include <vector>

const int INF = 1e9;
typedef std::pair<int, int> pii; // {mesafe, dugum}
std::vector<pii> adj[MAXN];      // {komsu, agirlik}
int dist[MAXN];

void dijkstra(int kaynak, int n) {
    for (int i = 0; i < n; i++) dist[i] = INF;

    // Kucukten buyuge siralayan min-heap
    std::priority_queue<pii, std::vector<pii>, std::greater<pii>> pq;

    dist[kaynak] = 0;
    pq.push({0, kaynak});
```

```
    while (!pq.empty()) {
        auto [d, u] = pq.top();
        pq.pop();

        // Eger kuyruktan cikan mesafe guncel mesafeden buyukse, bu eski
    bir kayittir
        if (d > dist[u]) continue;

        for (auto edge : adj[u]) {
            int v = edge.first;
            int w = edge.second;

            if (dist[u] + w < dist[v]) {
                dist[v] = dist[u] + w;
                pq.push({dist[v], v});
            }
        }
    }
}
```

### 28.6.4 Neden Negatif Ağırlıklarda Çalışmaz?

Dijkstra algoritmasının doğruluğu, "pozitif kenarlar eklendikçe yol maliyetinin yalnızca artabileceği" kabulüne dayanır.

Bir $u$ düğümü kuyruktan en küçük mesafe değeriyle çekildiği anda, ona başka bir düğüm üzerinden daha kısa bir yoldan ulaşılamaz; çünkü incelenebilecek tüm alternatifler başlangıçtan zaten daha uzaktadır ve pozitif kenarlar mesafeyi yalnızca büyütür.

Graf negatif kenar içerdiğinde bu varsayım bozulur: Uzaktaki bir düğümden çıkan negatif bir kenar, toplam mesafeyi $dist[u]$ değerinin de altına çekebilir. Negatif kenarlı graflarda Bellman-Ford gibi farklı yöntemlere başvurulur; bu yöntemler kitabın kapsamı dışındadır.

**Örnek 28.16.** *Köşeleri* $0, 1, 2, 3$ *olan yönlü ve ağırlıklı bir grafta kenarlar şöyledir:* $(0 \to 1, w{=}4)$, $(0 \to 2, w{=}1)$, $(2 \to 1, w{=}2)$, $(1 \to 3, w{=}1)$, $(2 \to 3, w{=}5)$ *Kaynak düğüm* $0$ *olmak üzere tüm düğümlere olan en kısa mesafeleri adım adım bulunuz.*

*Çözüm.* Başlangıç: $dist = [0, \infty, \infty, \infty]$. Kuyruk: $\{(0, 0)\}$.

1. $(0, 0)$ çekilir. $u = 0$ kesinleşir ($dist[0] = 0$).
   - Komşu 1: $dist[0] + 4 = 4 < \infty \implies dist[1] = 4$. Kuyruğa $(4, 1)$ eklenir.
   - Komşu 2: $dist[0] + 1 = 1 < \infty \implies dist[2] = 1$. Kuyruğa $(1, 2)$ eklenir.

   Kuyruk: $\{(1, 2), (4, 1)\}$.

2. En küçük mesafe olan $(1, 2)$ çekilir. $u = 2$ kesinleşir ($dist[2] = 1$).
   - Komşu 1: $dist[2] + 2 = 3 < dist[1] = 4$ olduğundan gevşetme yapılır: $dist[1] = 3$. Kuyruğa $(3, 1)$ eklenir.
   - Komşu 3: $dist[2] + 5 = 6 < \infty \implies dist[3] = 6$. Kuyruğa $(6, 3)$ eklenir.

   Kuyruk: $\{(3, 1), (4, 1), (6, 3)\}$.
3. En küçük mesafe olan $(3, 1)$ çekilir. $u = 1$ kesinleşir ($dist[1] = 3$).
   - Komşu 3: $dist[1] + 1 = 4 < dist[3] = 6$ olduğundan gevşetme yapılır: $dist[3] = 4$. Kuyruğa $(4, 3)$ eklenir.

   Kuyruk: $\{(4, 1), (4, 3), (6, 3)\}$.
4. $(4, 1)$ çekilir: $4 > dist[1] = 3$ olduğundan işlem yapılmadan atlanır (geçersiz kayıt).
5. $(4, 3)$ çekilir. $u = 3$ kesinleşir ($dist[3] = 4$). Çıkış kenarı yoktur.
6. $(6, 3)$ çekilir: $6 > dist[3] = 4$ olduğundan atlanır.

Sonuç mesafeleri: $dist[0] = 0$, $dist[1] = 3$, $dist[2] = 1$, $dist[3] = 4$ olarak bulunur. □

**Dikkat:** Kuyruktan çekilen $(d, u)$ çifti için `if (d > dist[u]) continue;` kontrolünü unutmak, daha önce güncellenmiş eski kopyaların gereksiz yere işlenmesine ve algoritmanın zaman limitini aşmasına (Time Limit Exceeded) yol açabilir.

## 28.7 Bölüm Özeti ve Algoritma Karşılaştırması

Temel graf algoritmalarının kullanım alanları ve çalışma koşulları aşağıdaki tabloda derlenmiştir:

| **Algoritma** | **Tipik Kullanım Alanı** | **Zaman** | **Kritik Koşul** |
|---|---|---|---|
| **DFS** | Bileşen bulma, döngü tespiti, iki kümelilik | $O(n + m)$ | - |
| **BFS** | Ağırlıksız en kısa yol | $O(n + m)$ | Tüm kenarlar eşit maliyetli |
| **Topolojik Sıra** | Bağımlılık sıralaması, iş planlama | $O(n + m)$ | Graf mutlaka DAG olmalı |
| **Kruskal / Prim** | Minimum spanning tree (MST) | $O(m \log n)$ | Graf bağlı ve yönsüz olmalı |
| **Dijkstra** | Ağırlıklı en kısa yol | $O(m \log n)$ | Negatif kenar **olmamalıdır** |

Graf problemleriyle karşılaşıldığında izlenecek temel adımlar şunlardır:

1. Problemdeki varlıklar düğüm, ilişkiler ise kenar olarak kurulabiliyor mu?
2. Graf yönlü mü, yönsüz mü? Kenarlarda maliyet veya ağırlık tanımlanmış mı?
3. Kenarlar ağırlıksız ise en kısa yol için **BFS**; kenarlar pozitif ağırlıklı ise **Dijkstra** tercih edilmelidir.
4. Bir öncelik sırası oluşturulacaksa **Topolojik Sıralama**, tüm düğümleri en az maliyetle bağlamak isteniyorsa **MST** aranmalıdır.

# Bölüm 29

# C'ye Giriş ve Kontrol Akışı

Bölüm 20'de algoritmik fikirleri sözde kod düzeyinde tasarlamayı ele almıştık. Ancak bu fikirleri donanımın anlayacağı, sınır durumları hatasız işleyen ve zaman kısıtına uyan bir koda dönüştürmek sürecin tamamlayıcı adımıdır. C dili, donanım kaynaklarına doğrudan erişim sağlaması, bellek modelini açık biçimde sunması ve minimum çalışma zamanı (runtime) ek yükü getirmesi nedeniyle TÜBİTAK 1. Aşama sınavının ve yarışma programlamasının temel araçlarındandır.

Bu doğrultuda C programlama dilinin temel yapı taşlarını oluşturan veri tipleri, değişkenlerin bellekteki temsili, standart girdi/çıktı işlemleri ve program akışını belirleyen koşullu dallanma mekanizmaları aşağıda adım adım açıklanmıştır.

## 29.1 Değişkenler, Tipler, printf/scanf

Bir algoritmanın çalışabilmesi için veriyi bellekte saklaması, dönüştürmesi ve girdi/çıktı kanallarıyla etkileşime girmesi gerekir. C dilinde her verinin donanım seviyesinde karşılık gelen bir boyutu, yorumlanma biçimi (tipi) ve bellekte belirli bir adresi bulunur.

### 29.1.1 İlk Program ve Bellek Sezgisi

Temel bir programla başlayarak işletim sistemi ile kod arasındaki çalışma ilişkisini görelim:

```
#include <stdio.h>

int main(void) {
    printf("Tubitak Bilgisayar Olimpiyati\n");
    return 0;
}
```

Bu program derlenip çalıştırıldığında işletim sistemi `main` fonksiyonunu çağırır. `#include <stdio.h>` yönergesi, standart girdi/çıktı (standard input/output)

fonksiyonlarının bildirimlerini içeren başlık dosyasını koda dahil eder. `return 0;` ifadesi ise işletim sistemine programın başarıyla sonlandığını bildiren bir çıkış kodudur (exit status). Değerlendirme sistemlerinde `return 0;` haricinde sıfır olmayan bir değerle çıkmak veya programın çökmesi, *Runtime Error* (Çalışma Zamanı Hatası) olarak kaydedilir.

### 29.1.2 Temel Veri Tipleri ve Temsil Sınırları

Bilgisayar belleği bayt dizilerinden oluşur. Bir değişken tanımlandığında derleyici, bu değişkene tipi kadar baytlık ardışık bellek alanı ayırır.

**Tanım 29.1** (Temel Tam Sayı Tipleri)**.** *C dilinde tam sayı değerlerini saklamak için kullanılan temel tipler ve 64-bit mimarilerdeki yaygın boyutları şunlardır:*

- ***`char`**: 1 bayt (8 bit). Genellikle karakterleri saklar; sayısal olarak* $[-128, 127]$ *veya* $[0, 255]$ *aralığında bir tam sayıdır.*
- ***`int`**: Genellikle 4 bayt (32 bit).* $[-2^{31}, 2^{31}-1]$ *aralığındaki tam sayıları temsil eder. Sayısal karşılığı* $[-2\,147\,483\,648, 2\,147\,483\,647]$ *aralığıdır (*$\approx \pm 2 \cdot 10^9$*).*
- ***`long long`**: En az 8 bayt (64 bit).* $[-2^{63}, 2^{63}-1]$ *aralığındaki tam sayıları saklar (*$\approx \pm 9.22 \cdot 10^{18}$*).*
- ***`unsigned`**: İşaretsiz belirteci, en soldaki işaret bitini de büyüklük hesabına katarak aralığı* $[0, 2^k - 1]$ *yapar (örneğin* ***`unsigned int`*** *için* $[0, 2^{32}-1] \approx [0, 4.29 \cdot 10^9]$*).*

**Tanım 29.2** (Kayan Noktalı Tipler)**.** *Gerçel sayıları IEEE 754 standardına göre temsil eden tiplerdir:*

- ***`float`**: Tek duyarlıklı (single precision), 4 bayt. Yaklaşık 7 anlamlı basamak hassasiyetine sahiptir.*
- ***`double`**: Çift duyarlıklı (double precision), 8 bayt. Yaklaşık 15–17 anlamlı basamak hassasiyetine sahiptir. Hassasiyet kaybı riskini azaltmak adına gerçel sayılarda genellikle* ***`double`*** *tercih edilir.*

**Teorem 29.3** (Tam Sayı Taşması ve Tanımsız Davranış)**.** *C standartlarına göre, işaretli tam sayılarda (`signed int`, `signed long long`) değerin temsil sınırlarını aşması* ***tanımsız davranıştır*** *(undefined behavior, kısaca UB). İşaretsiz tam sayılarda (`unsigned`) ise taşma tanımsız değildir; Bölüm 14'te ele aldığımız modüler aritmetik kurallarına uygun biçimde hesaplama doğal olarak* $2^k$ *modülünde yürür:*

$$a + b \pmod{2^k}$$

*burada* $k$ *değişkenin bit genişliğidir.*

*Açıklama ve İspat Taslağı.* Derleyiciler optimizasyon adımlarında işaretli tam sayı taşmasının hiç gerçekleşmeyeceğini varsayar. Örneğin işaretli bir $x$ için `x + 1 > x` ifadesi doğrudan `1` (doğru) olarak sadeleştirilebilir; oysa $x = 2^{31} - 1$ iken taşma olursa değer $-2^{31}$ olur. Derleyici bu taşma ihtimalini hesaba katmak zorunda olmadığından, kod optimize edilirken beklenmeyen mantıksal hatalar doğabilir. İşaretsiz türlerde ise taşma davranışı standart tarafından $2^k$ modülüyle güvence altına alınmıştır. □

### 29.1.3 Biçimlendirilmiş Girdi ve Çıktı: printf ve scanf

`printf` ve `scanf`, C dilinde giriş ve çıkış işlemlerinin merkezinde yer alır. Her iki fonksiyon da biçim dizgileri (format strings) ve yer tutucular üzerinden çalışır.

**Tanım 29.4** (Sık Kullanılan Biçim Yer Tutucuları)**.** *Temel tiplerin* `printf` *ve* `scanf` *ile okunup yazdırılmasında kullanılan belirteçler:*

- `\{}%d` *veya* `\{}%i`*: İşaretli 32-bit tam sayı (*`int`*).*
- `\{}%u`*: İşaretsiz 32-bit tam sayı (*`unsigned int`*).*
- `\{}%lld`*: İşaretli 64-bit tam sayı (*`long long`*).*
- `\{}%llu`*: İşaretsiz 64-bit tam sayı (*`unsigned long long`*).*
- `\{}%f`*:* `printf` *içinde* `float` *veya* `double`*,* `scanf` *içinde yalnızca* `float`*.*
- `\{}%lf`*:* `scanf` *içinde* `double` *okumak için zorunludur.* `printf` *içinde* `double` *için hem* `\{}%f` *hem de* `\{}%lf` *geçerlidir.*
- `\{}%c`*: Tek bir karakter (*`char`*).*
- `\{}%s`*: Boşluk karakteriyle sonlanan karakter dizisi (string).*

`scanf` fonksiyonuna argüman aktarırken temel kural şudur: Değişkenin doğrudan değeri değil, bellekteki adresi iletilmelidir. Bu işlem adres operatörü olan `&` (ampersand) ile yapılır:

```
int x;
scanf("%d", &x); // x'in bellek adresine girilen sayiyi yazar
```

### 29.1.4 Çözümlü Örnekler

**Örnek 29.5.** *Klavyeden girilen iki adet 32-bit tam sayıyı okuyup toplamlarını ekrana yazdıran, ancak toplam 32-bit sınırını aştığında dahi doğru sonucu veren bir C programı yazınız.*

*Çözüm.* Girdiler $a$ ve $b$ olsun. Problemde $a, b \in [-2\cdot10^9, 2\cdot10^9]$ aralığında değerler alabilir. Bu durumda toplam uç noktalarda yaklaşık $\pm 4 \cdot 10^9$ seviyesine ulaşır ve 32-bit işaretli `int` türünün $[-2^{31}, 2^{31}-1]$ sınırını aşar.

Bu nedenle toplama işlemi yapılmadan önce değerlerden en az biri `long long` türüne dönüştürülmelidir. Aksi halde `int c = a + b;` ifadesinde toplama işlemi önce `int` genişliğinde yapılır, taşma gerçekleşir (UB) ve sonuç hatalı olur.

Program şu şekilde yazılabilir:

```
#include <stdio.h>

int main(void) {
    int a, b;
    if (scanf("%d %d", &a, &b) != 2) {
        return 0;
    }
    long long toplam = (long long)a + b;
    printf("%lld\n", toplam);
    return 0;
}
```

Burada `(long long)a` ifadesi açık tip dönüşümü (type casting) gerçekleştirir. C'nin tür dönüşüm kurallarına göre bir terim `long long` olduğunda, diğer terim $b$ de otomatik olarak bu türe yükseltilir ve toplama işlemi 64-bit genişliğinde taşmasız tamamlanır. □

**Örnek 29.6.** *Aşağıdaki C kod parçası çalıştırıldığında çıktısı ne olur? Nedenini Bölüm 15'te gördüğümüz ikiye tümleyen (two's complement) gösterimi üzerinden açıklayınız.*

```
unsigned int u = 0;
u = u - 1;
printf("%u\n", u);
```

*Çözüm.* İşaretsiz 32-bit tam sayı türündeki bir değişken 0 değerindeyken bellekteki 32 bitin tamamı sıfırdır:

$$(00000000\ 00000000\ 00000000\ 00000000)_2$$

Bu değerden 1 çıkarıldığında işaretsiz taşma kuralı devreye girer. Aritmetik $2^{32}$ modülünde çalıştığından:

$$0 - 1 \equiv -1 \equiv 2^{32} - 1 \pmod{2^{32}}$$

elde edilir. Ortaya çıkan bit dizilimi 32 adet 1 içerir:

$$(11111111\ 11111111\ 11111111\ 11111111)_2 = 2^{32} - 1 = 4\,294\,967\,295$$

Böylece ekrana `4294967295` yazdırılır. □

**Örnek 29.7.** *Aşağıdaki kod parçasının ekrana ne yazdıracağını belirleyiniz:*

```
int a = 7, b = 2;
double d1 = a / b;
double d2 = (double)a / b;
double d3 = (double)(a / b);
printf("%.2f %.2f %.2f\n", d1, d2, d3);
```

*Çözüm.* İfadeler adım adım incelendiğinde:

1. `a / b`: İki değişken de `int` türündedir. C'de tam sayı bölmesinde kesirli kısım doğrudan atılır; dolayısıyla $7/2 = 3$ elde edilir. Bu 3 değeri `double` türündeki `d1` değişkenine aktarılırken 3.0 olur.
2. `(double)a / b`: Tür dönüşümünün işlem önceliği yüksektir. `(double)a` ifadesi 7.0 gerçel sayısını üretir. İşlem 7.0/2 haline gelir; C'nin örtük tür yükseltme kuralı gereği 2 de 2.0 değerine çevrilir ve kayan noktalı bölme ile 3.5 bulunur. `d2` değişkeni 3.5 değerini alır.
3. `(double)(a / b)`: Parantez içindeki `a / b` bölmesi önce yapılır ve tam sayı bölmesiyle 3 sonucunu verir. Ardından bu değer `(double)` ile dönüştürülerek `d3` değişkenine 3.0 olarak atanır.

`\{}%.2f` belirteci virgülden sonra 2 basamak yazdırdığı için çıktı:

3.00 3.50 3.00

olur. □

**Örnek 29.8.** *`scanf` ile ardışık bir tam sayı ve bir karakter okunurken ortaya çıkan boşluk/satır sonu karakteri sorununu ve çözümünü gösteriniz.*

*Çözüm.* Şu kod parçasını ele alalım:

```
int sayi;
char harf;
scanf("%d", &sayi);
scanf("%c", &harf);
```

Kullanıcı konsola `42` yazıp enter tuşuna bastığında, girdi akışına (stdin) sırasıyla `'4'`, `'2'` ve satır sonu belirten `'\{}n'` karakteri eklenir.

- İlk `scanf("\{}%d", &sayi)` çağrısı sayısal karakterleri okuyup `sayi = 42` yapar, ancak `'\{}n'` karakterini akışta bırakır.
- İkinci `scanf("\{}%c", &harf)` çağrısı ise `\{}%c` belirtecinin boşluk veya satır sonu karakterlerini otomatik atlamaması nedeniyle doğrudan akışta bekleyen `'\{}n'` karakterini okur ve değişkene yazar. Program yeni bir karakter beklemeden akışı tamamlar.

Bu durumu önlemek için biçim dizesinde `\{}%c` önüne bir boşluk eklenir:

```
scanf(" %c", &harf);
```

Baştaki bu boşluk karakteri, `scanf`'in girdi akışındaki tüm ardışık boşluk, sekme (`\{}t`) ve yeni satır (`\{}n`) karakterlerini tüketmesini sağlar. □

### 29.1.5 En Sık Yapılan Hatalar

- **`scanf` içinde `&` işaretini unutmak:** `scanf("\{}%d", x);` yazıldığında derleyici uyarı verebilse de kod derlenebilir. Program çalışma anında $x$'in mevcut değerini bir bellek adresi olarak kullanmaya kalkışır ve bellek erişim ihlali (*Segmentation Fault*) ile sonlanır.
- **`double` okurken `\{}%f` kullanmak:** `printf` fonksiyonunda `double` için `\{}%f` yeterlidir; ancak `scanf` ile `double` okunurken mutlaka `\{}%lf` (long float) kullanılmalıdır. `\{}%f` kullanıldığında belleğe yalnızca 4 bayt yazılacağından veri bozulur.
- **İşaretli tam sayı taşmasını gözden kaçırmak:** Örneğin $10^5 \times 10^5 = 10^{10}$ işlemi ara adımda 32-bit sınırını aşar. `long long res = a * b;` yazıldığında $a$ ve $b$ türü `int` ise çarpım önce 32-bit olarak hesaplanır (ve taşar), ardından taşmış değer `res` içine atanır. Güvenli yaklaşım `(long long)a * b` biçiminde dönüşüm yapmaktır.

## 29.2 if-else ve switch

Algoritmalar yalnızca ardışık adımlardan ibaret değildir; verinin durumuna göre farklı hesaplama yollarını seçmek gerekir. C dilinde bu karar mekanizmaları `if-else` blokları ve `switch` çoklu dallanma yapılarıyla kurulur.

### 29.2.1 Mantıksal Doğruluk Sezgisi ve Karşılaştırma Operatörleri

Bölüm 1'de mantıksal önermeleri ve doğruluk değerlerini görmüştük. C dilinin ilk standartlarında yerleşik bir boole (`bool`) tipi yer almıyordu; doğruluk mantığı doğrudan tam sayılar üzerinden tanımlanmıştır:

- 0 değeri **YANLIŞ** (false) kabul edilir.
- Sıfır dışındaki **bütün tam sayılar** (negatif sayılar dahil: $-1, 5, 100$) **DOĞRU** (true) kabul edilir.

Daha sonraki standartlarda `<stdbool.h>` başlığıyla `bool`, `true` (1) ve `false` (0) tanımlanmış olsa da arka plandaki sayısal kural değişmemiştir.

**Tanım 29.9** (İlişkisel ve Mantıksal Operatörler)**.** *C dilindeki temel karşılaştırma araçları:*

- *İlişkisel: `==` (eşit mi), `!=` (farklı mı), `<`, `>`, `<=`, `>=`.*
- *Mantıksal: `&&` (mantıksal VE), `||` (mantıksal VEYA), `!` (mantıksal DEĞİL).*

*Bu operatörler, koşul sağlandığında tamsayı olarak `1`, sağlanmadığında `0` değeri üretir.*

**Teorem 29.10** (Kısa Devre Değerlendirmesi / Short-Circuit Evaluation)**.** *Mantıksal ifadeler soldan sağa doğru incelenir. İfadenin genel sonucu kesinleştiği anda kalan kısım **çalıştırılmaz**:*

1. *`A && B` ifadesinde $A$ yanlış ($0$) ise, $B$'nin sonucuna bakılmaksızın ifadenin tamamı yanlıştır; dolayısıyla $B$ hiç hesaplanmaz.*
2. *`A || B` ifadesinde $A$ doğru (sıfır dışı) ise, $B$'nin sonucuna bakılmaksızın ifade doğrudur; dolayısıyla $B$ hiç hesaplanmaz.*

*Açıklama ve İspat Taslağı.* Kısa devre davranışı, derleyici tarafından örtük koşullu atlama komutlarına dönüştürülür:

```
if (A \{}&\{}& B) ⟺ if (A) \{}{ if (B) \{}{ ... \{}} \{}}
```

Bu mekanizma tanımsız durumlara ve program çökmelerine karşı önemli bir güvencedir. Örneğin `b != 0 && a / b > 2` kontrolünde $b = 0$ olduğunda sağ taraftaki `a / b` bölmesi işletilmez; böylece sıfıra bölme hatası önlenir. □

### 29.2.2 if-else Yapısı ve Asılı Else (Dangling Else) Problemi

Temel koşullu dallanma kalıbı şu şekildedir:

```
if (kosul1) {
    // kosul1 dogruysa calisir
} else if (kosul2) {
    // kosul1 yanlis ve kosul2 dogruysa calisir
} else {
    // hicbiri saglanmazsa calisir
}
```

Süslü parantez kullanılmadığında derleyici yalnızca ilk komutu blok kapsamına alır. Bu durum klasik *Asılı Else* sorununu doğurabilir:

```
if (x > 0)
    if (y > 0)
        printf("1");
else
    printf("2");
```

Buradaki girintileme yanıltıcıdır. C dilinin sözdizimi kuralına göre: **Bir `else`, kendisinden önce gelen ve henüz eşleşmemiş olan en yakın `if` bloğuna bağlanır.** Dolayısıyla kodun gerçek işleyişi şöyledir:

```
if (x > 0) {
    if (y > 0) {
        printf("1");
    } else {
        printf("2");
    }
}
```

$x \leq 0$ durumunda dış koşul sağlanmadığından ekrana herhangi bir çıktı verilmez.

### 29.2.3 switch-case Yapısı ve Düşme (Fall-Through) Özelliği

Bir tamsayı değişkenin çok sayıda sabit değere göre dallanması gerektiğinde `switch` yapısı tercih edilir:

```
switch (ifade) {
    case sabit1:
        // ifade == sabit1 ise buradan itibaren calisir
        break;
    case sabit2:
        // ifade == sabit2 ise buradan itibaren calisir
        break;
    default:
        // hicbir case tutmazsa calisir
        break;
```

```
}
```

**Tanım 29.11** (Düşme (Fall-Through))**.** *Bir* `case` *bloğunun sonuna* `break;` *konulmadığında, program sonraki* `case` *etiketinin koşulunu denetlemeden alt satırları yürütmeye devam eder. Bu duruma düşme (fall-through) adı verilir. Genellikle unutulan bir* `break` *nedeniyle hata kaynağı olsa da kimi zaman bilinçli olarak kullanılır.*

### 29.2.4 Çözümlü Örnekler

**Örnek 29.12.** *Verilen bir yılın artık yıl (leap year) olup olmadığını belirleyen mantıksal ifadeyi ve C kodunu oluşturunuz. (Kural: Bir yıl 4 ile tam bölünüyorsa artık yıldır; ancak 100 ile tam bölünen yıllardan yalnızca 400 ile tam bölünenler artık yıldır).*

*Çözüm.* Girdi yılı $Y$ olsun. Şartları inceleyelim:

- Durum 1: Yıl 400'e bölünüyorsa koşulsuz artık yıldır ($Y \pmod{400} == 0$).
- Durum 2: Yıl 4'e bölünüp aynı zamanda 100'e bölünmüyorsa artık yıldır ($Y \pmod{4} == 0$ ve $Y \pmod{100} \neq 0$).

Bu iki durum mantıksal VEYA (`||`) ile birleştirilebilir:

$$(Y \pmod{400} == 0) \lor (Y \pmod{4} == 0 \land Y \pmod{100} \neq 0)$$

Kısa devre kuralından yararlanmak için, yılların büyük bölümü 4'e bölünmediğinden `yil % 4 == 0` kontrolünü en başa almak kontrol sürecini hızlandırır:

```
#include <stdio.h>

int main(void) {
    int yil;
    if (scanf("%d", &yil) != 1) return 0;

    if ((yil % 4 == 0 && yil % 100 != 0) || (yil % 400 == 0)) {
        printf("%d artik yildir.\n", yil);
    } else {
        printf("%d artik yil degildir.\n", yil);
    }
    return 0;
}
```

Operatör önceliğinde `&&` operatörü `||` operatöründen önce gelse de açık parantez kullanımı kodun doğruluğunu ve okunabilirliğini korur. □

**Örnek 29.13.** *Aşağıdaki C programı çalıştırıldığında ekrana ne basar?*

```c
#include <stdio.h>

int main(void) {
    int x = 2;
    switch (x) {
        case 1:
            printf("A");
        case 2:
            printf("B");
        case 3:
            printf("C");
            break;
        case 4:
            printf("D");
            break;
        default:
            printf("E");
    }
    printf("\n");
    return 0;
}
```

*Çözüm.* Program akışını adım adım takip edelim:

1. `x` değişkeninin değeri 2'dir.
2. `switch` ifadesi doğrudan eşleşen `case 2:` etiketine dallanır.
3. `printf("B");` çalışır ve ekrana `B` yazdırılır.
4. `case 2:` bloğunun sonunda `break;` bulunmadığından akış bir sonraki satıra düşer (fall-through).
5. `case 3:` etiketi denetlenmeksizin altındaki kod işletilir: `printf("C");` çalışır ve ekrana `C` eklenir.
6. `case 3:` bloğunun sonundaki `break;` ifadesi `switch` gövdesini sonlandırır.

Böylece ekrana basılan metin:

BC

olur. □

**Örnek 29.14.** *Aşağıdaki kod parçasının çıktısını belirleyiniz:*

```c
int a = 0, b = 5;
if (a++ && ++b) {
    printf("Dogru: %d %d\n", a, b);
} else {
```

```
    printf("Yanlis: %d %d\n", a, b);
}
```

*Çözüm.* Bu soru kısa devre değerlendirmesi ile artırma operatörlerinin etkileşimini sınar:

1. `a++ && ++b` ifadesinde önce sol taraf değerlendirilir: `a++`.
2. `a++` ifadesinin o anki değeri $a$'nın başlangıçtaki değeri olan 0'dır. İfade değerlendirildikten hemen sonra yan etki (side-effect) olarak $a$ bir artırılarak 1 olur.
3. Mantıksal VE (`&&`) işleminin sol tarafı 0 (yanlış) değerini aldığından kısa devre kuralı devreye girer.
4. Sağ taraftaki `++b` ifadesi hiç çalıştırılmaz; dolayısıyla $b$ değişkeni 5 olarak kalır.
5. Koşul sağlanmadığı için `else` bloğuna geçilir.

Sonuç olarak ekrana basılan çıktı:

Yanlis: 1 5

şeklindedir. □

**Örnek 29.15.** *Üç kenar uzunluğu verilen bir üçgenin geçerli olup olmadığını, geçerliyse eşkenar, ikizkenar veya çeşitkenar olduğunu belirleyen kodu yazınız.*

*Çözüm.* Kenar uzunlukları $a, b, c > 0$ olsun. Üçgen eşitsizliği gereği üçgenin çizilebilmesi için her kenar diğer iki kenarın toplamından kesinlikle küçük olmalıdır:

$$a < b + c \quad \text{ve} \quad b < a + c \quad \text{ve} \quad c < a + b$$

Eğer bu koşullar sağlanıyorsa üçgen geçerlidir ve türü belirlenebilir:

- $a = b = c$ ise eşkenardır.
- Yalnızca iki kenar eşitse ikizkenardır: $(a = b) \vee (b = c) \vee (a = c)$.
- Aksi halde çeşitkenardır.

Program şu biçimde kurulabilir:

```
#include <stdio.h>

int main(void) {
    long long a, b, c;
    if (scanf("%lld %lld %lld", &a, &b, &c) != 3) return 0;
```

```
    // Ucgen esitsizligi kontrolu
    if (a + b <= c || a + c <= b || b + c <= a) {
        printf("Gecersiz Ucgen\n");
    } else if (a == b && b == c) {
        printf("Eskenar\n");
    } else if (a == b || b == c || a == c) {
        printf("Ikizkenar\n");
    } else {
        printf("Cesitkenar\n");
    }

    return 0;
}
```

Kenar uzunluklarının büyük tam sayılar girilebileceği göz önüne alınarak `a + b` toplamında taşmayı önlemek amacıyla değişkenler `long long` tipinde tanımlanmıştır. □

### 29.2.5 En Sık Yapılan Hatalar

- **Eşitlik operatörü `==` yerine atama operatörü `=` yazmak:** `if (x = 5)` yazıldığında $x$ değişkenine 5 atanır. Atanan değer sıfırdan farklı olduğu için koşul her zaman doğru kabul edilir.
- **Matematiksel zincirleme karşılaştırmaları doğrudan C sözdizimine aktarmak:** Matematikteki $1 < x < 10$ ifadesi C dilinde `1 < x < 10` biçiminde yazılırsa sözdizimi hatası vermez ancak mantıksal olarak yanlış çalışır. C bu ifadeyi soldan sağa `(1 < x) < 10` şeklinde hesaplar. `1 < x` alt ifadesi duruma göre `0` veya `1` sonucunu üretir; bu değerler de her halükarda 10'dan küçük olduğundan koşul $x$'ten bağımsız olarak daima doğru çıkar. Doğru yazım `1 < x && x < 10` şeklindedir.
- **`switch` yapısında `break` komutunu unutmak:** İstenmeyen bir fall-through davranışı sonraki tüm durumların sırayla çalışmasına neden olur.
- **Koşul satırının sonuna noktalı virgül koymak:** `if (x > 0);` şeklinde bir noktalı virgül yerleştirildiğinde koşulun gövdesi boş sayılır; ardından gelen blok koşuldan bağımsız biçimde daima işletilir.

# Bölüm 30

# Döngüler ve Fonksiyonlar

Algoritmik düşüncenin kalbinde iki temel ihtiyaç yatar: bir işlemi belirli bir kural dahilinde defalarca tekrarlamak ve karmaşık bir problemi bağımsız, yönetilebilir alt parçalara bölmek. Önceki bölümde gördüğümüz sıralı yürütme ve koşullu dallanma yapıları, bilgisayara yalnızca sonlu ve dallanan yollar çizebilmemizi sağlıyordu. Ancak giriş boyutu $n$ olduğunda, $n$'e bağlı sayıda adımı tek tek elle yazmak imkânsızdır.

Algoritmaların yürütme gücünü katlayan **döngü** mekanizmaları ile program mimarisinin omurgasını oluşturan **fonksiyon** soyutlaması bu bölümün ana konusudur. TÜBİTAK 1. Aşama sınavında sıkça karşımıza çıkan döngü izleme (kod okuma), iterasyon sayısı hesaplama, döngü değişmezleri (loop invariants) kurma ve fonksiyonların bellek modelini (çağrı yığını) anlama becerilerini somut örneklerle ele alacağız.

## 30.1 Döngüler

### 30.1.1 Motivasyon ve Temel Sezgi

Bir matematik probleminde 1'den 100'e kadar olan tam sayıların toplamını bulmamız istendiğinde, Gauss'un yöntemini kullanarak $\frac{100\times101}{2} = 5050$ sonucuna doğrudan ulaşabiliriz. Ancak bilgisayar programlama dünyasında her bağıntının kapalı bir cebirsel formülü bulunmaz. Örneğin, klavyeden girilen bir sayının asal olup olmadığını sınamak, bir tam sayının iki tabanındaki basamaklarını saymak veya Collatz dizisinin kaç adımda 1'e ulaştığını görmek için adımları teker teker yürütmek gerekir.

Döngüler, belirli bir mantıksal koşul doğru (`true`) olduğu sürece bir kod bloğunu art arda çalıştıran kontrol yapılarıdır. Programlamanın sunduğu en temel güç, sonlu satırda kod yazarak dinamik işlem adımları üretebilmektir.

### 30.1.2 `while` Döngüsü

En yalın döngü yapısı `while` döngüsüdür. Çalışma mantığı doğrudan bir koşullu ifadeye dayanır: "Koşul sağlandığı sürece bu bloğu tekrarla."

**Tanım 30.1** (while Döngüsü)**.** *Bir koşul ifadesi ve bir döngü gövdesinden oluşan; koşul her adımın başında doğru (`1` veya `true`) değerini ürettiği müddetçe gövdesini işleten döngü yapısına* ***while döngüsü*** *denir. Koşul yanlış (`0` veya `false`) olduğu anda döngü sonlanır ve akış döngünün ardındaki ilk satırdan devam eder.*

Somut bir problem ele alalım: Pozitif bir $n$ tam sayısının onluk tabandaki basamaklarının toplamını bulmak istiyoruz. Örneğin $n = 3845$ olsun. Son basamağı elde etmek için $n \pmod{10}$ işlemini yaparız (5). Ardından bu basamaktan kurtulmak için tamsayı bölmesiyle $n \leftarrow \lfloor n/10 \rfloor$ yazarız (384). Sayı 0 olana kadar bu adımı tekrarlarsak tüm basamakları tüketmiş oluruz.

```
int n = 3845;
int toplam = 0;

while (n > 0) {
    toplam += n % 10; // Son basamagi ekle
    n /= 10;          // Son basamagi at
}
```

Bu işlem adım adım şöyle ilerler:

1. Başlangıçta: $n = 3845$, toplam $= 0$. Koşul $3845 > 0$ doğru.
2. 1. Adım: toplam $= 0 + 5 = 5$, $n = 384$. Koşul $384 > 0$ doğru.
3. 2. Adım: toplam $= 5 + 4 = 9$, $n = 38$. Koşul $38 > 0$ doğru.
4. 3. Adım: toplam $= 9 + 8 = 17$, $n = 3$. Koşul $3 > 0$ doğru.
5. 4. Adım: toplam $= 17 + 3 = 20$, $n = 0$. Koşul $0 > 0$ **yanlış**.
6. Döngü biter; sonuç 20'dir.

Aynı mantığı C programında tam bir akışla görelim:

```
int main(void) {
    int n = 3845;
    int sum = 0;
    while (n > 0) {
        sum += n % 10;
        n /= 10;
    }
    return 0;
}
```

Burada dikkat edilmesi gereken önemli bir nokta, C dilinde değişken türü tamsayı (`int`) olduğunda `/` operatörünün doğrudan alt taban değerini hesaplamasıdır.

### 30.1.3 `for` Döngüsü

Genellikle tekrar sayısı önceden bilinen ya da belirli bir aralıkta bir sayaç değişkeninin adım adım ilerletildiği durumlarda `for` döngüsü tercih edilir.

C dilinde `for` döngüsü üç temel bileşenden oluşur:

```
for (başlangıç; koşul; güncelleme) { gövde }
```

Bu üçlünün çalışma sırası belirli bir kurala bağlıdır:

1. **Başlangıç (Initialization):** Döngüye girilmeden önce yalnızca bir kez çalıştırılır.
2. **Koşul (Condition):** Her iterasyonun (adımın) başında denetlenir. Yanlış ise döngü anında terk edilir.
3. **Gövde (Body):** Koşul doğru ise gövdedeki kodlar icra edilir.
4. **Güncelleme (Increment/Update):** Gövde tamamlandıktan hemen sonra sayaç güncellenir ve doğrudan 2. adıma (koşul denetimine) dönülür.

```
int toplam = 0;
for (int i = 1; i <= n; i++) {
    toplam += i;
}
```

Sayaç temelli klasik bir toplama döngüsü şu şekildedir:

```
int sum = 0;
for (int i = 1; i <= n; i++) {
    sum += i;
}
```

Döngü sınırlarını belirlerken son değerin aralığa dahil olup olmadığı dikkatle izlenmelidir.

### 30.1.4 `do-while` Döngüsü

Klasik `while` ve `for` döngülerinde koşul gövdeden *önce* denetlenir; bu nedenle koşul en başta sağlanmıyorsa gövde hiç çalışmayabilir. Ancak bazı algoritmik durumlarda gövdenin **en az bir kez** çalışması gerekir.

**Tanım 30.2** (do-while Döngüsü)**.** *Koşul denetimini gövdenin sonunda gerçekleştiren döngü türüdür. C dilinde sözdizimi:*

```
do {
    // Govde (en az bir kez calisir)
} while (kosul);
```

*şeklindedir.*

Örneğin, kullanıcının girdiği pozitif tam sayının basamak sayısını hesaplarken $n = 0$ özel durumunu ele alalım. Eğer standart `while (n > 0)` kurarsak, $n = 0$ için döngüye hiç girilmez ve basamak sayısı hatalı şekilde 0 bulunur. Oysa `do-while` ile yazılırsa:

```
int n = 0;
int basamak_sayisi = 0;
do {
    basamak_sayisi++;
    n /= 10;
} while (n > 0);
```

Kod bloğu bir kez çalışır, `basamak_sayisi` 1 olur, $n$ yine 0 kalır ve koşul $0 > 0$ yanlış olduğu için döngü sonlanır. Sayı 0 dahi olsa doğru basamak sayısı olan 1'e ulaşılır.

### 30.1.5 Akış Kontrol İfadeleri: `break` ve `continue`

Döngünün normal mantıksal akışını değiştirmek amacıyla iki anahtar kelime kullanılır:

- **`break`:** İçinde bulunduğu en yakın döngüyü anında sonlandırır. Program akışı döngü bloğunun hemen sonrasına atlar.
- **`continue`:** Döngü gövdesindeki geri kalan komutları atlayarak doğrudan bir sonraki iterasyona geçer. `for` döngüsünde bu, güncelleme adımının çalıştırılması anlamına gelir.

**Örnek 30.3.** *Aşağıdaki C kod parçası çalıştırıldığında ekrana ne yazdırılır?*

```
int toplam = 0;
for (int i = 1; i <= 10; i++) {
    if (i % 3 == 0) {
        continue;
    }
    if (i > 7) {
        break;
    }
    toplam += i;
}
printf("%d\n", toplam);
```

*Çözüm.* Adım adım sayaç değerlerini izleyelim:

- $i = 1$: $1 \pmod 3 \neq 0$, $1 \not> 7 \implies$ toplam $= 0 + 1 = 1$.

- $i = 2$: $2 \pmod 3 \neq 0$, $2 \not> 7 \implies$ toplam $= 1 + 2 = 3$.
- $i = 3$: $3 \pmod 3 == 0 \implies$ `continue` çalışır, döngünün sonuna atlar, toplama eklenmez.
- $i = 4$: $4 \pmod 3 \neq 0$, $4 \not> 7 \implies$ toplam $= 3 + 4 = 7$.
- $i = 5$: $5 \pmod 3 \neq 0$, $5 \not> 7 \implies$ toplam $= 7 + 5 = 12$.
- $i = 6$: $6 \pmod 3 == 0 \implies$ `continue` çalışır, atlanır.
- $i = 7$: $7 \pmod 3 \neq 0$, $7 \not> 7 \implies$ toplam $= 12 + 7 = 19$.
- $i = 8$: $8 \pmod 3 \neq 0$, fakat $8 > 7$ koşulu sağlanır. `break` çalışır ve döngü tamamen terk edilir.

Sonuç olarak ekrana 19 yazdırılır. □

### 30.1.6 Döngü Değişmezi (Loop Invariant)

Bir döngünün hedeflenen işi doğru yaptığını matematiksel kesinlikle göstermek için döngü değişmezlerinden yararlanırız. Bilgisayar biliminde döngü doğruluğu, Bölüm 2'de gördüğümüz matematiksel tümevarımın ve Bölüm 18'de incelediğimiz invaryant kavramının programlama dünyasındaki izdüşümüdür.

**Tanım 30.4** (Döngü Değişmezi)**.** *Bir döngünün her iterasyonundan önce ve sonra doğru kalmayı sürdüren mantıksal önermeye* ***döngü değişmezi (loop invariant)*** *denir.*

Bir döngü değişmezinin geçerliliği üç adımda ispatlanır:

1. **Başlangıç (Initialization):** Döngünün ilk iterasyonundan önce önerme doğrudur (tümevarımın taban adımı).
2. **Sürdürme (Maintenance):** Döngünün herhangi bir iterasyonundan önce önerme doğruysa, iterasyonun gövdesi icra edildikten sonra da doğru kalır (tümevarım adımı).
3. **Sonlanma (Termination):** Döngü sona erdiğinde, değişmezin doğruluğu ve döngünün sonlanma koşulu bir araya gelerek algoritmanın hedeflenen sonucu ürettiğini gösterir.

**Örnek 30.5.** *Dizi toplamını hesaplayan bir algoritmanın doğruluğunu döngü değişmezi ile gösteriniz:*

```
int s = 0;
for (int i = 0; i < n; i++) {
    s += A[i];
}
```

*Çözüm.* Değişmez önermemiz şu olsun: "$k$'ıncı iterasyon başlamadan önce, $s = \sum_{j=0}^{k-1} A[j]$ eşitliği geçerlidir."

- **Başlangıç:** $i = 0$ iken henüz döngüye girilmemiştir. Tanım gereği $s = 0$'dır ve boş toplam 0'a eşittir. Değişmez doğrudur.
- **Sürdürme:** $i = k$ adımı başlamadan önce değişmezin doğru olduğunu, yani $s = \sum_{j=0}^{k-1} A[j]$ olduğunu varsayalım. Gövde çalıştırıldığında $s$ değerine $A[k]$ eklenir:
$$s_{\text{yeni}} = s + A[k] = \left(\sum_{j=0}^{k-1} A[j]\right) + A[k] = \sum_{j=0}^{k} A[j]$$
Güncelleme adımında $i$ değeri $k+1$ yapılır. Böylece yeni iterasyon başlamadan önce $s = \sum_{j=0}^{(k+1)-1} A[j]$ şartı sağlanmış olur. Değişmez korunur.
- **Sonlanma:** Döngü $i = n$ olduğunda durur. Değişmezimiz bu anda $s = \sum_{j=0}^{n-1} A[j]$ olduğunu söyler. Bu da tam olarak $A[0], A[1], \ldots, A[n-1]$ elemanlarının toplamıdır. Algoritma doğru biçimde sonlanır.

□

### 30.1.7 İç İçe Döngüler (Nested Loops) ve Adım Sayısı Analizi

TÜBİTAK 1. Aşama sınavında sık rastlanan soru tiplerinden biri, iç içe geçmiş döngülerin toplam kaç kez çalıştığını (veya ekrana kaç kez çıktı verdiğini) hesaplamaktır.

Bir döngünün gövdesinde başka bir döngü yer aldığında, dıştaki döngünün *her bir* adımında içteki döngü baştan sona tekrar çalışır. Eğer dış döngü $N$ kez, iç döngü her seferinde $M$ kez dönüyorsa toplam adım sayısı $N \times M$ olur. Ancak çoğu algoritmada iç döngünün sınırları dış döngünün değişkenine bağlıdır.

**Örnek 30.6.** *Aşağıdaki kod parçasında* `sayac++` *satırı toplam kaç kez çalıştırılır?*

```
int sayac = 0;
for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= i; j++) {
        sayac++;
    }
}
```

*Çözüm.* Dış döngüdeki $i$ değişkeninin aldığı her değer için iç döngünün kaç kez çalıştığını listeleyelim:

- $i = 1$ için: $j \in \{1\} \implies$ 1 kez,
- $i = 2$ için: $j \in \{1, 2\} \implies$ 2 kez,

- $i = 3$ için: $j \in \{1, 2, 3\} \implies 3$ kez,
- $\dots$
- $i = n$ için: $j \in \{1, 2, \dots, n\} \implies n$ kez.

Toplam adım sayısı, ardışık sayıların toplamıdır:

$$T(n) = \sum_{i=1}^{n} i = 1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$$

Bu toplam $O(n^2)$ karmaşıklığına karşılık gelir. □

**Örnek 30.7.** *Aşağıdaki kod parçasında* `printf` *fonksiyonu kaç kez çağrılır?*

```
for (int i = 1; i <= n; i *= 2) {
    for (int j = 1; j <= n; j++) {
        printf("*");
    }
}
```

*Çözüm.* Dış döngünün artış biçimine dikkat edelim: $i$ değişkeni birer birer değil, ikiye katlanarak artmaktadır: $1, 2, 4, 8, 16, \dots, 2^k \leq n$.

Burada dış döngü tam olarak $\lfloor \log_2 n \rfloor + 1$ kez döner. İç döngü ise $i$'den bağımsız olarak her seferinde $n$ kez çalışır. Dolayısıyla toplam çağrı sayısı:

$$T(n) = n \cdot (\lfloor \log_2 n \rfloor + 1)$$

olur. Bu işlem $O(n \log n)$ karmaşıklığındadır. □

### 30.1.8 Çözümlü Örnekler

**Örnek 30.8.** *Aşağıdaki kod bloğu yürütüldükten sonra $x$ değişkeninin son değeri ne olur?*

```
int x = 0;
for (int i = 1; i <= 4; i++) {
    for (int j = i; j <= 4; j++) {
        for (int k = 1; k <= j; k++) {
            x++;
        }
    }
}
```

*Çözüm.* Üç adet iç içe döngü bulunmaktadır ve en içteki işlem $x$'i 1 artırmaktadır. Dolayısıyla $x$'in son değeri, aşağıdaki koşulları sağlayan $(i, j, k)$ tamsayı üçlülerinin sayısına eşittir:

$$1 \leq i \leq 4, \quad i \leq j \leq 4, \quad 1 \leq k \leq j$$

Sabit bir $j$ değeri için $i$'nin alabileceği değerler $1 \leq i \leq j$ aralığındadır (çünkü $i \leq j$ şartı geçerlidir). Yani belirli bir $j$ için tam $j$ tane geçerli $i$ değeri bulunur.

Aynı zamanda $k$ değişkeni de 1'den $j$'ye kadar ilerlemektedir; dolayısıyla $k$ da $j$ farklı değer alabilir.

Buna göre her bir $j \in \{1, 2, 3, 4\}$ değeri için oluşturulabilecek $(i, k)$ çiftlerinin sayısı $j \times j = j^2$ olur. Toplam adım sayısı:

$$x = \sum_{j=1}^{4} j^2 = 1^2 + 2^2 + 3^2 + 4^2 = 1 + 4 + 9 + 16 = 30$$

olarak hesaplanır. □

**Örnek 30.9.** *Pozitif bir $n$ tam sayısının asal olup olmadığını $O(\sqrt{n})$ karmaşıklıkta sınayan algoritmayı tasarlayınız ve sonlanma garantisini gösteriniz.*

*Çözüm.* Bölüm 12'de gördüğümüz üzere, bir $n$ sayısı bileşik ise $n = a \cdot b$ şeklinde yazılabilir. Eğer hem $a > \sqrt{n}$ hem de $b > \sqrt{n}$ olsaydı, çarpımları $a \cdot b > \sqrt{n} \cdot \sqrt{n} = n$ çelişkisini doğururdu. O halde çarpanlardan en az biri $\leq \sqrt{n}$ olmak zorundadır. Bu nedenle 2'den başlayarak $\lfloor \sqrt{n} \rfloor$'ye kadar olan sayıları denemek yeterlidir.

```
int asal_mi = (n >= 2); // 0 ve 1 asal degildir

for (int d = 2; d * d <= n; d++) {
    if (n % d == 0) {
        asal_mi = 0; // Bolen bulundu, asal degil
        break;       // Bosuna devam etme, erken cik
    }
}
```

Burada döngü koşulu olarak $d \leq \sqrt{n}$ yerine $d * d \leq n$ yazılmıştır. Bu tercih, karekök fonksiyonundaki kayan noktalı sayı (floating point) yuvarlama hatalarından ve ek fonksiyon çağrısı maliyetinden kaçınmak için standart bir yaklaşımdır. Eğer $n$ bir bölen içeriyorsa `break` ile döngüden derhal çıkılır; bölen yoksa $d$ sayısı $\lfloor \sqrt{n} \rfloor + 1$'e ulaştığında koşul bozulur ve döngü sonlanır. □

### 30.1.9 En Sık Yapılan Hatalar

1. **Bir Eksiği / Bir Fazlası Hatası (Off-by-One Error):** Döngü sınırlarında $<$ ile $\leq$ ayrımını yanlış yapmak. Örneğin $n$ elemanlı bir diziyi gezerken `for (int i = 0; i <= n; i++)` yazılırsa, bellek sınırlarının dışına çıkılarak $A[n]$ elemanına erişilmeye çalışılır (segmentation fault veya tanımsız bellek okuma).

2. **Sonsuz Döngü (Infinite Loop):** Döngü koşulunu etkileyen değişkenin gövdede güncellenmesinin unutulması:

```c
int i = 1;
while (i <= 10) {
    printf("%d\n", i);
    // i++ yazilmasi unutulursa sonsuz dongu olusur!
}
```

3. **Döngü İçinde Döngü Değişkenini Bozmak:** İç içe döngülerde yanlışlıkla her iki döngüde de aynı sayacı kullanmak:

```c
for (int i = 0; i < n; i++) {
    for (int i = 0; i < m; i++) { // HATA: Dis dongunun i'si ezildi!
        // ...
    }
}
```

## 30.2 Fonksiyonlar

### 30.2.1 Motivasyon ve Modülerlik İlkesi

Uzun bir algoritmayı tek bir blok halinde (`main` içinde) yazmak üç temel soruna yol açar:

1. **Kod Tekrarı:** Aynı matematiksel hesabı (örneğin iki sayının EBOB'unu bulma veya kombinasyon hesaplama) programın birkaç farklı yerinde yapmamız gerektiğinde, kodu kopyalayıp yapıştırmak hata olasılığını artırır.

2. **Hata Ayıklama Güçlüğü:** Bir değişkenin değerinin nerede bozulduğunu takip etmek zorlaşır.

3. **Zihinsel Yük:** Büyük problemleri küçük, bağımsız ve tek bir işi yerine getiren alt birimlere ayırmak programın anlaşılırlığını sağlar.

Fonksiyonlar, belirli bir görevi yerine getiren, parametre alabilen ve bir sonuç üretebilen bağımsız alt programlardır.

### 30.2.2 Fonksiyon Anatomisi ve Çağrı Yığını (Call Stack)

C dilinde bir fonksiyon şu genel şablona sahiptir:

```
donus_turu fonksiyon_adi(parametre_listesi) { govde }
```

Bir fonksiyon çağrıldığında bilgisayar belleğindeki işleyişi kavramak, değişken kapsamı (scope) sorularını çözmenin temelidir.

**Tanım 30.10** (Çağrı Yığını ve Yığın Çerçevesi)**.** *Program yürütülürken çağrılan her fonksiyon için belleğin* ***yığın (stack)*** *bölgesinde bir* ***yığın çerçevesi (stack frame)*** *açılır. Bu çerçevede:*

- *Fonksiyona aktarılan parametreler,*
- *Fonksiyonun içinde tanımlanan yerel değişkenler (local variables),*
- *Fonksiyon işini bitirdiğinde geri döneceği adres (return address)*

*saklanır. Fonksiyon tamamlandığında (`return`), açılan yığın çerçevesi bellekten silinir (**pop** edilir) ve kontrol çağıran fonksiyona geri döner.*

Yerel değişkenlerin ömrü (lifetime) ait oldukları fonksiyonun çalışmasıyla sınırlıdır; kapsamları (scope) ise yalnızca o fonksiyon bloğudur.

### 30.2.3 Parametre Aktarım Mekanizmaları

Programlama dillerinde argümanların fonksiyonlara nasıl aktarıldığı, değişkenlerin değerlerinin kalıcı olarak değişip değişmeyeceğini belirler.

#### 1. Değer ile Aktarım (Pass-by-Value)

C dilinde temel türler fonksiyona kopyalanarak aktarılır. Fonksiyon parametre olarak orijinal değişkenin kendisini değil, o anki değerinin bir **kopyasını** alır. Fonksiyon içinde bu parametre üzerinde yapılan hiçbir değişiklik çağıran yerdeki orijinal değişkeni etkilemez.

**Örnek 30.11.** *Aşağıdaki C programı çalıştırıldığında çıktısı ne olur?*

```c
void arttir(int a) {
    a = a + 5;
}

int main() {
    int x = 10;
    arttir(x);
    printf("%d\n", x);
    return 0;
}
```

*Çözüm.* `main` fonksiyonunda $x$ değişkeni için bellekte bir yer ayrılır ve içine 10 yazılır. `arttir(x)` çağrıldığında, `arttir` fonksiyonunun yığın çerçevesinde yeni bir $a$ değişkeni oluşturulur ve $x$'in değeri olan 10 bu değişkene kopyalanır.

Fonksiyon içinde $a$ değişkeni 15 yapılır; ancak bu işlem yalnızca kopyayı değiştirmiştir. Fonksiyon tamamlandığında `arttir`'ın yığın çerçevesi kaldırılır. Kontrol `main`'e döndüğünde $x$ değişkeni hâlâ 10'dur. Dolayısıyla ekrana 10 yazdırılır. □

### 2. İşaretçi ile Aktarım (Pass-by-Pointer)

Eğer bir fonksiyonun çağıran taraftaki bir değişkeni doğrudan güncellemesini istiyorsak, fonksiyon parametresi olarak değişkenin değerini değil, bellekteki **adresini** göndermeliyiz. Bu mekanizmaya C dilinde işaretçiler (pointers) aracılığıyla ulaşılır.

**Örnek 30.12** (İki Değişkenin Değerini Takas Etme – Swap)**.** *İki tamsayı değişkeninin değerlerini birbiriyle değiştiren fonksiyonu doğru kuralım:*

```c
void takas(int *x, int *y) {
    int gecici = *x; // x'in isaret ettigi adresteki degeri al
    *x = *y;         // y'nin adresindeki degeri x'in adresine yaz
    *y = gecici;     // gecici degeri y'nin adresine yaz
}

int main() {
    int a = 3, b = 7;
    takas(&a, &b); // a ve b'nin bellek adreslerini gonderiyoruz
    printf("a = %d, b = %d\n", a, b);
    return 0;
}
```

*Açıklama.* Burada `&a` ifadesi "$a$'nın bellek adresi" demektir. `takas` fonksiyonundaki `int *x` ise "bir tamsayının adresini tutan işaretçi" anlamına gelir. Fonksiyon, adresleri kullanarak doğrudan `main`'in belleğindeki değerlere erişir ve bunları yerinde değiştirir. Çıktı: `a = 7, b = 3` olur. □

### 3. Gösterici ile Dinamik Yapıların Güncellenmesi

C dilinde dinamik yapıları bir fonksiyona aktarırken yapının adresini işaretçi olarak vermek, fonksiyon içindeki değişikliklerin kalıcı olmasını sağlar:

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct {
    int *data;
    size_t size;
    size_t capacity;
} List;

void modify_list(List *list) {
    if (list->size >= list->capacity) {
        size_t new_capacity = list->capacity == 0 ? 4 : list->capacity *
    2;
        int *new_data = (int *)realloc(list->data, new_capacity * sizeof(
    int));
```

```
        if (!new_data) {
            return;
        }
        list->data = new_data;
        list->capacity = new_capacity;
    }
    list->data[list->size++] = 99;
}

int main(void) {
    List a;
    a.size = 3;
    a.capacity = 4;
    a.data = (int *)malloc(a.capacity * sizeof(int));
    if (!a.data) {
        return 1;
    }

    a.data[0] = 1;
    a.data[1] = 2;
    a.data[2] = 3;

    modify_list(&a);

    printf("[");
    for (size_t i = 0; i < a.size; i++) {
        printf("%d%s", a.data[i], (i == a.size - 1) ? "" : ", ");
    }
    printf("]\n");

    free(a.data);
    return 0;
}
```

### 30.2.4 Çözümlü Örnekler

**Örnek 30.13.** *Aşağıdaki C programı çalıştırıldığında çıktısı ne olur?*

```
int f(int *p, int c) {
    c = c - 1;
    if (c == 0) return 1;
    *p = *p + 2;
    return f(p, c) * (*p);
}

int main() {
    int x = 2;
    int sonuc = f(&x, 3);
    printf("x = %d, sonuc = %d\n", x, sonuc);
```

```
    return 0;
}
```

*Çözüm.* Fonksiyona $x$'in adresi (`&x`) ve $c = 3$ değeri aktarılmaktadır. Fonksiyon çağrılarını yığın mantığıyla adım adım izleyelim:

1. **1. Çağrı:** $f(\&x, 3)$
   - $c = 3 - 1 = 2$.
   - $c == 0$ yanlış ($c = 2$).
   - $*p = *p + 2 \implies x = 2 + 2 = 4$ olur.
   - Dönecek değer: $f(\&x, 2) \times (*p)$. ($*p$ çarpımı, içerideki rekürsif çağrı sonuçlandıktan sonra hesaplanacaktır.)
2. **2. Çağrı:** $f(\&x, 2)$
   - $c = 2 - 1 = 1$.
   - $c == 0$ yanlış ($c = 1$).
   - $*p = *p + 2 \implies x = 4 + 2 = 6$ olur.
   - Dönecek değer: $f(\&x, 1) \times (*p)$.
3. **3. Çağrı:** $f(\&x, 1)$
   - $c = 1 - 1 = 0$.
   - $c == 0$ doğru. Doğrudan **1** döndürür. $x$ değeri değişmez, 6 olarak kalır.

Geri dönüş yolunu (yığının çözülmesini) takip edelim:

- 2. Çağrının dönüşü: $f(\&x, 1) \times (*p) = 1 \times 6 = 6$.
- 1. Çağrının dönüşü: $f(\&x, 2) \times (*p) = 6 \times 6 = 36$.

Sonuç olarak `main` içinde $x = 6$ ve `sonuc` $= 36$ olarak bulunur. Ekrana basılan satır: `x = 6, sonuc = 36`. □

**Örnek 30.14** (Öklid Algoritması ile EBOB)**.** *İki pozitif $a$ ve $b$ tam sayısının en büyük ortak bölenini ($\gcd(a, b)$) döngüsel olarak hesaplayan fonksiyonu yazınız ve döngü değişmezini kurunuz.*

*Çözüm.* Öklid algoritmasının matematiksel temelini Bölüm 13'te ele almıştık; burada aynı algoritmayı döngü değişmezi çerçevesinde, C dilinde kodlamaya odaklanacağız. Öklid bağıntısına göre $\gcd(a, b) = \gcd(b, a \pmod b)$ ve $\gcd(a, 0) = a$'dır. Bu indirgemeyi $b$ sıfır olana kadar sürdürürüz:

```
int ebob(int a, int b) {
    while (b != 0) {
        int kalan = a % b;
        a = b;
        b = kalan;
    }
    return a;
}
```

**Döngü Değişmezi:** Döngünün her adımında $\gcd(a_{\text{yeni}}, b_{\text{yeni}}) = \gcd(a_{\text{eski}}, b_{\text{eski}})$ eşitliği korunur. $a \pmod b < b$ olduğundan, her adımda $b$ kesin olarak küçülür; dolayısıyla döngü sonlu sayıda adımda $b = 0$ durumuna ulaşmak zorundadır. $b = 0$ olduğunda $\gcd(a, 0) = a$ eşitliğinden dolayı sonuç $a$ olur. □

**Örnek 30.15** (İkili Üs Alma – Binary Exponentiation)**.** *Verilen bir taban $a$ ve pozitif tamsayı üs $n$ için $a^n$ değerini $O(\log n)$ zamanda döngüsel olarak hesaplayan fonksiyonu tasarlayınız.*

*Çözüm.* Buradaki temel fikir şudur: Eğer $n$ çift ise $a^n = (a^2)^{n/2}$ yazılabilir. Eğer $n$ tek ise $a^n = a \cdot a^{n-1}$ olur. Üssün ikili tabandaki basamaklarını soldan sağa veya sağdan sola tarayarak $O(n)$ yerine $O(\log n)$ adımda sonuca ulaşabiliriz.

```
long long us_al(long long a, long long n) {
    long long sonuc = 1;
    while (n > 0) {
        if (n % 2 == 1) { // n'in son biti 1 mi?
            sonuc *= a;
        }
        a = a * a;       // Tabani karele
        n /= 2;          // Ussu ikiye bol
    }
    return sonuc;
}
```

Değişmezimiz her adım başında $\text{sonuc} \cdot a^n = A^N$ (başlangıçtaki değerler) eşitliğidir. $n = 0$ olduğunda $\text{sonuc} = A^N$ elde edilir. □

### 30.2.5 En Sık Yapılan Hatalar

1. **Dönüş Değeri Olmayan Dal Bırakmak:** Bir fonksiyonda dönüş türü `int` tanımlanmışsa, her olası koşul dalından mutlaka bir `return` ifadesi geçmelidir. Aksi halde tanımsız davranış (undefined behavior) oluşur:

```
int isaret(int n) {
    if (n > 0) return 1;
    else if (n < 0) return -1;
    // HATA: n == 0 durumunda ne donecegi tanimsizdir!
}
```

2. **Yerel Değişkenin Bellek Adresini Döndürmek (Dangling Pointer):** Fonksiyon tamamlandığında yerel değişkenlerin yığın çerçevesi bellekten silinir. Bu nedenle fonksiyon içinde tanımlanan bir yerel değişkenin adresini `return` ile döndürmek ciddi bir hatadır:

```
int* hatali_fonksiyon() {
    int x = 42;
    return &x; // HATA! x fonksiyon bittigi an yok olur.
}
```

3. **Global Değişken Bağımlılığı:** Fonksiyonların girdilerini parametre almak yerine global değişkenlerden okuması ve değiştirmesi, öngörülemeyen yan etkilere (side-effects) yol açar. Fonksiyonlar mümkün mertebe saf (pure) ve dış ortamdan bağımsız tasarlanmalıdır.

# Bölüm 31

# Diziler, Stringler ve Pointer

## 31.1 Diziler

Bilgisayar biliminde en sık karşılaşılan durumlardan biri, aynı türden çok sayıda veriyi bir arada tutmak, bu verilere hızlıca erişmek ve üzerlerinde toplu işlemler gerçekleştirmektir. Örneğin, bir sınıftaki 100 öğrencinin sınav notlarını işlemek istediğimizi düşünelim. Dizi yapısı kullanılmasaydı, programda şu şekilde ayrı ayrı tanımlamalar yapmak gerekirdi:

$$\text{not}_0, \text{not}_1, \text{not}_2, \ldots, \text{not}_{99}$$

Bu yaklaşım iki temel soruna yol açar: İlki, 100 farklı değişken ismi tanımlamanın kodun okunabilirliğini ve bakımını zorlaştırmasıdır. İkincisi ve daha önemlisi, $i$. öğrencinin notuna bir döngü değişkeni aracılığıyla dinamik olarak erişilememesidir; yani $\text{not}_i$ biçiminde bir değişkene çalışma zamanı indisiyle doğrudan ulaşılamaz.

Diziler (arrays), aynı türden verilerin bellekte kesintisiz biçimde ardışık yerleştirildiği ve her bir elemanına bir tam sayı indisi (index) yardımıyla doğrudan, sabit zamanda ($O(1)$) erişilebilen en temel veri yapısıdır.

### 31.1.1 Tek Boyutlu Diziler ve Bellek Modeli

Bir dizinin çalışma mantığını anlamak için bilgisayarın ana belleği (RAM), bayt bayt numaralandırılmış doğrusal bir cetvel gibi düşünülebilir.

**Tanım 31.1** (Tek Boyutlu Dizi)**.** *Aynı veri tipinden $n$ adet elemanın bellekte ardışık hücrelerde saklandığı veri yapısına* ***tek boyutlu dizi*** *denir. $T$ tipindeki elemanlardan oluşan $n$ uzunluğundaki bir dizi $A$ olsun. İndis kümesi $\{0, 1, \ldots, n-1\}$ olmak üzere, $i$. eleman $A[i]$ simgesiyle gösterilir.*

C dilinde ve pek çok modern programlama dilinde diziler **0 tabanlı** (zero-indexed) olarak numaralandırılır. Bu kural keyfi bir tercih olmayıp donanım düzeyindeki adresleme aritmetiğinin doğrudan bir sonucudur.

**Teorem 31.2** (Dizi Elemanının Bellek Adresi)**.** *Eleman boyutu* $S = \text{sizeof}(T)$ *bayt olan bir* $A$ *dizisinin başlangıç adresi (ilk elemanın adresi)* $\text{TabanAdres}(A)$ *olsun. Bu durumda* $i$*. indisteki elemanın bellekteki başlangıç adresi:*

$$\text{Adres}(A[i]) = \text{TabanAdres}(A) + i \times S$$

*formülüyle hesaplanır.*

*Kanıt.* Dizinin elemanları bellekte boşluk bırakılmadan art arda yerleşir. İlk eleman olan $A[0]$ doğrudan başlangıç noktasında bulunduğundan ofset (öteleme) miktarı 0 bayttır:

$$\text{Adres}(A[0]) = \text{TabanAdres}(A) + 0 \times S = \text{TabanAdres}(A)$$

$A[1]$ elemanı, $A[0]$'ın kapladığı $S$ baytlık alanın hemen ardından başlar; dolayısıyla adresi $\text{TabanAdres}(A) + 1 \times S$ olur. Benzer şekilde, $A[i]$ elemanından önce tam $i$ adet eleman yer alır ($A[0], A[1], \ldots, A[i-1]$). Bu $i$ adet elemanın her biri bellekte $S$ bayt yer kapladığından, $A[i]$ elemanının başlangıcına kadar geçilmesi gereken toplam mesafe $i \times S$ bayttır. Buradan:

$$\text{Adres}(A[i]) = \text{TabanAdres}(A) + i \times S$$

elde edilir. □

Bu formül, dizilerin sunduğu $O(1)$ zamanlı **rastgele erişim** (random access) garantisini açıklar. İşlemci, $A[i]$ elemanına erişmek için önceki $i-1$ elemanı tek tek dolaşmaz; yalnızca tek bir çarpma ve tek bir toplama işlemiyle hedeflenen bellek adresine doğrudan ulaşır.

İndislerin 1 yerine 0 ile başlamasının temel gerekçesi de budur: İndisler 1'den başlasaydı, formül $\text{TabanAdres} + (i-1) \times S$ haline gelecek ve işlemci her dizi erişiminde fazladan bir çıkarma işlemi yapmak durumunda kalacaktı.

**Örnek 31.3.** *32-bit bir mimaride bir* `int` *değişkeni 4 bayt yer kaplamaktadır. Bir programda* `int A[10];` *şeklinde bir dizi tanımlanmış ve derleyici bu dizinin başlangıç adresini onluk tabanda* 2000 *olarak belirlemiştir.*

*a)* $A[4]$ *elemanının bellek adresi nedir?*

*b) Bellek adresi* 2028 *olan elemanın indisi kaçtır?*

*Çözüm.* Doğrusal adres formülünü uygularız: $\text{Adres}(A[i]) = \text{TabanAdres} + i \times S$. Burada $\text{TabanAdres} = 2000$ ve $S = 4$ bayttır.

a) $i = 4$ için:

$$\text{Adres}(A[4]) = 2000 + 4 \times 4 = 2000 + 16 = 2016$$

Böylece $A[4]$ elemanı bellekte $2016, 2017, 2018, 2019$ numaralı baytları kaplar.

b) Adresi verilen indisi bulmak için eşitlik $i$ cinsinden çözülür:

$$2028 = 2000 + i \times 4 \implies 4i = 28 \implies i = 7$$

Aranan eleman $A[7]$'dir. □

### 31.1.2 Çok Boyutlu Diziler ve Satır Öncelikli Bellek Serimi

Matematikte matrisler veya iki boyutlu ızgaralar sıklıkla karşımıza çıkar. Örneğin $R$ satır ve $C$ sütundan oluşan bir matris iki boyutlu bir yapı sunar. Bilgisayar belleği ise iki boyutlu bir tablo değil, tek boyutlu doğrusal bir dizilimdir.

C dilinde çok boyutlu diziler **satır öncelikli düzen** (row-major order) ile belleğe serilir. Önce 0. satırın tüm elemanları peş peşe yazılır, ardından 1. satırın elemanları eklenir ve bu işlem son satıra kadar devam eder.

**Teorem 31.4** (İki Boyutlu Dizi Adres Formülü)**.** *$R$ satır ve $C$ sütundan oluşan $M[R][C]$ dizisinde, her bir elemanın boyutu $S$ bayt olsun. $i$. satır ve $j$. sütunda bulunan $M[i][j]$ elemanının bellek adresi:*

$$\text{Adres}(M[i][j]) = \text{TabanAdres}(M) + (i \times C + j) \times S$$

*formülü ile hesaplanır ($0 \leq i < R$ ve $0 \leq j < C$).*

*Kanıt.* $M[i][j]$ elemanına ulaşmak için:

1. Önceki $i$ satırın tamamı atlanmalıdır ($0, 1, \ldots, i-1$. satırlar). Her satırda $C$ adet eleman bulunduğundan, atlanan toplam eleman sayısı $i \times C$'dir.
2. $i$. satıra ulaşıldıktan sonra, bu satır içerisindeki ilk $j$ adet eleman geçilerek $j$. sütuna gelinir.

Böylece $M[0][0]$ elemanından $M[i][j]$ elemanına kadar geride bırakılan toplam eleman sayısı $i \times C + j$ olur. Her eleman $S$ bayt yer tuttuğundan, toplam öteleme miktarı $(i \times C + j) \times S$ bayttır. Taban adrese bu değer eklendiğinde teorem ispatlanmış olur. □

**Örnek 31.5.** *`short` tipi bellekte 2 bayt kaplamaktadır. Bir programda `short M[5][8];` tanımlanmıştır. Taban adresi* 1000 *olduğuna göre:*

*a) $M[3][4]$ elemanının bellek adresini hesaplayınız.*

*b) Bellek adresi* 1054 *olan hücre matrisin hangi indisine ($M[i][j]$) karşılık gelir?*

*Çözüm.* Matris boyutları $R = 5, C = 8$ ve eleman boyutu $S = 2$'dir. Taban adres 1000'dir.

a) $i = 3, j = 4$ değerleri için formülü uygularız:

$$\text{Adres}(M[3][4]) = 1000 + (3 \times 8 + 4) \times 2 = 1000 + (24 + 4) \times 2 = 1000 + 28 \times 2 = 1056$$

b) 1054 adresindeki indisi belirleyelim:

$$1054 = 1000 + (i \times 8 + j) \times 2 \implies (i \times 8 + j) \times 2 = 54 \implies i \times 8 + j = 27$$

$0 \leq j < 8$ kısıtı altında, Bölüm 11'de ele aldığımız bölme algoritması gereğince 27 sayısı 8'e bölünür:

$$27 = 3 \times 8 + 3$$

Buradan bölüm $i = 3$, kalan ise $j = 3$ bulunur. İlgili eleman $M[3][3]$'tür. □

Bu yapı $k$ boyuta genellenebilir. Boyutları $D_0 \times D_1 \times \cdots \times D_{k-1}$ olan bir dizide $(i_0, i_1, \ldots, i_{k-1})$ indisindeki elemana erişirken geride bırakılan eleman sayısı:

$$\sum_{m=0}^{k-1} \left( i_m \prod_{r=m+1}^{k-1} D_r \right)$$

şeklinde hesaplanır.

**Örnek 31.6.** *TÜBİTAK 1. Aşama sınavlarında sıkça karşılaşılan aşağıdaki C kod parçasının ekran çıktısını belirleyiniz:*

```c
#include <stdio.h>

int main() {
    int A[3][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
    int *p = &A[0][0];
    printf("%d ", *(p + 5));
    printf("%d", p[7]);
    return 0;
}
```

*Çözüm.* Dizi bellekte satır öncelikli sırada ardışık dizilmiştir. $A$ matrisi $3 \times 3$ boyutunda olduğundan bellekteki yerleşim sırası şöyledir:

| Ofset: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| Eleman: | $A[0][0]$ | $A[0][1]$ | $A[0][2]$ | $A[1][0]$ | $A[1][1]$ | $A[1][2]$ | $A[2][0]$ | $A[2][1]$ | $A[2][2]$ |
| Değer: | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

$p$ işaretçisi dizinin ilk elemanı olan $A[0][0]$'ın adresini tutmaktadır.

- $*(p+5)$ ifadesi, başlangıç adresinden itibaren 5 eleman ilerideki değeri okur. Ofset tablosuna göre 5. ofset $A[1][2]$ elemanıdır ve değeri 6'dır.
- $p[7]$ ifadesi, işaretçi aritmetiği kuralı gereğince $*(p + 7)$ demektir. 7. ofset $A[2][1]$ elemanıdır ve değeri 8'dir.

Programın çıktısı `6 8` olur. □

### 31.1.3 Dizilerde Sınır Aşımı ve Tanımsız Davranış

C dilinde dizi sınırları derleyici veya çalışma zamanı tarafından otomatik olarak denetlenmez. Dolayısıyla $n$ elemanlı bir dizide $A[n]$ veya $A[-1]$ elemanına erişildiğinde derleyici bir hata üretmeyebilir; program çalışmayı sürdürerek o bellek adresindeki mevcut baytları okur ya da üzerine yazar. Bu durum **tanımsız davranış** (undefined behavior) olarak adlandırılır.

Bellek düzeyinde bakıldığında, $A[n]$ ifadesi $\text{TabanAdres} + n \times S$ adresine erişmeye çalışır. Bu adres dizinin hemen sonrasında belleğe yerleşmiş başka bir değişkene ait olabilir.

**Örnek 31.7.** *Aşağıdaki C kod bloğu çalıştırıldığında kuramsal olarak neden sonsuz döngüye yol açabileceğini bellek yerleşimi açısından açıklayınız:*

```
int i;
int arr[4];
for (i = 0; i <= 4; i++) {
    arr[i] = 0;
}
```

*Çözüm.* Döngü $i = 0, 1, 2, 3, 4$ değerleri üzerinden ilerler. Dizi boyutu ise 4'tür ($arr[0], arr[1], arr[2], arr[3]$ geçerli indislerdir). Döngünün son adımında $i = 4$ iken $arr[4] = 0$ ataması gerçekleştirilir.

Yığın (stack) mimarisinde yerel değişkenler ardışık olarak tahsis edildiğinde, derleyicinin yerleşim düzenine bağlı olarak $i$ değişkeni bellekte $arr[3]$'ün hemen ardına ($arr[4]$'ün denk geldiği adrese) yerleşmiş olabilir. Bu senaryoda:

$$\text{Adres}(arr[4]) = \text{Adres}(i)$$

olur. Döngü $i = 4$ adımına geldiğinde $arr[4] = 0$ komutu çalışır ve $i$'nin saklandığı bellek gözüne 0 yazılır. Böylece sayaç $i = 0$ değerine sıfırlanır ve döngü sonlanamaz. Sınavlarda bu tür sınır aşımları yaygın analiz soruları arasındadır. □

**Tanım 31.8** (En Sık Yapılan Hata: Boyut ve İndis Karışıklığı)**.** *Bir dizi* `int A[N];` *biçiminde bildirildiğinde, kullanılabilecek geçerli indis kümesi* $\{0, 1, 2, \ldots, N-1\}$ *olur.* $A[N]$ *elemanı* ***tanımlı değildir****. Yapılan en temel hatalardan biri,* $N$ *elemanlı bir diziyi* 1*'den* $N$*'ye kadar döngüyle taramaya çalışırken* $A[N]$*'e erişerek bellek hatası (Segmentation Fault) veya beklenmedik hesaplamalar üretmektir.*

## 31.2 Stringler

Algoritmik problemlerde sayıların yanı sıra metinsel verilerle de çalışılır. Python veya Java gibi dillerde metinler bağımsız bir temel veri türüyken, C dilinde yerleşik bir string türü bulunmaz; metinler karakter dizileri (`char[]`) olarak temsil edilir.

Peki bir karakter dizisi bellekte saklandığında metnin nerede bittiği nasıl anlaşılır? Örneğin 20 karakterlik bir diziye 5 harfli `"bilgi"` kelimesi yazıldığında, kalan 15 karakterin metne dahil olmadığı nasıl ayırt edilir?

### 31.2.1 Null Sonlandırıcı (`'\0'`) Kavramı

C dilinde metinlerin uzunluğunu belirlemek amacıyla bir sonlandırıcı işaret (sentinel value) kullanılır. Bu işaret, sayısal ASCII değeri 0 olan **null karakteridir** (`'\0'`).

**Tanım 31.9** (C-Tipi String (Karakter Dizisi))**.** *Bellekte ardışık baytlar halinde saklanan ve son elemanı sayısal değeri sıfır olan null sonlandırıcı (`'\0'`) karakteri ile işaretlenmiş karakter dizilerine* ***C-string*** *denir.*

Burada sayısal değeri 0 olan `'\0'` karakteri ile yazı karakteri olan `'0'` (ASCII değeri 48) birbirine karıştırılmamalıdır.

**Teorem 31.10** (String Bellek Boyutu İlkesi)**.** *$L$ adet karakterden oluşan bir metni C dilinde bir string olarak saklayabilmek için en az $L+1$ elemanlı bir* ***`char`*** *dizisi ayrılmalıdır.*

*Kanıt.* Metnin okunabilmesi için algoritmanın başlangıçtan itibaren ilerleyerek metnin bittiğini belirten `'\0'` karakterine ulaşması gerekir. Metindeki $L$ karakter ilk $L$ indise yerleştirildiğinde $(0, 1, \ldots, L-1)$, $L$. indise null sonlandırıcının konulması zorunludur. Dolayısıyla toplam ayrılan alan en az $L+1$ bayt olmalıdır. □

Örneğin, `char s[] = "Tubitak";` tanımlaması yapıldığında, derleyici metindeki 7 harfe ek olarak dizinin sonuna görünmeyen bir `'\0'` yerleştirir. Bu nedenle `s` dizisi bellekte 8 bayt kaplar:

| 'T' | 'u' | 'b' | 'i' | 't' | 'a' | 'k' | '\0' |
|---|---|---|---|---|---|---|---|
| indis 0 | indis 1 | indis 2 | indis 3 | indis 4 | indis 5 | indis 6 | indis 7 |

**Örnek 31.11.** *Aşağıdaki C programının ekran çıktısını belirleyiniz:*

```c
#include <stdio.h>

int main() {
    char str[] = "OLIMPIYAT";
    str[4] = '\textbackslash 0';
    printf("%s\n", str);
    return 0;
}
```

*Çözüm.* `str` dizisinin başlangıç durumu şöyledir:

$$\text{str} = ['O', 'L', 'I', 'M', 'P', 'I', 'Y', 'A', 'T', '\backslash 0']$$

4. indisteki karakter başlangıçta `'P'` harfidir. `str[4] = '\0';` komutu ile bu konuma null sonlandırıcı atanır:

$$\text{str} = [\text{'O'}, \text{'L'}, \text{'I'}, \text{'M'}, \text{'\textbackslash 0'}, \text{'I'}, \text{'Y'}, \text{'A'}, \text{'T'}, \text{'\textbackslash 0'}]$$

`printf` fonksiyonu `%s` biçim belirteciyle çalışırken diziyi başlangıç adresinden itibaren okur ve karşılaştığı ilk `'\0'` karakterinde yazdırma işlemini sonlandırır. Dizinin devamındaki `'I', 'Y', 'A', 'T'` karakterleri işleme alınmaz.

Dolayısıyla ekrana yalnızca `OLIM` yazdırılır. □

### 31.2.2 Temel String Fonksiyonlarının İç Mantığı ve Karmaşıklığı

Standart `<string.h>` kütüphanesindeki fonksiyonlar, arka planda karakter dizileri üzerinde döngüler çalıştırır. Algoritma analizinde bu işlemlerin zaman karmaşıklığını doğru değerlendirmek önemlidir.

#### Uzunluk Bulma: `strlen`

Bir karakter dizisinin uzunluğu, baştan başlanarak `'\0'` karakterine kadar olan elemanların sayısıdır (null karakter uzunluğa katılmaz).

```
size_t my_strlen(const char *s) {
    size_t len = 0;
    while (s[len] != '\textbackslash 0') {
        len++;
    }
    return len;
}
```

Bu fonksiyon diziyi baştan sona taradığı için zaman karmaşıklığı doğrusal, yani $O(n)$'dir ($n$ karakter sayısıdır).

**Tanım 31.12** (En Sık Yapılan Hata: Döngü Koşulunda `strlen` Kullanımı)**.** *Döngü koşullarında sıklıkla yapılan bir hata şudur:*

```
for (int i = 0; i < strlen(s); i++) {
    // islemler
}
```

*Burada her adımda döngü koşulu kontrol edilirken `strlen(s)` tekrar çağrılır. Fonksiyonun kendisi $O(n)$ sürdüğünden ve döngü $n$ defa tekrar ettiğinden, toplam çalışma karmaşıklığı:*

$$\sum_{i=1}^{n} O(n) = O(n^2)$$

*seviyesine yükselir. $n = 10^5$ düzeyindeki girdilerde bu durum zaman aşımına (TLE) yol açar. Doğru yöntem, uzunluğu döngü öncesinde tek bir değişkende saklamaktır: `int n = strlen(s); for (int i = 0; i < n; i++)`.*

### Karşılaştırma: `strcmp`

İki karakter dizisinin eşitliği C dilinde `s1 == s2` biçiminde denetlenemez. Dizi isimleri bellek adreslerini temsil ettiğinden bu ifade, dizilerin içeriğini değil, bellekte aynı adresi gösterip göstermediklerini sorgular.

İçerik karşılaştırması **sözlükbilimsel** (lexicographical) sırada karakter bazında gerçekleştirilir:

```
int my_strcmp(const char *s1, const char *s2) {
    while (*s1 && (*s1 == *s2)) {
        s1++;
        s2++;
    }
    return *(const unsigned char*)s1 - *(const unsigned char*)s2;
}
```

İki metin birebir aynıysa fonksiyon 0 döndürür. İlk farklı karakterde $s1$'in ASCII değeri $s2$'den küçükse negatif, büyükse pozitif bir fark üretilir.

**Örnek 31.13.** *Aşağıdaki fonksiyon bir karakter dizisini yerinde (in-place) tersine çevirmektedir. Fonksiyonun adımlarını açıklayınız.*

```
void reverse_string(char str[]) {
    int i = 0;
    int j = strlen(str) - 1;
    while (i < j) {
        char temp = str[i];
        str[i] = str[j];
        str[j] = temp;
        i++;
        j--;
    }
}
```

*Çözüm.* Bölüm 22'de incelediğimiz two pointers yaklaşımıyla, baştaki ve sondaki elemanlar karşılıklı olarak takas edilir (swap):

- $i$ göstergesi dizinin başını (0), $j$ göstergesi ise null karakterden hemen önceki son harfi (uzunluk $-$ 1) işaret eder.
- Her adımda `str[i]` ile `str[j]` yer değiştirir.
- $i$ indisi artırılırken ($i++$), $j$ indisi azaltılır ($j--$).
- Göstergeler ortada karşılaştığında ($i \geq j$) işlem biter. Toplamda $\lfloor n/2 \rfloor$ takas yapılır. Zaman karmaşıklığı $O(n)$, ek bellek kullanımı $O(1)$'dir.

□

**Örnek 31.14.** *Aşağıdaki C kodunun ekran çıktısını bulunuz:*

```c
#include <stdio.h>

int main() {
    char s[] = "BAP2024";
    char *p = s;
    while (*p) {
        if (*p >= 'A' && *p <= 'Z')
            *p = *p + ('a' - 'A');
        p++;
    }
    printf("%s", s);
    return 0;
}
```

*Çözüm.* ASCII tablosunda büyük harfler ('A'-'Z') ile küçük harfler ('a'-'z') ardışık bloklar halinde sıralanmıştır. Bir büyük harfin koduna `'a' - 'A'` (yani $97 - 65 = 32$) eklendiğinde doğrudan küçük harf karşılığı elde edilir.

- $p$ işaretçisi dizinin ilk karakteri olan `'B'` ile başlar.
- Karakter büyük harf aralığındaysa küçük harfe dönüştürülür.
- Karakter rakam ise (`'2', '0', '2', '4'`) koşul sağlanmaz ve değer korunur.
- `*p` sonlandırıcı karaktere (`'\0'`) ulaştığında döngü biter.

Dönüşümler:

$$\text{'B'} \to \text{'b'}, \quad \text{'A'} \to \text{'a'}, \quad \text{'P'} \to \text{'p'}$$

Rakamlar aynen kalır. Sonuç olarak ekrana `bap2024` yazdırılır. □

## 31.3 Pointer ve Hafıza

Programlama dilleri değişken kavramı üzerinden belleği soyutlar. Bir değişkene `x = 5` değeri atandığında donanım düzeyinde "x" ismi bulunmaz; sistem yalnızca o değişkenin ayrıldığı bellek adresini tanır.

**Pointer** (işaretçi), doğrudan bir veri değerini değil, **başka bir değişkenin bellek adresini** saklayan bir değişkendir. TÜBİTAK Bilgisayar Olimpiyatı 1. Aşama sınavlarında pointer aritmetiği ve bellek yönetimi önemli bir yer tutar.

### 31.3.1 Adres Operatörü (&) ve Dolaylı Erişim Operatörü (*)

Her değişkenin iki temel niteliği bulunur:

1. **Değeri (r-value):** Hücrede saklanan içerik (örneğin 42).
2. **Adresi (l-value):** Bellekte ayrılan yerin başlangıç konumu (örneğin 0x7ffd).

C dilinde bu iki nitelik iki temel tekli operatörle yönetilir:

- `&` (Adres Operatörü): Değişkenin bellekteki başlangıç adresini üretir.
- `*` (Dolaylı Erişim Operatörü): Verilen adresteki bellek hücresine ulaşarak oradaki değeri okur ya da günceller.

**Tanım 31.15** (Pointer Tanımlama)**.** *$T$ tipindeki bir verinin adresini saklayacak bir pointer değişkeni:*

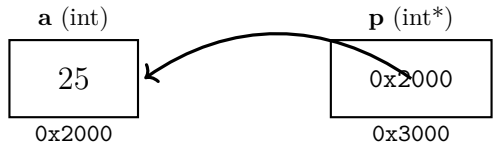
$$T \ *p;$$

*şeklinde bildirilir. Burada $p$ adresin kendisidir; $*p$ ise o adreste saklanan $T$ tipindeki veridir.*

Temel bir örnek üzerinden inceleyelim:

```
int a = 10;
int *p = &a; // p, a'nin bellek adresini tutar
*p = 25;     // p'nin gosterdigi adrese git ve oraya 25 yaz
```

Bu işlem sonucunda $a$'nın değeri 25 olur; çünkü $*p$ ifadesi doğrudan $a$'nın ayrıldığı bellek hücresine erişir.




Şekil 31.1: Pointer ve bellek ilişkisi: $p$ değişkeni $a$'nın adresi olan `0x2000` değerini tutar.

### 31.3.2 Pointer Aritmetiği

Pointer'lar üzerinde toplama ve çıkarma işlemleri yapılabilir. Ancak bu aritmetik, yalın tam sayı işlemlerinden farklı kurallarla işler. Bir pointer'a 1 eklemek, sayısal adres değerini 1 bayt artırmak anlamına **gelmez**.

**Teorem 31.16** (Pointer Aritmetiği Kuralı)**.** *$T$ tipinde bir nesneyi gösteren bir $p$ pointer'ı ve bir $k \in \mathbb{Z}$ tam sayısı verilsin. Bu durumda:*

$$\text{SayısalAdres}(p + k) = \text{SayısalAdres}(p) + k \times \text{sizeof}(T)$$

*olur. Benzer şekilde aynı dizi içerisindeki iki hücreyi gösteren aynı türden $p$ ve $q$ pointer'ları arasındaki fark:*

$$q - p = \frac{\text{SayısalAdres}(q) - \text{SayısalAdres}(p)}{\text{sizeof}(T)}$$

*eleman sayısını verir.*

*Kanıt.* Pointer aritmetiğinin amacı, baytları tek tek saymak yerine ardışık veri blokları arasında eleman bazında adım atmaktır. Bir sonraki elemana geçmek $(p+1)$, geçerli elemanın kapladığı alan kadar ilerlemek demektir. Dolayısıyla adım boyutu, işaret edilen türün bayt boyutu olan sizeof$(T)$ kadardır. $k$ adım ilerlemek $k \times \text{sizeof}(T)$ bayt yer değiştirmeye denk gelir. İki adresin farkı ise aradaki eleman miktarını verir. □

**Örnek 31.17.** *64-bit bir sistemde* `double` *türü 8 bayttır.*

```
double arr[5];
double *ptr1 = &arr[1];
double *ptr2 = &arr[4];
```

*`ptr1`'in sayısal bellek adresi $0x1000$ (onluk sistemde 4096) olduğuna göre:*

*a) `ptr2`'nin sayısal bellek adresi kaçtır?*

*b) `ptr2 - ptr1` işleminin sonucu kaçtır?*

*Çözüm.* a) $ptr1$ adresi $arr[1]$'e, $ptr2$ adresi ise $arr[4]$'e aittir. İki eleman arasında $4 - 1 = 3$ elemanlık fark bulunur. Her `double` elemanı 8 bayt olduğundan:

$$\text{Adres}(ptr2) = 4096 + (3 \times 8) = 4096 + 24 = 4120$$

Onaltılık tabanda: $4120 = 4096 + 24 = 0x1000 + 0x18 = 0x1018$.

b) Pointer farkı doğrudan aradaki eleman adedini üretir:

$$ptr2 - ptr1 = \frac{4120 - 4096}{8} = \frac{24}{8} = 3$$

Bu işlemin sonucu bayt cinsinden değil, eleman sayısı cinsinden bir tam sayıdır. □

### 31.3.3 Dizi ve Pointer İkiliği (Duality)

C dilinde dizi adları ile pointer'lar arasında doğrudan bir bağ bulunur.

**Teorem 31.18** (Dizinin Pointer'a Bozunumu (Array Decay))**.** *Bir ifadenin içerisinde dizi ismi geçtiğinde (`sizeof` ve `&` operatörleri hariç), dizi ismi otomatik olarak **ilk elemanın bellek adresine** dönüşür:*

$$A \equiv \&A[0]$$

Bu özelliğin sonucu olarak dizi erişimi ile pointer aritmetiği birbirine denktir:

$$A[i] \iff *(A + i)$$

Toplama işleminin değişme özelliği ($A + i = i + A$) nedeniyle şu eşitlik geçerlidir:

$$A[i] = *(A + i) = *(i + A) = i[A]$$

Dolayısıyla C dilinde `A[3]` ifadesi ile `3[A]` ifadesi derleyici nezdinde aynı işlemi ifade eder.

**Örnek 31.19.** *Aşağıdaki C kodunun ekran çıktısını bulunuz:*

```c
#include <stdio.h>

int main() {
    int arr[] = {10, 20, 30, 40, 50};
    int *p = arr;

    printf("%d ", *(p + 2));
    printf("%d ", 2[arr]);
    printf("%d\n", *p + 2);
    return 0;
}
```

*Çözüm.* Adımları inceleyelim:

1. `p`, `arr`'nin ilk elemanının adresini tutar (`&arr[0]`). `*(p + 2)` ifadesi, 2 eleman ilerideki değeri okur; bu da `arr[2]` = 30'dur.
2. `2[arr]` ifadesi derleyici tarafından `*(2 + arr)` = `*(arr + 2)` biçiminde çözülür; değer yine 30'dur.
3. `*p + 2` ifadesinde operatör önceliği devreye girer. `*` operatörünün önceliği `+` operatöründen yüksektir. Önce `*p` değerlendirilir (`arr[0]` yani 10), ardından bu değere 2 eklenir: $10 + 2 = 12$.

Program çıktısı: `30 30 12` olur. □

### 31.3.4 Kritik Soru Tipi: Artırma Operatörleri ve Pointerlar

Sınavlarda sıklıkla karşılaşılan kalıplardan biri `*p++`, `(*p)++`, `*++p` ve `++*p` gibi artırma ifadelerinin oluşturduğu yan etkilerdir. Bu ifadeleri doğru yorumlamak için operatör önceliği ve son ek/ön ek ayrımını iyi kavramak gerekir.

**Tanım 31.20** (Operatör Önceliği Tablosu)**.** *Tekli son ek operatörleri (`++`, `--`) en yüksek önceliğe sahiptir ve soldan sağa işletilir. Tekli ön ek operatörleri (`*`, `&`, ön ek `++`, ön ek `--`) aynı öncelik düzeyindedir ve* ***sağdan sola*** *değerlendirilir.*

Bu dört temel durumu karşılaştıralım:

1. `*p++`: Son ek `++` operatörü $p$'ye uygulanır. Son ek olduğundan ifadenin değeri $p$'nin mevcut halidir; işaret edilen değer okunur ve *ardından* $p$ bir sonraki adrese ilerletilir.
2. `(*p)++`: Parantez nedeniyle önce $p$'nin işaret ettiği değer okunur. Ardından bu **sayısal değer 1 artırılır**. Pointer'ın tuttuğu adres değişmez.
3. `*++p`: Ön ek `++` önce $p$'yi bir sonraki adrese taşır. Ardından `*` operatörü yeni adresteki veriyi okur.
4. `++*p`: Sağdan sola öncelik kuralı uyarınca önce `*p` hesaplanır (işaret edilen değer). Ardından bu değer ön ek `++` ile 1 artırılır ve yeni değer işlenir.

**Örnek 31.21.** *Aşağıdaki C kod bloğu çalıştırıldığında dizinin elemanlarının son durumu ve ekrana basılan değerler ne olur?*

```c
#include <stdio.h>

int main() {
    int arr[] = {10, 20, 30};
    int *p = arr;

    int x = *p++;
    int y = (*p)++;
    int z = *++p;

    printf("x=%d, y=%d, z=%d\n", x, y, z);
    printf("arr: %d %d %d\n", arr[0], arr[1], arr[2]);
    return 0;
}
```

*Çözüm.* Bellek durumunu adım adım takip eden bir izleme tablosu (trace table) oluşturalım:

1. Başlangıçta: $arr = \{10, 20, 30\}$. $p$, $arr[0]$ hücresini gösterir ($*p = 10$).
2. `int x = *p++;`

- $p$'nin geçerli adresindeki değer alınır: $x = 10$.
- Yan etki: $p$ bir eleman ilerler, artık $arr[1]$'i gösterir.

3. `int y = (*p)++;`
   - $p$ şu anda $arr[1]$ hücresini göstermektedir; buradaki değer 20'dir.
   - Son ek artırma nedeniyle $y$'ye mevcut değer atanır: $y = 20$.
   - Yan etki: $arr[1]$ hücresindeki değer 1 artarak 21 olur. Pointer adresi değişmez.
4. `int z = *++p;`
   - Önce $p$ bir sonraki adrese kaydırılır: $p$ artık $arr[2]$ hücresindedir.
   - Ardından bu adresteki değer okunur: $arr[2] = 30$, dolayısıyla $z = 30$.

Ekrana yazdırılan sonuçlar:

```
x=10, y=20, z=30
arr: 10 21 30
```

□

### 31.3.5 Pointer Dizileri ile Dizi Pointer'ları Arasındaki Ayrım

Köşeli parantez ile işaretçi bildiriminin birleşimi dikkat gerektiren iki ayrı yapı oluşturur.

**Tanım 31.22** (Pointer Dizisi vs. Dizi Pointer'ı)**.** *Aşağıdaki iki bildirim farklı türleri ifade eder:*

1. *`int *a[5];` $\Longrightarrow$* ***Pointer Dizisi (Array of Pointers):*** *Köşeli parantezin önceliği yıldızdan yüksek olduğundan bu ifade "5 elemanlı bir dizidir, her elemanı `int*` türündendir" şeklinde yorumlanır.*
2. *`int (*b)[5];` $\Longrightarrow$* ***Dizi Pointer'ı (Pointer to an Array):*** *Parantez önceliği belirler. Bu ifade "5 elemanlı bir `int` dizisini gösteren tek bir pointer'dır" anlamına gelir.*

Aralarındaki aritmetik fark belirleyicidir:

- $a$ bir pointer dizisi olduğunda, $a + 1$ ifadesi bir sonraki pointer adresine (sizeof(int*) kadar) ilerler (genellikle 4 veya 8 bayt).
- $b$ ise 5 elemanlı bir diziye işaret ettiğinden, $b + 1$ ifadesi tam bir dizi boyutu kadar, yani $5 \times \text{sizeof(int)} = 20$ bayt birden ilerler.

**Örnek 31.23.** *Aşağıdaki program parçasının çıktısını belirleyiniz:*

```c
#include <stdio.h>

int main() {
    int matris[3][4] = {
        {1,  2,  3,  4},
        {5,  6,  7,  8},
        {9, 10, 11, 12}
    };
    int (*ptr)[4] = matris;

    printf("%d ", **ptr);
    printf("%d ", *(*(ptr + 1) + 2));
    printf("%d\n", *(*ptr + 5));
    return 0;
}
```

*Çözüm.* `ptr`, 4 elemanlı bir tam sayı dizisini gösteren bir pointer'dır. Başlangıçta 0. satırı (`matris[0]`) işaret eder.

1. `**ptr`: `*ptr` ifadesi 0. satırın başlangıç adresini verir (`&matris[0][0]`). Baştaki `*` ise o adresteki veriyi okur. Değer 1'dir.
2. `*(*(ptr + 1) + 2)`:
   - `ptr + 1`, bir satır atlayarak 1. satıra geçer (`matris[1]`).
   - `*(ptr + 1)`, 1. satırın başlangıç adresidir.
   - +2 eklemek, bu satırda 2 sütun sağa kaydırır ($j = 2$).
   - En dıştaki `*` operatörü `matris[1][2]` değerini okur. Bu değer 7'dir.
3. `*(*ptr + 5)`:
   - `*ptr`, 0. satırın başlangıcı olan `&matris[0][0]` adresidir.
   - Bu eleman adresine doğrudan +5 eklenmiştir. Satır genişliği 4 eleman olduğundan 5 adım ilerlemek 0. satırı aşarak 1. satırın 1. indisine ($5 = 1 \times 4 + 1$) ulaşır.
   - Okunan eleman `matris[1][1]`'dir ve değeri 6'dır.

Programın çıktısı: `1 7 6` olur. □

### 31.3.6 Fonksiyonlara Parametre Aktarımı: Değer ile mi, Referans ile mi?

C dilinde fonksiyon argümanları kural olarak **değeriyle çağrılır** (call by value). Bir fonksiyona değişken gönderildiğinde bellekte o değişkenin bir kopyası oluşturulur ve fonksiyona bu kopya iletilir. Fonksiyon içerisindeki değişiklikler asıl değişkeni etkilemez.

**Örnek 31.24** (Klasik Hatalı Swap)**.** *Aşağıdaki takas fonksiyonu iki değişkenin değerini neden değiştiremez?*

```
void swap_hatali(int a, int b) {
    int temp = a;
    a = b;
    b = temp;
}
```

*Açıklama.* Fonksiyon çağrıldığında yığında $a$ ve $b$ isimli yerel değişkenler ayrılır ve argümanların değerleri buraya kopyalanır. Değişiklik yalnızca bu yerel kopyalar üzerinde gerçekleşir. Fonksiyon tamamlandığında yerel değişkenler bellekten silinir; çağıran taraftaki değişkenlerin değerleri aynı kalır. □

Bir fonksiyonun çağıran taraftaki değişkenleri güncelleyebilmesi için o değişkenlerin **bellek adreslerini** parametre olarak alması gerekir:

```
void swap_dogru(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}
```

Çağrı `swap_dogru(&x, &y);` biçiminde yapılır. Fonksiyon doğrudan verilen adreslerdeki verileri takas ettiğinden çağıran taraftaki $x$ ve $y$ kalıcı olarak yer değiştirir.

**Tanım 31.25** (Dizilerin Fonksiyonlara Aktarımı ve Boyut Yitimi)**.** *Bir fonksiyon parametresi olarak dizi tanımlandığında:*

```
void f(int arr[])  // veya void f(int arr[100])
```

*derleyici bu ifadeyi otomatik olarak bir pointer bildirimine dönüştürür:*

```
void f(int *arr)
```

*Bu nedenle fonksiyon içinde* ***sizeof(arr)*** *işlemi yapıldığında dizinin boyutu değil,* ***pointer'ın boyutu*** *(64-bit sistemlerde 8 bayt) elde edilir. Dizinin eleman sayısı fonksiyon içinde doğrudan bilinemeyeceğinden, C'de diziler aktarılırken boyut bilgisi de genellikle ek bir parametre olarak (* ***int n****) fonksiyona verilir.*

**Örnek 31.26.** *Aşağıdaki C kodunun ekran çıktısını adım adım belirleyiniz:*

```
#include <stdio.h>

void gizemli(int *p, int **q) {
    p = *q;
    *p = 40;
    **q = 50;
```

```
}

int main() {
    int a = 10, b = 20;
    int *p1 = &a;
    int *p2 = &b;

    gizemli(p1, &p2);

    printf("a=%d, b=%d, *p1=%d, *p2=%d\n", a, b, *p1, *p2);
    return 0;
}
```

*Çözüm.* İki seviyeli pointer (`int**`) mekanizmasını ve değer aktarımı kurallarını adım adım takip edelim.

**1. Başlangıç Durumu (main yığını):**

- $a = 10$, $b = 20$.
- $p1$, $a$'nın adresini tutar ($\&a$).
- $p2$, $b$'nin adresini tutar ($\&b$).

**2. Fonksiyon Çağrısı: `gizemli(p1, &p2)`** Fonksiyonun yerel parametreleri tahsis edilir:

- Yerel $p$ pointer'ı, $p1$'in bir kopyasıdır ve $a$'nın adresini tutar.
- Yerel $q$ pointer'ı ise $p2$'nin **adresini** saklar ($q = \&p2$). Dolayısıyla $*q$ ifadesi doğrudan `main`'deki $p2$ pointer'ına karşılık gelir.

**3. Fonksiyon Gövdesi:**

- `p = *q;`: $*q$, `main`'deki $p2$'dir ve değeri $\&b$'dir. Yerel $p$ artık $b$'nin adresini tutar. Bu atama yalnızca yerel $p$ üzerinde etkilidir; `main`'deki $p1$ pointer'ı değişmez.
- `*p = 40;`: $p$ şu an $b$'yi gösterdiğinden $b$'nin değeri 40 olur.
- `**q = 50;`: $q$, $p2$'yi; $p2$ ise $b$'yi göstermektedir. Dolayısıyla bu satır $b$'nin değerini 50 yapar.

**4. Fonksiyon Bitişi ve Değerler:**

- $a$ değişkeni değiştirilmemiştir; değeri 10'dur.
- $b$ değişkeni fonksiyon içinde güncellenerek 50 olmuştur.

- $p1$ halen $a$'yı göstermektedir; dolayısıyla $*p1 = 10$'dur.
- $p2$ halen $b$'yi göstermektedir; dolayısıyla $*p2 = 50$'dir.

Programın çıktısı:

```
a=10, b=50, *p1=10, *p2=50
```

□

### 31.3.7 Vahşi ve Sarkan Pointer Tehlikeleri

Bellek güvenliği açısından iki kavrama dikkat edilmelidir:

**Tanım 31.27** (Vahşi Pointer (Wild Pointer))**.** *Tanımlanmış fakat geçerli bir adresle ilklendirilmemiş (uninitialized) pointer'lara* ***vahşi pointer*** *denir:*

```
int *p; // p rastgele bir cop adres tutmaktadir!
*p = 100; // TANIMSIZ DAVRANIS! Cok tehlikeli bir bellek bolgesini
    bozabilir.
```

*Pointer tanımlandığında doğrudan geçerli bir adrese atanmalı veya en azından* `NULL` *ile ilklendirilmelidir.*

**Tanım 31.28** (Sarkan Pointer (Dangling Pointer))**.** *Geçerli bir nesneyi gösterirken o nesnenin ömrü sona erdikten sonra (örneğin yerel değişkenin bulunduğu fonksiyondan çıkıldığında veya tahsis edilen alan serbest bırakıldığında) eski adresi tutmaya devam eden pointer'a* ***sarkan pointer*** *denir:*

```
int* hatali_fonksiyon() {
    int x = 10;
    return &x; // HATA! x fonksiyon bitince yok olur.
}
```

*Bu fonksiyonun döndürdüğü adrese erişmeye çalışmak tanımsız davranışa yol açar.*

# Bölüm 32

# Bitwise Operatörler ve Struct

Bilgisayar biliminde veriyi temsil etmenin ve işlemenin iki temel boyutu vardır: en alt düzeyde, donanımın doğrudan anladığı bitler (0 ve 1) üzerinde yüksek bir hızla çalışabilmek; daha üst düzeyde ise birbiriyle ilişkili verileri mantıksal bir bütün halinde paketleyip soyutlayabilmek.

TÜBİTAK 1. Aşama sınavlarındaki algoritmik problemler sıklıkla bu iki yaklaşımı bir arada gerektirir. Kimi zaman bir kümenin tüm altkümelerini tek bir tamsayının bitleriyle kodlayıp birkaç adımda taramak, kimi zaman da iki boyutlu düzlemdeki noktaları koordinatları ve ek bilgileriyle tek bir nesne olarak takip etmek gerekir. İlk olarak donanımın temel yapıtaşları olan bitwise işlemleri (bitwise operatörler), ardından C dilinde karmaşık veri modelleri kurmamızı sağlayan yapıları (`struct`) ele alacağız.

## 32.1 Bitwise Operatörler

### 32.1.1 Motivasyon ve Sayıların Bellekteki Temsili

Bölüm 29'da temel veri tiplerini ve aritmetik operatörleri gördük. Orada $+$, $-$, $*$, / gibi işlemlerle sayıları onluk tabandaki değerleri üzerinden işliyorduk. Ancak Bölüm 15'te incelediğimiz üzere, modern bir işlemcinin yazmaçlarında (register) saklanan her tamsayı ikili tabanda (2 tabanında) dizilmiş bitlerden oluşur.

Örneğin, 8 bitlik bir işaretsiz tamsayı olarak 43 sayısını ele alalım. $43 = 32 + 8 + 2 + 1 = 2^5 + 2^3 + 2^1 + 2^0$ olduğundan, ikili tabanda $(43)_{10} = (00101011)_2$ şeklinde yazılır. Buradaki her bir basamağa **bit** adı verilir. En sağdaki basamağa en önemsiz bit (least significant bit - LSB, $2^0$ basamağı), en soldaki basamağa ise en önemli bit (most significant bit - MSB) denir.

İşte **bitwise operatörler**, sayıları bir bütün olarak değil, bu bit dizilimlerindeki her bir biti bağımsız biçimde ya da kaydırarak doğrudan yönetmemizi sağlar. Bu operatörler donanım seviyesinde tek bir işlemci çevriminde (cycle) yürütüldüğünden son derece hızlıdır. Dahası, mantıksal durumları ve altkümeleri bitler

aracılığıyla kodlamaya imkan tanır.

### 32.1.2 Temel Bitwise Operatörler

C dilinde 6 temel bitwise operator bulunur:

- `&` (Bitwise VE / AND)
- `|` (Bitwise VEYA / OR)
- `^` (Bitwise ÖZEL VEYA / XOR)
- `~` (Bitwise DEĞİL / NOT - Bir'e tümleyen)
- `<<` (Sola Kaydırma / Left Shift)
- `>>` (Sağa Kaydırma / Right Shift)

**Tanım 32.1** (Bitwise VE, VEYA, XOR ve DEĞİL)**.** *a ve b aynı uzunluktaki iki bit dizisi olsun. Her i. bit için:*

- ***Bitwise VE*** *(a & b)**: Ancak ve ancak hem a'nın hem de b'nin i. biti* 1 *ise sonuç bit* 1 *olur; aksi halde* 0*'dır.*
- ***Bitwise VEYA*** *(a | b)**: a veya b'nin i. bitinden en az biri* 1 *ise sonuç bit* 1 *olur; her ikisi de* 0 *ise* 0*'dır.*
- ***Bitwise XOR*** *(a ^ b)**: a ve b'nin i. bitleri birbirinden farklıysa sonuç bit* 1*, aynıysa* 0 *olur.*
- ***Bitwise DEĞİL*** *(∼ a)**: Tekil (unary) bir operatördür. Sayının her* 0 *bitini* 1*, her* 1 *bitini* 0 *yapar.*

Somut bir örnekle bu işlemleri görelim. $A = 12$ ve $B = 10$ sayılarını 8 bitlik ikili tabanda yazalım:

$$A = 12 = (00001100)_2, \quad B = 10 = (00001010)_2$$

Şimdi operatörleri basamak basamak uygulayalım:

$$\begin{aligned} A \,\&\, B &= (00001100)_2 \,\&\, (00001010)_2 = (00001000)_2 = 8 \\ A \mid B &= (00001100)_2 \mid (00001010)_2 = (00001110)_2 = 14 \\ A \,\hat{}\, B &= (00001100)_2 \,\hat{}\, (00001010)_2 = (00000110)_2 = 6 \\ \sim A &= \sim (00001100)_2 = (11110011)_2 \end{aligned}$$

*x pozitif bir tamsayı ve* $k \geq 0$ *olsun:*

**Tanım 32.2** (Bit Kaydırma Operatörleri: `<<` ve `>>`)**.** ***Sola Kaydırma*** *($x \ll k$): $x$'in tüm bitleri $k$ pozisyon sola kaydırılır. Sağdan boşalan $k$ adet basamağa 0 yazılır. Taşma (overflow) olmadığı sürece bu işlem sayıyı $2^k$ ile çarpmaya denktir:*

$$x \ll k = x \cdot 2^k$$

- ***Sağa Kaydırma*** *($x \gg k$): $x$'in tüm bitleri $k$ pozisyon sağa kaydırılır. Sağdan taşan $k$ adet bit atılır. Sayı işaretsiz (unsigned) ise soldan açılan basamaklara 0 yazılır. Bu işlem sayıyı $2^k$'ye tam bölmeye (aşağı yuvarlamaya) denktir:*

$$x \gg k = \lfloor x/2^k \rfloor$$

Örneğin $x = 5 = (00000101)_2$ olsun:

$$x \ll 1 = (00001010)_2 = 10 \quad (5 \cdot 2^1)$$

$$x \ll 3 = (00101000)_2 = 40 \quad (5 \cdot 2^3)$$

$$x \gg 1 = (00000010)_2 = 2 \quad (\lfloor 5/2 \rfloor)$$

### 32.1.3 Temel Hileler ve Pratik Teknikler

**1. Teklik–Çiftlik (Parite) Testi:** Bölüm 17'de ele aldığımız parite kavramını hatırlarsak, normalde bir $n$ sayısının tek mi çift mi olduğunu anlamak için `n % 2 == 1` kontrolü yaparız. Ancak 2'ye bölme işleminde kalan, sayının ikili gösterimindeki son biti ($2^0$ basamağı) tarafından belirlenir. Çift sayıların son biti 0, tek sayıların son biti 1'dir. Dolayısıyla:

- `(n & 1) == 1` ise $n$ tektir.
- `(n & 1) == 0` ise $n$ çifttir.

Bitwise VE işlemi işlemcide bölme/mod işleminden belirgin şekilde daha ucuzdur.

**2. İkiyle Çarpma ve Bölme:** `n << 1` ifadesi $2n$, `n >> 1` ifadesi $\lfloor n/2 \rfloor$ değerini verir.

**3. $k$. Biti Okuma, Açma (1 Yapma), Kapatma (0 Yapma) ve Tersleme:** Sadece $k$. biti 1, diğer tüm bitleri 0 olan sayı $1 \ll k$ şeklinde elde edilir. Bu sayıya **maske** (mask) denir.

- $n$'in $k$. bitini kontrol etmek: `(n >> k) & 1` veya `n & (1 << k)`.
- $n$'in $k$. bitini 1 yapmak: `n = n  (1 « k)|`.
- $n$'in $k$. bitini 0 yapmak: `n = n & ~(1 << k)`.
- $n$'in $k$. bitini terslemek ($0 \to 1, 1 \to 0$): `n = n ^ (1 << k)`.

**4. 2'nin Kuvveti Olma Kontrolü:** Bir $n > 0$ tamsayısının 2'nin tam bir kuvveti olup olmadığını doğrudan bit yapısından anlayabiliriz. Örneğin $n = 8 = (1000)_2$. Bir eksiğine bakalım: $n - 1 = 7 = (0111)_2$. Bu iki sayıyı VE işlemine sokarsak:

$$8 \;\&\; 7 = (1000)_2 \;\&\; (0111)_2 = (0000)_2 = 0$$

Genel kural olarak: $n$ bir 2'nin kuvveti ise tek bir biti 1'dir. $n-1$ hesaplandığında o bit 0 olur ve sağındaki tüm bitler 1'e dönüşür. Ortak hiçbir 1 biti kalmaz.

**Teorem 32.3.** *Pozitif bir $n$ tamsayısı için $n$ ancak ve ancak* `(n & (n - 1)) == 0` *ise 2'nin bir tam kuvvetidir.*

**5. En Sağdaki 1 Bitini Temizleme ve İzole Etme:** Yukarıdaki gözlem şunu gösterir: `n & (n - 1)` işlemi, $n$ sayısının ikili gösterimindeki en sağda bulunan 1 bitini siler. Öte yandan, negatif sayıların ikinin tümleyeni (two's complement) gösteriminde saklandığını hatırlarsak, $-n = \sim n + 1$ olur. Buradan çıkan kullanışlı bir özdeşlik şudur:

$$\text{En sağdaki 1 biti izole etmek} = n \;\&\; (-n)$$

Örneğin $n = 12 = (00001100)_2$ ise, $-n = (11110100)_2$ olur. VE işlemi yapılırsa:

$$(00001100)_2 \;\&\; (11110100)_2 = (00000100)_2 = 4$$

En sağdaki 1 bitini tek başına elde etmiş oluruz.

**Örnek 32.4.** *Verilen bir $n$ tamsayısının ikili yazımında kaç tane 1 biti (popcount / nüfus sayısı) olduğunu bulan etkin bir algoritma tasarlayınız.*

*Çözüm.* İlk akla gelen yaklaşım sayıyı sürekli 2'ye bölüp (`n >>= 1`) son bitini (`n & 1`) sayaca eklemektir. Bu yöntem toplam bit sayısı (örneğin 32 bit tamsayılar için 32) kadar adım sürer. Ancak sayıda az miktarda 1 biti varsa 32 adım gereksizdir.

Daha önce gördüğümüz `n = n & (n - 1)` özelliğini kullanalım: Bu işlem her adımda en sağdaki bir adet 1 bitini yok eder. Döngüyü sayı 0 olana kadar çalıştırdığımızda, adım sayısı doğrudan sayıdaki 1 bitlerinin sayısına eşit olur. Bu yöntem *Brian Kernighan Algoritması* olarak bilinir.

```
int bit_sayisi(unsigned int n) {
    int sayac = 0;
    while (n > 0) {
        n = n & (n - 1); // en sagdaki 1 bitini siler
        sayac++;
    }
    return sayac;
}
```

Bu algoritmanın karmaşıklığı $O(k)$'dir; burada $k$, $n$'deki 1'lerin sayısıdır. En kötü durumda bile işlem sayısı $O(\log n)$ mertebesinde kalır. □

### 32.1.4 Kümelerin Bitmask Olarak Temsili

Olimpiyatlarda bitwise operatörlerin en etkili kullanım alanlarından biri **küme temsili**dir. $S = \{0, 1, 2, \ldots, n-1\}$ kümesinin herhangi bir $A \subseteq S$ altkümesini $n$ bitlik tek bir tamsayı ile temsil edebiliriz:

$$i \in A \iff \text{maskenin } i.\text{ biti 1'dir.}$$

Örneğin $n = 4$ elemanlı $S = \{0, 1, 2, 3\}$ evrensel kümesini ve $A = \{0, 3\}$ altkümesini ele alalım. 0. ve 3. bitler 1, diğerleri 0 olmalıdır:

$$\text{mask} = 2^0 + 2^3 = 1 + 8 = 9 = (1001)_2$$

Bu eşleme sayesinde küme işlemleri doğrudan bit operatörleriyle ifade edilir:

- Kesişim ($A \cap B$): `maskA & maskB`
- Birleşim ($A \cup B$): `maskA | maskB`
- Simetrik Fark ($A \Delta B$): `maskA ^ maskB`
- Tümleyen ($S \setminus A$): `((1 << n) - 1) ^ maskA`
- Boş küme $\emptyset$: 0
- Evrensel küme $S$: $(1 \ll n) - 1$

**Örnek 32.5.** *$n$ elemanlı bir kümenin toplam $2^n$ adet altkümesi vardır. Bir dizideki tüm altkümelerin eleman toplamlarını ekrana yazdıran bir program parçasını bitmask kullanarak kurunuz.*

*Çözüm.* Her altküme 0 ile $2^n - 1$ arasında tek bir tamsayı ile birebir eşlenir. Bir döngüyle `mask` değerini 0'dan $2^n - 1$'e kadar ilerletip, her `mask` için hangi bitlerin 1 olduğuna bakarak ilgili dizi elemanlarını toplamak yeterlidir.

```
void altkume_toplamlari(int arr[], int n) {
    int toplam_altkume = 1 << n; // 2^n
    for (int mask = 0; mask < toplam_altkume; mask++) {
        int toplam = 0;
        for (int i = 0; i < n; i++) {
            if ((mask >> i) & 1) { // i. eleman kumedeyse
                toplam += arr[i];
            }
        }
        printf("Mask %d: Toplam = %d\n", mask, toplam);
    }
}
```

Bu yaklaşım, $n \leq 20$ civarındaki arama ve kombinatorik problemlerde altkümeleri dolaşmanın en pratik ve düzenli yoludur. □

**Örnek 32.6.** *Size verilen bir dizide, biri hariç her eleman tam olarak iki kez geçmektedir. Yalnızca bir eleman tek bir kez geçmiştir. Bu elemanı $O(n)$ zamanda ve $O(1)$ ek bellek kullanarak bulunuz.*

*Çözüm.* Elemanların frekansını saymak için ek bir dizi tutmak $O(n)$ ek bellek gerektirir. Oysa XOR operatörünün temel cebirsel özellikleri bize bellek kullanmadan bir çözüm sunar:

1. $x \hat{} x = 0$ (Bir sayının kendisiyle XOR'u sıfırdır.)
2. $x \hat{} 0 = x$ (Sıfır etkisiz elemandır.)
3. XOR işlemi değişmeli (commutative) ve birleşmelidir (associative).

Dizideki tüm elemanları sırayla XOR'larsak, çift sayıda gelen elemanlar birbirini sıfırlar ($x \hat{} x = 0$). Geriye yalnızca tek bir kez geçen eleman kalır.

```
int yalniz_eleman(int arr[], int n) {
    int sonuc = 0;
    for (int i = 0; i < n; i++) {
        sonuc ^= arr[i];
    }
    return sonuc;
}
```

Örneğin dizi $\{4, 1, 2, 1, 2\}$ olsun.

$$4 \hat{} 1 \hat{} 2 \hat{} 1 \hat{} 2 = 4 \hat{} (1 \hat{} 1) \hat{} (2 \hat{} 2) = 4 \hat{} 0 \hat{} 0 = 4.$$

Sonuç doğrudan 4 olarak elde edilir. □

## En Sık Yapılan Hatalar

1. **Operatör Önceliği:** C dilinde karşılaştırma operatörleri ($==, !=, <, >$) bitwise operatörlerden (`&`, `|`, `^`) daha yüksek önceliğe sahiptir. Örneğin: `if (x & 1 == 0)` yazıldığında C derleyicisi bunu `if (x & (1 == 0))` yani `if (x & 0)` olarak yorumlar. Doğru kullanım için mutlaka parantez kullanılmalıdır: `if ((x & 1) == 0)`.

2. **Taşma (Overflow) ve 64-bit:** Sola kaydırma yaparken `1 << 40` yazılırsa tanımsız davranış (undefined behavior) oluşur; çünkü `1` sabiti 32-bit tamsayıdır (`int`). 30'dan büyük kaydırmalar için `1LL << 40` (`long long`) yazılmalıdır.

## 32.2 Struct (Yapılar)

### 32.2.1 Motivasyon: İlişkili Verileri Gruplama

Şimdiye kadar gördüğümüz temel tiplerde (`int`, `double`, `char`) her değişken bağımsız bir bellek hücresiydi. Ancak gerçek hayatta veya geometri/graf problemlerinde veriler parçalanamaz kümeler halinde gelir.

Örneğin, 2 boyutlu Kartezyen düzlemde $N$ adet nokta olduğunu varsayalım. Her noktanın bir $x$ koordinatı, bir $y$ koordinatı ve bir de etiket ismi (örneğin bir karakter) olsun. Noktaları saklamak için ayrı ayrı diziler açabiliriz:

```
double x[100];
double y[100];
char isim[100];
```

Buna "paralel diziler" (parallel arrays) yaklaşımı denir. Ancak bu modelleme biçimi hata yapmaya çok açıktır:

- Noktaları $x$ koordinatına göre sıralamak istediğimizde $x$, $y$ ve *isim* dizilerinin elemanlarını aynı anda tek tek takas (swap) etmemiz gerekir. Biri atlandığında veri bütünlüğü bozulur.
- Bir fonksiyon bir noktayı parametre alacaksa fonksiyona üç ayrı argüman göndermek gerekir: `void yazdir(double x, double y, char isim)`.

İşte `struct` (yapı), birbiriyle ilişkili farklı türdeki değişkenleri tek bir çatı altında toplayarak kullanıcı tanımlı yeni bir veri tipi oluşturmamızı sağlar.

### 32.2.2 Struct Tanımı ve Kullanımı

**Tanım 32.7** (Struct Tanımı)**.** *Bir struct aşağıdaki sözdizimi ile bildirilir:*

```
struct Nokta {
    double x;
    double y;
    char isim;
};
```

*Buradaki `x`, `y` ve `isim` yapıya ait* ***alanlar*** *(field ya da member) olarak adlandırılır. Artık `struct Nokta` ifadesi, tıpkı `int` gibi bir tür adıdır.*

Bu yapıdan değişken oluşturmak ve alanlarına erişmek için nokta (`.`) operatörü kullanılır:

```
struct Nokta p1;
p1.x = 3.0;
p1.y = 4.0;
p1.isim = 'A';
```

```
// Veya suslu parantezle dogrudan ilklendirme (initialization):
struct Nokta p2 = {0.0, 0.0, 'O'};
```

#### `typedef` ile Kolaylaştırma

Her defasında `struct Nokta` yazmak kod kalabalığı yaratabilir. C dilindeki `typedef` anahtar kelimesi ile bu türe doğrudan bir takma isim verilebilir:

```
typedef struct {
    double x;
    double y;
} Nokta;

// Artik dogrudan Nokta turunu kullanabiliriz:
Nokta p = {1.5, 2.5};
```

**Örnek 32.8.** *Düzlemde iki nokta arasındaki Öklid mesafesini hesaplayan fonksiyonu `Nokta` yapısını kullanarak yazınız.*

*Çözüm.* İki nokta $P_1 = (x_1, y_1)$ ve $P_2 = (x_2, y_2)$ arasındaki uzaklık formülü şöyledir:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Fonksiyonumuza dört ayrı skaler parametre yerine iki `Nokta` nesnesi göndermek kodun okunabilirliğini ve yönetimini belirgin biçimde kolaylaştırır.

```
#include <math.h>

typedef struct {
    double x;
    double y;
} Nokta;

double mesafe(Nokta p1, Nokta p2) {
    double dx = p1.x - p2.x;
    double dy = p1.y - p2.y;
    return sqrt(dx * dx + dy * dy);
}
```

□

### 32.2.3 Struct Dizileri ve İşaretçiler

Struct nesneleri de tıpkı temel tipler gibi dizilerde saklanabilir. Örneğin 100 kişilik bir öğrenci listesi için:

```
typedef struct {
    int id;
    int puan;
} Ogrenci;

Ogrenci liste[100];
liste[0].id = 101;
liste[0].puan = 85;
```

Bir fonksiyona struct türünde bir değişkeni değer olarak (by-value) geçtiğimizde yapının tamamı kopyalanır. Struct çok sayıda alan içeriyorsa bu kopyalama işlemi gereksiz maliyet oluşturabilir. Bölüm 31'de incelediğimiz işaretçileri (pointer) kullanarak struct'ın yalnızca bellek adresini fonksiyona aktarabiliriz.

İşaretçi üzerinden bir alana erişirken `(*ptr).alan` sözdizimi kullanılır. C dili bu yazımı sadeleştirmek adına **ok operatörünü** (`->`) sunar:

$$\texttt{ptr->alan} \iff \texttt{(*ptr).alan}$$

**Örnek 32.9.** *Bir olimpiyat takımındaki öğrencilerin puanlarını tutan ve puanı en yüksek olan öğrenciyi bulan bir program yazınız.*

*Çözüm.* Bir `Ogrenci` dizisi üzerinde tek bir geçiş yaparak en yüksek puana sahip olan öğrenciyi belirler ve yapının kendisini saklarız.

```
#include <stdio.h>

typedef struct {
    int id;
    int puan;
} Ogrenci;

Ogrenci en_basarili(Ogrenci ogrenciler[], int n) {
    Ogrenci en_iyi = ogrenciler[0];
    for (int i = 1; i < n; i++) {
        if (ogrenciler[i].puan > en_iyi.puan) {
            en_iyi = ogrenciler[i];
        }
    }
    return en_iyi;
}
```

Fonksiyonlar geriye bir `struct` nesnesi de döndürebilir. Böylece tek bir çağrıyla birbiriyle ilişkili birden fazla veri geri iletilmiş olur. □

### 32.2.4 Struct ve Bellek Hizalaması (Padding)

Struct'lar hakkında teorik sınavlarda sıklıkla karşılaşılan bir ayrıntı, bir yapının bellekte kapladığı alandır (`sizeof`). İlk bakışta yapının boyutunun, içindeki ele-

manların boyutlarının toplamına eşit olması beklenir. Ancak işlemciler bellek adresleri 4'ün veya 8'in katı olduğunda verilere daha hızlı erişir. Bu nedenle derleyici, alanlar arasına görünmez boşluklar (padding) yerleştirir.

Örneğin:

```
struct Test {
    char c;     // 1 bayt
    int x;      // 4 bayt
};
```

Burada `sizeof(struct Test)` değeri $1 + 4 = 5$ bayt **değildir**; çoğu modern 32/64-bit derleyicide 8 bayttır. Çünkü `char`'dan sonra işlemcinin `int` verisine 4 baytlık hizalı adresten ulaşabilmesi için 3 baytlık boşluk eklenir.

**Örnek 32.10.** *Kesirli sayıları $(a/b)$ temsil etmek ve iki kesirli sayıyı çarpmak için bir struct yapısı tasarlayınız. Sonucu sadeleştirilmiş biçimde saklayınız.*

*Çözüm.* Bir kesir iki bileşenden oluşur: pay ($a$) ve payda ($b$). Çarpma kuralı doğrudan payların ve paydaların kendi aralarında çarpılmasıdır:

$$\frac{a}{b} \cdot \frac{c}{d} = \frac{a \cdot c}{b \cdot d}$$

Sadeleştirme adımı için Bölüm 13'te ele aldığımız Öklid algoritmasıyla pay ve paydanın en büyük ortak bölenini (gcd) bulur, ardından her iki değeri de bu EBOB'a böleriz.

```
#include <stdio.h>

typedef struct {
    long long pay;
    long long payda;
} Kesir;

long long ebob(long long a, long long b) {
    while (b != 0) {
        long long t = b;
        b = a % b;
        a = t;
    }
    return a;
}

Kesir kesir_carp(Kesir k1, Kesir k2) {
    Kesir sonuc;
    sonuc.pay = k1.pay * k2.pay;
    sonuc.payda = k1.payda * k2.payda;

    long long g = ebob(sonuc.pay, sonuc.payda);
    sonuc.pay /= g;
```

```
    sonuc.payda /= g;

    return sonuc;
}
```

Bu yapı sayesinde rasyonel sayılar üzerindeki işlemler modüler ve tutarlı bir çatı altında toplanmış olur. □

### En Sık Yapılan Hatalar

1. **Struct Kopyalamasında Atama vs Karşılaştırma:** C dilinde `k1 = k2;` geçerli bir ifadedir ve tüm alanları kopyalar. Ancak `if (k1 == k2)` GEÇERSİZDİR. C dili iki struct nesnesini `==` ile doğrudan karşılaştırmayı desteklemez; alanların tek tek karşılaştırılması gerekir.

2. **Noktalı Virgülü Unutmak:** Struct tanımının kapanış parantezinden sonra noktalı virgül (`};`) koymak zorunludur. Unutulduğunda derleyici yanıltıcı sözdizimi hataları üretebilir.

# Bölüm 33

# Alıştırmalar: Sayma

Kombinatorik ve sayma kuramında ustalaşmanın en etkili yolu, öğrenilen soyut ilkeleri farklı kurgularda sınamak ve problem karşısında doğru temsili (birebir eşleme, çifte sayım, tersine düşünme) kurabilmektir. Bu alıştırma setinde; Bölüm 4 ve 5'te gördüğümüz permütasyon ve kombinasyon tekniklerini, Bölüm 6'daki binom özdeşliklerini, Bölüm 7'deki güvercin yuvası ilkesini ve Bölüm 8'deki içerme–dışarma yöntemini bir araya getiren kapsamlı problemleri ele alacağız.

Her alt konudaki problemler; olimpiyat müfredatına uygun olarak [**Kolay**], [**Orta**] ve [**Zor**] düzeylerine ayrılmıştır. Çözümlerde yalnızca işlem adımları verilmemiş, her adımın ardındaki temel düşünce ve yöntemin gerekçesi de açıklanmıştır.

## 33.1 Permütasyon-kombinasyon alıştırmaları

Sayma problemlerine başlarken ilk sorulması gereken soru şudur: *"Elimizdeki nesneler birbirinden ayırt edilebilir mi, yoksa özdeş mi? Sıralama sonucu değiştiriyor mu?"* Çoğu olimpiyat sorusu, bu iki temel ayrımın üzerine ek kısıtlar (örneğin yan yana gelmeme, belirli bir sırada bulunma veya alt–üst sınırlar) ekler.

Bu kısıtları yönetmek için Bölüm 4 ve 5'te ele aldığımız iki temel yaklaşım öne çıkar:

1. **Bağlama (Bloklama) Yöntemi:** Yan yana durması zorunlu olan nesneleri tek bir "blok nesne" kabul edip sürece dahil etmek.

2. **Boşluk (Ayırma) Yöntemi:** Hiçbir ikisi yan yana gelmemesi gereken nesneleri dışarıda bırakıp, serbest nesneleri dizdikten sonra aralarında oluşan boşluklara seçilen nesneleri yerleştirmek.

Özdeş nesnelerin ayırt edilebilir kutulara dağıtımında ise Bölüm 5'te incelediğimiz ayraç (stars and bars) yöntemi kullanılır.

**Teorem 33.1** (Ayraç Yöntemi / Stars and Bars)**.** *$n$ adet özdeş nesne, $k$ adet farklı kutuya:*

1. *Hiçbir kısıt olmaksızın (kutular boş kalabilir) $\binom{n+k-1}{k-1}$ farklı yolla,*
2. *Her kutuda en az bir nesne bulunması koşuluyla (pozitif tam sayılarda çözüm) $\binom{n-1}{k-1}$ farklı yolla*

*dağıtılır.*

*İspat Taslağı.* $n$ özdeş nesneyi yan yana dizelim. Bu nesneleri $k$ gruba ayırmak için aralarına $k-1$ adet ayraç yerleştirmemiz gerekir. Boş kutu olabilecekse toplam $n+k-1$ konumdan $k-1$ tanesini ayraçlar için seçeriz: $\binom{n+k-1}{k-1}$. Her kutu en az bir eleman alacaksa, $n$ nesnenin arasındaki $n-1$ boşluktan $k-1$ tanesi seçilir: $\binom{n-1}{k-1}$. □

Şimdi kolaydan zora doğru kurgulanan çözümlü örnekleri inceleyelim.

**Örnek 33.2** (Kolay: Kısıtlı Dizilim ve Boşluk Yöntemi)**.** 4 *matematikçi ve* 5 *bilgisayar bilimcisi düz bir sıra boyunca dizilecektir. Hiçbir iki matematikçinin yan yana gelmediği kaç farklı dizilim vardır?*

*Çözüm.* Hiçbir iki matematikçinin yan yana gelmemesi koşulu, aralarına en az bir bilgisayar bilimcisinin girmesini gerektirir. Bu nedenle önce serbest durumdaki bilgisayar bilimcilerini dizip, ardından matematikçileri oluşan boşluklara yerleştiririz.

**1. Adım (Serbest nesnelerin dizilimi):** 5 bilgisayar bilimcisi kendi aralarında $5! = 120$ farklı biçimde sıralanır:

$$\_\, B_1 \,\_\, B_2 \,\_\, B_3 \,\_\, B_4 \,\_\, B_5 \,\_$$

**2. Adım (Boşlukların belirlenmesi):** Bilgisayar bilimcilerinin sağı, solu ve aralarında toplam $5+1=6$ adet boşluk oluşur.

**3. Adım (Kısıtlı nesnelerin yerleştirilmesi):** Hiçbir iki matematikçi aynı boşluğa giremez; aksi takdirde yan yana düşerler. Dolayısıyla bu 6 boşluktan 4 tanesini seçip matematikçileri bu boşluklara sıralamalıyız. 6 boşluk arasından 4'ünü seçip 4 matematikçiyi yerleştirmenin yolu permütasyon ile hesaplanır:

$$P(6,4) = 6 \times 5 \times 4 \times 3 = 360$$

**4. Adım (Çarpma kuralı):** Toplam geçerli dizilim sayısı:

$$5! \times P(6,4) = 120 \times 360 = 43\,200$$

olarak bulunur. □

**Örnek 33.3** (Orta: Alt ve Üst Sınırlı Ayraç Yöntemi)**.** $x_1 + x_2 + x_3 + x_4 = 20$ *denkleminin, her* $i \in \{1, 2, 3, 4\}$ *için* $1 \leq x_i \leq 8$ *koşulunu sağlayan kaç farklı tam sayı çözümü vardır?*

*Çözüm.* Alt sınır ($x_i \geq 1$), Bölüm 5'teki ayraç yöntemiyle doğrudan çözülebilir. Ancak üst sınır ($x_i \leq 8$) ek bir kısıttır. Burada en elverişli yaklaşım; tüm çözümlerden, en az bir değişkenin 8'i aştığı durumları Bölüm 8'de gördüğümüz içerme–dışarma ilkesiyle çıkarmaktır.

Önce değişken dönüşümü yapalım: $y_i = x_i - 1$ dersek, $0 \leq y_i \leq 7$ ve toplam:

$$y_1 + y_2 + y_3 + y_4 = 20 - 4 = 16$$

olur. Problem, $y_1 + y_2 + y_3 + y_4 = 16$ denkleminin $0 \leq y_i \leq 7$ aralığındaki tam sayı çözümlerini bulmaya dönüştü.

**Evrensel küme ($U$):** Hiçbir üst sınır olmaksızın ($y_i \geq 0$) çözüm sayısı:

$$|U| = \binom{16 + 4 - 1}{4 - 1} = \binom{19}{3} = \frac{19 \times 18 \times 17}{6} = 969$$

**Kısıtı ihlal eden durumlar:** $A_i$ durumu, $y_i \geq 8$ olsun. Bir değişkenin $y_i \geq 8$ olması için $y_i' = y_i - 8 \geq 0$ tanımlanır. Yeni toplam:

$$y_i' + \sum_{j \neq i} y_j = 16 - 8 = 8$$

Her bir $i$ için çözüm sayısı:

$$|A_i| = \binom{8 + 4 - 1}{3} = \binom{11}{3} = \frac{11 \times 10 \times 9}{6} = 165$$

4 değişken olduğundan, $\sum |A_i| = 4 \times 165 = 660$.

**İki değişkenin birden ihlal etmesi:** İki değişken en az 8 olursa, harcanan miktar en az $8 + 8 = 16$ olur. $y_i \geq 8$ ve $y_j \geq 8$ ($i \neq j$) için:

$$y_i' + y_j' + \sum_{k \notin \{i,j\}} y_k = 16 - 16 = 0$$

Bu durumda çözüm sayısı:

$$|A_i \cap A_j| = \binom{0 + 4 - 1}{3} = \binom{3}{3} = 1$$

$\binom{4}{2} = 6$ farklı çift seçilebileceğinden, $\sum |A_i \cap A_j| = 6 \times 1 = 6$.

Üç değişkenin aynı anda $\geq 8$ olması imkânsızdır çünkü $8 \times 3 = 24 > 16$.

İçerme–dışarma formülünden geçerli çözüm sayısı:

$$N = |U| - \sum |A_i| + \sum |A_i \cap A_j| = 969 - 660 + 6 = 315$$

bulunur. □

**Örnek 33.4** (Zor: Özdeş Elemanlar ve Döngüsel Ayrıklık)**.** *12 kişinin oturduğu yuvarlak bir masadan rastgele 4 kişi seçilecektir. Seçilen hiçbir iki kişinin masada yan yana oturmuyor olma olasılığı kaçtır?*

*Çözüm.* Düz bir sırada yan yana gelmeyen nesneleri boşluk yöntemiyle sayabiliyoruz. Masanın dairesel olması nedeniyle 1. ve 12. kişiler de komşudur. Dairesel simetriyi yönetmek adına belirli bir kişiyi (örneğin 1 numaralı sandalyedekini) referans alarak problemi iki duruma ayırırız: bu kişinin seçildiği ve seçilmediği durumlar.

12 sandalyeyi saat yönünde $1, 2, \ldots, 12$ olarak numaralandıralım. Toplam seçim sayısı:

$$|S| = \binom{12}{4} = \frac{12 \times 11 \times 10 \times 9}{24} = 495$$

Şimdi hiçbir ikisi ardışık olmayan 4 kişilik seçimleri iki ayrık duruma ayıralım:

**Durum 1: 1 numaralı kişi seçilmiştir.** 1 seçildiğine göre komşuları olan 2 ve 12 seçilemez. Geriye kalan $\{3, 4, 5, 6, 7, 8, 9, 10, 11\}$ kümesinden (ardışık 9 kişi) yan yana gelmeyen 3 kişi seçmeliyiz. Bu, düz bir sırada 9 kişiden yan yana gelmeyen 3 kişi seçme problemidir. Seçilmeyen $9 - 3 = 6$ kişi düz bir sıraya dizildiğinde $6 + 1 = 7$ boşluk oluşur:

$$\binom{7}{3} = \frac{7 \times 6 \times 5}{6} = 35$$

**Durum 2: 1 numaralı kişi seçilmemiştir.** Bu durumda geriye kalan $\{2, 3, 4, \ldots, 12\}$ kümesinden (ardışık 11 kişi) düz bir sırada yan yana gelmeyen 4 kişi seçmeliyiz. Seçilmeyen $11 - 4 = 7$ kişi dizildiğinde $7 + 1 = 8$ boşluk oluşur:

$$\binom{8}{4} = \frac{8 \times 7 \times 6 \times 5}{24} = 70$$

**Toplam geçerli seçim sayısı:**

$$35 + 70 = 105$$

İstenen olasılık:

$$P = \frac{105}{495} = \frac{7}{33}$$

□

**Örnek 33.5** (Zor: Monoton Fonksiyonlar ve Çifte Sayım)**.** $A = \{1, 2, 3, 4, 5\}$ *ve* $B = \{1, 2, 3, \ldots, 10\}$ *olsun. Her* $x < y$ *için* $f(x) \leq f(y)$ *şartını sağlayan ve kümesinde tam olarak 3 farklı değer alan (yani* $|f(A)| = 3$ *olan) kaç farklı* $f : A \to B$ *fonksiyonu vardır?*

*Çözüm.* Fonksiyon monoton artandır ($f(1) \leq f(2) \leq f(3) \leq f(4) \leq f(5)$) ve görüntü kümesinde tam 3 farklı eleman bulunmalıdır. Bu görüntü kümesi

$\{y_1, y_2, y_3\} \subset B$ $(y_1 < y_2 < y_3)$ olsun. Tanım kümesindeki 5 elemanı, artanlık sırasını bozmadan bu 3 değere paylaştırırken her değerin en az bir kez kullanılmasını sağlamalıyız.

**1. Adım (Görüntü kümesinin seçimi):** 10 elemanlı $B$ kümesinden 3 farklı değer seçmeliyiz:

$$\binom{10}{3} = \frac{10 \times 9 \times 8}{6} = 120$$

Seçilen bu değerleri küçükten büyüğe $y_1 < y_2 < y_3$ olarak sıralarız (sıralama tek türlüdür).

**2. Adım (Girdilerin paylaştırılması):** $f(1), f(2), f(3), f(4), f(5)$ dizisi zayıf artan olduğundan, ilk $c_1$ tanesi $y_1$, sonraki $c_2$ tanesi $y_2$, son $c_3$ tanesi $y_3$ değerini almalıdır. Burada her değer en az bir kez kullanılacağından:

$$c_1 + c_2 + c_3 = 5 \quad \text{ve} \quad c_1, c_2, c_3 \geq 1$$

olmalıdır. Bu denklem, 5 nesnenin arasına 2 ayraç koymakla eşdeğerdir:

$$\binom{5-1}{3-1} = \binom{4}{2} = 6$$

**3. Adım (Bileşke):** Her görüntü kümesi seçimi için 6 geçerli fonksiyon yazılabildiğinden, toplam fonksiyon sayısı:

$$\binom{10}{3} \times \binom{4}{2} = 120 \times 6 = 720$$

olur. □

```c
#include <stdio.h>
#include <stdbool.h>
#include <string.h>

bool is_valid(const char *arrangement, int length) {
    for (int i = 0; i < length - 1; i++) {
        if (arrangement[i] == 'M' && arrangement[i + 1] == 'M') {
            return false;
        }
    }
    return true;
}

void swap(char *a, char *b) {
    char temp = *a;
    *a = *b;
    *b = temp;
}
```

```c
void reverse(char *arr, int start, int end) {
    while (start < end) {
        swap(&arr[start], &arr[end]);
        start++;
        end--;
    }
}

bool next_permutation(char *arr, int length) {
    int i = length - 2;
    while (i >= 0 && arr[i] >= arr[i + 1]) {
        i--;
    }
    if (i < 0) {
        return false;
    }
    int j = length - 1;
    while (arr[j] <= arr[i]) {
        j--;
    }
    swap(&arr[i], &arr[j]);
    reverse(arr, i + 1, length - 1);
    return true;
}

long long factorial(int n) {
    long long result = 1;
    for (int i = 2; i <= n; i++) {
        result *= i;
    }
    return result;
}

int main(void) {
    char objects[] = "BBBBBMMMM";
    int length = 9;
    int valid_count = 0;

    do {
        if (is_valid(objects, length)) {
            valid_count++;
        }
    } while (next_permutation(objects, length));

    long long total = valid_count * (factorial(4) * factorial(5));
    printf("Toplam geçerli dizilim: %lld\n", total);

    return 0;
}
```

**En sık yapılan hata:** Boşluk yönteminde, boşluk sayısını serbest nesne sayısına eşit almak yaygın bir dikkatsizliktir. $k$ nesne düz bir sıraya dizildiğinde her zaman $k+1$ adet boşluk oluşturur. Dairesel dizilimde ise boşluk sayısı tam olarak $k$ adettir.

## 33.2 Binom ve güvercin yuvası alıştırmaları

Bölüm 6'da gördüğümüz binom teoremi ve kombinatorik özdeşlikler cebirsel ifadelerin katsayılarını sayma argümanlarıyla çözmeyi sağlarken; Bölüm 7'de işlediğimiz Güvercin Yuvası İlkesi (GYİ), belirli bir büyüklüğe ulaşan veri kümelerinde aranan bir yapının ya da çakışmanın kaçınılmaz olarak var olduğunu garanti eder.

**Tanım 33.6** (Güvercin Yuvası İlkesi / Pigeonhole Principle)**.** *$n$ ve $k$ pozitif tam sayılar olsun. $n$ adet nesne $k$ adet yuvaya dağıtıldığında, en az bir yuvada en az $\lceil n/k \rceil$ nesne bulunur. Benzer biçimde, en az bir yuvada en fazla $\lfloor n/k \rfloor$ nesne yer alır.*

Olimpiyatlarda güvercin yuvası ilkesini uygularken en kritik aşama şudur: **Güvercinler nedir, yuvalar nedir?** Yuvalar genellikle modüler sınıflar (Bölüm 14), geometrik bölgeler veya toplamları/farkları sabit olan sayı çiftleri olarak tasarlanır.

**Teorem 33.7** (Vandermonde Özdeşliği)**.** *$m, n, r \in \mathbb{N}$ ve $r \leq m+n$ olmak üzere:*

$$\sum_{k=0}^{r} \binom{m}{k}\binom{n}{r-k} = \binom{m+n}{r}$$

*İspat Taslağı.* $m$ erkek ve $n$ kadından oluşan $m+n$ kişilik bir gruptan $r$ kişilik bir kurul seçilecektir. Toplam seçim sayısı $\binom{m+n}{r}$'dir. Kuruldaki erkek sayısını $k$ $(0 \leq k \leq r)$ olarak sabitlersek, $m$ erkekten $k$, $n$ kadından $r-k$ kişi seçilir. Toplama kuralı gereği iki taraf birbirine eşittir. □

**Örnek 33.8** (Kolay: Bölünebilirlik ve Güvercin Yuvası)**.** *Rastgele seçilen 6 tam sayı arasından, farkları 5 ile tam bölünen en az iki sayının mutlaka bulunacağını gösteriniz.*

*Çözüm.* Bölüm 11 ve 14'te gördüğümüz üzere iki sayının farkının 5 ile tam bölünmesi, bu iki sayının 5 ile bölümünden kalanlarının eşit olması anlamına gelir. Olası kalan sayısı sınırlı olduğundan, seçilen sayı adedi kalan türlerinin sayısını aştığında en az iki sayının aynı kalanı vermesi zorunludur.

**1. Adım (Yuvaların tanımı):** Bir tam sayının 5 ile bölümünden kalanlar $\{0, 1, 2, 3, 4\}$ kümesinin elemanlarıdır. Dolayısıyla $k = 5$ farklı yuva mevcuttur.

**2. Adım (Güvercinlerin dağıtımı):** Seçilen 6 tam sayıyı güvercin olarak alalım $(n = 6)$. Her sayıyı, 5 ile bölümünden kalanın temsil ettiği yuvaya yerleştirelim.

**3. Adım (GYİ uygulanması):** $n = 6 > k = 5$ olduğundan, Güvercin Yuvası İlkesi gereği en az bir yuvaya en az 2 sayı düşer. Bu iki sayı $a$ ve $b$ olsun. $a \equiv b \pmod 5$ olduğundan, $a - b \equiv 0 \pmod 5$, yani $5 \mid (a - b)$ elde edilir. □

**Örnek 33.9** (Orta: Çifte Sayım ile Binom Toplamı)**.** *$n \geq 1$ için aşağıdaki eşitliği kombinatorik bir hikâye kurarak (çift sayma yöntemiyle) ispatlayınız:*

$$\sum_{k=1}^{n} k\binom{n}{k}^2 = n\binom{2n-1}{n-1}$$

*Çözüm.* Bölüm 10'da incelediğimiz çifte sayım yöntemini uygulayalım. Sol tarafta $k\binom{n}{k}^2 = k\binom{n}{k}\binom{n}{n-k}$ ifadesi yer alır. Buradaki $k$ çarpanı gruptan bir başkan seçildiğine, $\binom{n}{k}\binom{n}{n-k}$ çarpımı ise toplam $2n$ kişiden $n$ kişilik bir kurul oluşturulduğuna işaret eder.

Hikâyemizi kuralım: $n$ erkek ve $n$ kadından oluşan toplam $2n$ kişilik bir topluluk olsun. Bu topluluktan $n$ kişilik bir komite ve bu komitenin içinden bir *kadın başkan* seçeceğiz.

**1. Yöntem (Önce komiteyi, sonra başkanı seçmek):** Komitedeki kadın sayısını $k$ $(1 \leq k \leq n)$ olarak belirleyelim.

- $n$ kadın arasından $k$ kadın: $\binom{n}{k}$ yolla,
- Kalan $n - k$ kişi erkeklerden: $\binom{n}{n-k} = \binom{n}{k}$ yolla,
- Seçilen $k$ kadın arasından başkan: $k$ yolla seçilir.

Tüm $k$ değerleri üzerinden toplarsak:

$$\sum_{k=1}^{n} k\binom{n}{k}^2$$

**2. Yöntem (Önce başkanı, sonra komitenin geri kalanını seçmek):**

- Önce $n$ kadın arasından başkan olacak 1 kişiyi seçelim: $n$ farklı seçenek vardır.
- Başkan belirlendikten sonra, komiteye $n - 1$ kişi daha seçilmelidir.
- Geriye kalan adaylar: $n - 1$ kadın ve $n$ erkek olmak üzere toplam $2n - 1$ kişidir.
- Bu $2n - 1$ kişi arasından geriye kalan $n - 1$ komite üyesi seçilir: $\binom{2n-1}{n-1}$ yolla.

Çarpma kuralı gereği toplam durum sayısı:

$$n\binom{2n-1}{n-1}$$

Her iki sayım da aynı nesneleri (başkanı kadın olan $n$ kişilik komiteleri) saydığından eşitlik ispatlanmıştır. □

**Örnek 33.10** (Zor: Erdős Bölünebilirlik Teoremi ve Güvercin Yuvası)**.** *$\{1, 2, 3, \ldots, 2n\}$ kümesinden seçilen herhangi $n+1$ elemanlı bir altkümede, biri diğerini bölen en az iki sayının mutlaka bulunacağını ispatlayınız.*

*Çözüm.* Bölüm 11 ve 12'de gördüğümüz üzere bölünebilirlik ($a \mid b$), sayıların çarpan yapısıyla doğrudan ilişkilidir. Pozitif her tam sayı, 2'nin bir kuvveti ile bir tek sayının çarpımı olarak tek türlü yazılabilir: $x = 2^k \cdot m$ ($m$ tek sayı). Eğer seçilen sayılardan ikisinin tek kısmı ($m$) aynı olursa, büyük sayı küçük sayının 2'nin bir kuvveti kadar katı olacağından biri diğerini mutlaka böler.

**1. Adım (Tek kısımları yuva yapmak):** Her $x \in \{1, 2, \ldots, 2n\}$ sayısını $x = 2^k \cdot m$ biçiminde yazalım; burada $k \geq 0$ bir tam sayı ve $m \in \{1, 3, 5, \ldots, 2n-1\}$ bir tek sayıdır.

**2. Adım (Yuvaları saymak):** 1'den $2n$'ye kadar olan sayılar arasında tam olarak $n$ adet tek sayı vardır:

$$1, 3, 5, \ldots, 2n-1$$

Her tek sayı bir yuvayı temsil etsin. Toplam yuva sayısı $n$'dir.

**3. Adım (Güvercinleri yerleştirmek):** Kümemizden $n+1$ adet sayı seçilmiştir. Her sayıyı kendi tek kısmı olan $m$ yuvasına atayalım. Güvercin sayısı $n+1$, yuva sayısı $n$'dir.

**4. Adım (Sonuç):** Güvercin Yuvası İlkesi gereği, en az bir yuvada birden fazla (en az iki) sayı bulunmalıdır. Bu sayılar $x$ ve $y$ olsun ($x < y$). Tek kısımları aynı olduğundan:

$$x = 2^a \cdot m \quad \text{ve} \quad y = 2^b \cdot m$$

Burada $x < y$ olduğundan $a < b$ olmalıdır. Dolayısıyla:

$$\frac{y}{x} = \frac{2^b \cdot m}{2^a \cdot m} = 2^{b-a} \in \mathbb{Z}^+$$

Yani $x \mid y$ sağlanır. □

**Örnek 33.11** (Zor: Düzlemde Kafes Noktaları ve Orta Nokta)**.** *İki boyutlu Kartezyen düzlemde koordinatları tam sayı olan noktalara kafes noktası denir. Düzlemde seçilen herhangi 5 kafes noktası arasında, birleştiren doğru parçasının orta noktası da bir kafes noktası olan en az iki noktanın mutlaka bulunduğunu gösteriniz.*

*Çözüm.* $A(x_1, y_1)$ ve $B(x_2, y_2)$ noktalarının orta noktası:

$$M = \left( \frac{x_1 + x_2}{2}, \frac{y_1 + y_2}{2} \right)$$

koordinatlarına sahiptir. $M$'nin bir kafes noktası olması için $x_1 + x_2$ ve $y_1 + y_2$ toplamlarının çift olması gerekir. Bölüm 17'de ele aldığımız parite ilkesi uyarınca,

iki tam sayının toplamı ancak ve ancak teklik–çiftlik durumları aynı olduğunda çifttir.

**1. Adım (Parite sınıfları):** Bir tam sayının paritesi 2 farklı değer alabilir: Tek (T) veya Çift (Ç). Bir noktanın koordinat ikilisi $(x, y)$ için 4 olası parite sınıfı vardır:

$$(\text{Ç}, \text{Ç}), \quad (\text{Ç}, \text{T}), \quad (\text{T}, \text{Ç}), \quad (\text{T}, \text{T})$$

Bu 4 sınıf bizim yuvalarımızdır ($k = 4$).

**2. Adım (Güvercinler):** Düzlemden $n = 5$ kafes noktası seçilmiştir.

**3. Adım (GYİ):** $5 > 4$ olduğundan, Güvercin Yuvası İlkesi gereği en az 2 nokta aynı parite sınıfına ait olmalıdır. Bu iki nokta $P_1(x_1, y_1)$ ve $P_2(x_2, y_2)$ olsun. $x_1 \equiv x_2 \pmod 2 \implies x_1 + x_2 \equiv 0 \pmod 2 \implies \frac{x_1+x_2}{2} \in \mathbb{Z}$, $y_1 \equiv y_2 \pmod 2 \implies y_1 + y_2 \equiv 0 \pmod 2 \implies \frac{y_1+y_2}{2} \in \mathbb{Z}$.

Böylece $P_1$ ve $P_2$'nin orta noktası kesinlikle bir kafes noktasıdır. □

```c
#include <stdio.h>

int get_odd_part(int x) {
    while (x % 2 == 0) {
        x /= 2;
    }
    return x;
}

int main(void) {
    int sample_set[] = {2, 3, 5, 7, 9, 11, 13, 15, 17, 19, 12};
    int size = sizeof(sample_set) / sizeof(sample_set[0]);
    int slots[100] = {0};

    for (int i = 0; i < size; i++) {
        int number = sample_set[i];
        int m = get_odd_part(number);

        if (slots[m] != 0) {
            printf("ışÇakma bulundu: %d ve %d (Tek ııksm: %d)\n", slots[m
    ], number, m);
            int min_val = (slots[m] < number) ? slots[m] : number;
            int max_val = (slots[m] > number) ? slots[m] : number;
            printf("%d böler %d: %s\n", min_val, max_val, (max_val %
    min_val == 0) ? "True" : "False");
            break;
        }
        slots[m] = number;
    }

    return 0;
}
```

**En sık yapılan hata:** Güvercin yuvası kurgulanırken yuva sayısını gereğinden büyük seçmek ilkeyi çalışmaz hale getirir. Örneğin kafes noktası probleminde koordinatların mod 3 veya mod 4 değerlerine bakılsaydı yuva sayısı çok büyük çıkacak ve 5 nokta için çakışma garanti edilemeyecekti. En kaba (en küçük) ortak özellik paritedir (mod 2).

## 33.3 İçerme-dışarma ve ızgara yolları alıştırmaları

Bölüm 9'da incelediğimiz ızgara yolları, iki temel hareketin ($S$: Sağa, $Y$: Yukarı) dizilimleriyle ifade edilir. $(0,0)$ noktasından $(m,n)$ noktasına gitmek, $m$ tane $S$ ve $n$ tane $Y$ adımından oluşan $m+n$ uzunluğundaki bir tekrarlı permütasyon sayısına eşittir: $\binom{m+n}{m}$.

Yol üzerinde engeller (yasaklı noktalar) bulunduğunda ya da sınır çizgilerinin ($y=x$ köşegeni gibi) aşılmaması istendiğinde şu iki yöntem devreye girer:

1. **İçerme–Dışarma İlkesi (Bölüm 8):** Birden fazla engelin bulunduğu durumlarda yasaklı noktalardan geçen yolları sistematik olarak ayıklamak.
2. **André Yansıma İlkesi:** Sınır çizgisini aşan yolları, simetrik bir bitiş noktasına giden yollarla birebir eşlemek (Bölüm 10).

**Teorem 33.12** (İçerme–Dışarma İlkesi / Inclusion–Exclusion)**.** *Sonlu $A_1, A_2, \ldots, A_k$ kümeleri için birleşim kümesinin eleman sayısı:*

$$\left|\bigcup_{i=1}^{k} A_i\right| = \sum_{i=1}^{k} |A_i| - \sum_{1\le i<j\le k} |A_i \cap A_j| + \sum_{1\le i<j<l\le k} |A_i \cap A_j \cap A_l| - \cdots + (-1)^{k-1} |A_1 \cap \cdots \cap A_k|$$

**Örnek 33.13** (Kolay: Tek Engelli Izgara Yolu)**.** *Bir robot $(0,0)$ noktasından $(6,5)$ noktasına her adımda yalnızca 1 birim sağa ($+x$) ya da 1 birim yukarı ($+y$) giderek ulaşacaktır. Robotun $(3,2)$ noktasından **geçmeden** $(6,5)$'e ulaşabileceği kaç farklı yol vardır?*

*Çözüm.* Yasaklı noktadan geçmeyen yolları doğrudan saymak yerine, tüm yolların sayısından bu noktaya uğrayan yolların sayısını çıkarmak (tümleyen sayma) işlemi belirgin biçimde kolaylaştırır.

**1. Adım (Tüm yollar):** $(0,0)$'dan $(6,5)$'e toplam $6+5=11$ adım atılmalıdır; bunların 6'sı sağa, 5'i yukarıdır:

$$|T| = \binom{11}{5} = \frac{11\times 10\times 9\times 8\times 7}{120} = 462$$

**2. Adım ($(3,2)$ noktasından geçen yollar):** Bir yolun $(3,2)$ noktasından geçmesi, hareketin iki bağımsız etaba bölünmesi demektir:

- $(0,0)$'dan $(3,2)$'ye: 3 sağ, 2 yukarı $\implies \binom{3+2}{2} = \binom{5}{2} = 10$.
- $(3,2)$'den $(6,5)$'e: Kalan mesafe $6-3=3$ sağ, $5-2=3$ yukarı $\implies \binom{3+3}{3} = \binom{6}{3} = 20$.

Çarpma kuralı gereği $(3,2)$'den geçen yol sayısı:

$$|A| = \binom{5}{2} \times \binom{6}{3} = 10 \times 20 = 200$$

**3. Adım (Tümleyen kuralı):** $(3,2)$'ye hiç uğramayan yol sayısı:

$$|T| - |A| = 462 - 200 = 262$$

bulunur. □

**Örnek 33.14** (Orta: İki Engelli Izgara ve İçerme–Dışarma)**.** $(0,0)$ *noktasından* $(7,6)$ *noktasına yalnızca sağa ve yukarı adımlarla gidecek olan bir hareketli,* $P(2,2)$ *ve* $Q(4,4)$ *noktalarının* ***hiçbirinden geçmemek*** *şartıyla bu hedefe kaç farklı yolla varabilir?*

*Çözüm.* İki ayrı yasaklı nokta ($P$ ve $Q$) bulunmaktadır. $P$'den geçen yollar kümesini $A$, $Q$'dan geçenleri $B$ ile gösterirsek, aranan değer $|(A \cup B)^c| = |T| - |A \cup B|$ olur. Her iki noktaya birden uğrayan yollar ($A \cap B$) iki kez çıkarılacağından, Bölüm 8'deki içerme–dışarma ilkesi gereği bir kez eklenmelidir.

**1. Adım (Tüm yollar $T$):** $(0,0) \to (7,6)$: Toplam 13 adım (7 sağ, 6 yukarı):

$$|T| = \binom{13}{6} = \frac{13 \times 12 \times 11 \times 10 \times 9 \times 8}{720} = 1716$$

**2. Adım ($P(2,2)$'den geçenler, $A$):**

$$|A| = \binom{2+2}{2} \times \binom{(7-2)+(6-2)}{6-2} = \binom{4}{2} \times \binom{9}{4} = 6 \times 126 = 756$$

**3. Adım ($Q(4,4)$'ten geçenler, $B$):**

$$|B| = \binom{4+4}{4} \times \binom{(7-4)+(6-4)}{6-4} = \binom{8}{4} \times \binom{5}{2} = 70 \times 10 = 700$$

**4. Adım (Hem $P$'den hem $Q$'dan geçenler, $A \cap B$):** $(0,0) \to (2,2) \to (4,4) \to (7,6)$:

- $(0,0) \to (2,2)$: $\binom{4}{2} = 6$,
- $(2,2) \to (4,4)$: $4-2=2$ sağ, $4-2=2$ yukarı $\implies \binom{4}{2} = 6$,
- $(4,4) \to (7,6)$: $7-4=3$ sağ, $6-4=2$ yukarı $\implies \binom{5}{2} = 10$.

$$|A \cap B| = 6 \times 6 \times 10 = 360$$

**5. Adım (Birleşim ve sonuç):** İçerme–dışarma ilkesinden en az bir engele çarpan yol sayısı:

$$|A \cup B| = |A| + |B| - |A \cap B| = 756 + 700 - 360 = 1096$$

Hiçbir engele çarpmayan geçerli yolların sayısı:

$$|T| - |A \cup B| = 1716 - 1096 = 620$$

olarak elde edilir. □

**Örnek 33.15** (Zor: André Yansıma İlkesi ve Köşegeni Aşmama)**.** *Bir parçacık* $(0,0)$ *noktasından* $(n,n)$ *noktasına sağa ve yukarı birim adımlarla ilerlemektedir. Parçacığın* $y = x$ *doğrusunun* ***kesinlikle üzerine çıkmadığı*** *(yani her adımda* $y \leq x$ *kaldığı) kaç farklı yol vardır?*

*Çözüm.* Bu kurgu, klasik Dyck yolu ve Catalan sayıları problemidir. İstenen koşul her adımda $y \leq x$ kalmaktır. Kuralı ihlal eden bir yol, en az bir noktada $y = x+1$ doğrusuna temas eder. André'nin yansıma ilkesi; bu ilk temas noktasından sonraki yol parçasını $y = x + 1$ doğrusuna göre simetriğine dönüştürerek Bölüm 10'daki birebir eşleme fikrini kullanır.

**1. Adım (Tüm yollar):** $(0,0)$'dan $(n,n)$'ye kısıtsız tüm yolların sayısı:

$$|T| = \binom{2n}{n}$$

**2. Adım (Yasaklı doğrunun belirlenmesi):** Yolun $y \leq x$ bölgesinden çıkması demek, en az bir noktada $y = x + 1$ doğrusuna temas etmesi demektir. Yasaklı doğru: $L : y = x + 1$.

**3. Adım (Yansıma bijeksiyonu):** $L$ doğrusuna en az bir kez dokunan bir yol ele alalım. Bu yolun $L$ doğrusuna *ilk dokunduğu* noktayı $K(x_k, y_k)$ olarak işaretleyelim. Burada $y_k = x_k + 1$'dir. Yolun $K$'dan bitiş noktası $B(n,n)$'ye kadar olan kısmını $L$ doğrusuna göre yansıtalım. $B(n,n)$ noktasının $y = x+1$ doğrusuna göre simetriği nedir? Doğru denklemi $y - x = 1$'dir. $(n,n)$ noktasının bu doğruya göre yansıması $B'(n-1, n+1)$ noktasıdır.

**4. Adım (Eşlemenin geçerliliği):** $(0,0)$'dan başlayıp $L$'ye değen her yol, yansıtıldıktan sonra $(0,0)$'dan $B'(n-1, n+1)$ noktasına giden bir yola dönüşür. Tersine, $B'(n-1, n+1)$ noktasına giden *her* yol, başlangıçta $y \leq x$ tarafında başlayıp sonunda $y > x$ tarafına geçtiği için $y = x + 1$ doğrusunu kesmek zorundadır. Dolayısıyla $L$'ye değen kötü yollar ile $(0,0) \to (n-1, n+1)$ yolları arasında birebir eşleme (bijeksiyon) vardır!

**5. Adım (Kötü yolların sayısı ve çıkarma):** $(0,0)$'dan $(n-1, n+1)$'e giden yolların sayısı:

$$|K\ddot{o}t\ddot{u}| = \binom{(n-1)+(n+1)}{n-1} = \binom{2n}{n-1}$$

Geçerli yolların sayısı:

$$C_n = |T| - |Kötü| = \binom{2n}{n} - \binom{2n}{n-1}$$

Bu ifadeyi açarsak:

$$\binom{2n}{n} - \frac{n}{n+1}\binom{2n}{n} = \left(1 - \frac{n}{n+1}\right)\binom{2n}{n} = \frac{1}{n+1}\binom{2n}{n}$$

Bu sayılar matematikte *Catalan sayıları* ($C_n$) olarak bilinir. □

**Örnek 33.16** (Zor: Kısıtlı Sayı Dizilimlerinde İçerme–Dışarma)**.** $\{1, 2, 3, 4, 5, 6\}$ *kümesinin elemanlarıyla yazılabilecek 6 basamaklı rakamları farklı sayılardan kaç tanesinde:*

- *"1" rakamı hemen "2"nin ardından gelmez (yani "21" alt dizisi bulunmaz),*
- *"3" rakamı hemen "4"ün ardından gelmez (yani "43" alt dizisi bulunmaz),*
- *"5" rakamı hemen "6"nın ardından gelmez (yani "65" alt dizisi bulunmaz)?*

*Çözüm.* Soruda üç ayrı yasaklı durum tanımlanmıştır: $B_1 = \texttt{21}$, $B_2 = \texttt{43}$ ve $B_3 = \texttt{65}$ blokları. Bu blokların hiçbirinin yer almadığı durumları bulmak için Bölüm 8'deki içerme–dışarma ilkesini uygularız. Blokları bağlama yöntemiyle tek bir nesne kabul ederek kesişim kümelerinin eleman sayılarını hesaplayabiliriz.

**Evrensel Küme ($U$):** 6 farklı rakamın dizilimleri:

$$|U| = 6! = 720$$

$A_1$: "21" bloğunun bulunduğu dizilimler, $A_2$: "43" bloğunun bulunduğu dizilimler, $A_3$: "65" bloğunun bulunduğu dizilimler.

**1. Derece Kesişimler ($\sum |A_i|$):** $A_1$ için $\{(21), 3, 4, 5, 6\}$ olmak üzere 5 nesne vardır:

$$|A_1| = |A_2| = |A_3| = 5! = 120$$

$$\sum |A_i| = 3 \times 120 = 360$$

**2. Derece Kesişimler ($\sum |A_i \cap A_j|$):** Örneğin $A_1 \cap A_2$ için $\{(21), (43), 5, 6\}$ olmak üzere 4 nesne vardır:

$$|A_1 \cap A_2| = |A_1 \cap A_3| = |A_2 \cap A_3| = 4! = 24$$

$\binom{3}{2} = 3$ çift olduğundan:

$$\sum |A_i \cap A_j| = 3 \times 24 = 72$$

**3. Derece Kesişim ($|A_1 \cap A_2 \cap A_3|$):** Üç blok da mevcuttur: $\{(21), (43), (65)\}$. Toplam 3 nesne vardır:

$$|A_1 \cap A_2 \cap A_3| = 3! = 6$$

**İçerme–Dışarma Formülü:** İstenen koşulları sağlayan permütasyon sayısı:

$$N = |U| - \sum |A_i| + \sum |A_i \cap A_j| - |A_1 \cap A_2 \cap A_3|$$

$$N = 720 - 360 + 72 - 6 = 426$$

bulunur. □

```
#include <stdio.h>
#include <stdbool.h>

typedef struct {
    int x;
    int y;
} Point;

long long count_paths_with_obstacles(int width, int height, const Point
    obstacles[], int obstacle_count) {
    long long dp[width + 1][height + 1];
    bool blocked[width + 1][height + 1];

    for (int x = 0; x <= width; x++) {
        for (int y = 0; y <= height; y++) {
            dp[x][y] = 0;
            blocked[x][y] = false;
        }
    }

    for (int i = 0; i < obstacle_count; i++) {
        int ox = obstacles[i].x;
        int oy = obstacles[i].y;
        if (ox >= 0 && ox <= width && oy >= 0 && oy <= height) {
            blocked[ox][oy] = true;
        }
    }

    dp[0][0] = 1;

    for (int x = 0; x <= width; x++) {
        for (int y = 0; y <= height; y++) {
            if (blocked[x][y]) {
                dp[x][y] = 0;
                continue;
            }
            if (x > 0) {
                dp[x][y] += dp[x - 1][y];
```

```
            }
            if (y > 0) {
                dp[x][y] += dp[x][y - 1];
            }
        }
    }

    return dp[width][height];
}

int main(void) {
    int width = 7;
    int height = 6;
    Point obstacles[] = {{2, 2}, {4, 4}};
    int obstacle_count = sizeof(obstacles) / sizeof(obstacles[0]);

    long long result = count_paths_with_obstacles(width, height,
    obstacles, obstacle_count);
    printf("Engellere çarpmayan yol ıısays: %lld\n", result);

    return 0;
}
```

**En sık yapılan hata:** Izgara üzerinde birden fazla engel varken, engeller arasındaki yolları çarparken göreli koordinatları almak yerine mutlak koordinatları almak en yaygın hatadır. $(x_1, y_1)$ noktasından $(x_2, y_2)$ noktasına giderken adım sayısı $x_2 + y_2$ değil, $(x_2 - x_1) + (y_2 - y_1)$'dir ve kombinasyon $\binom{(x_2-x_1)+(y_2-y_1)}{x_2-x_1}$ biçiminde yazılmalıdır. Ayrıca $x_2 < x_1$ veya $y_2 < y_1$ ise o iki engel arasında hiç yol bulunamaz (yol sayısı 0'dır).

# Bölüm 34

# Alıştırmalar: Sayılar ve Teknikler

Sayılar teorisi ve problem çözme teknikleri boyunca edindiğimiz kuramsal altyapıyı, TÜBİTAK Bilgisayar Olimpiyatı 1. Aşama sınavında ve benzeri yarışmalarda karşımıza çıkan özgün sorular üzerinde sınayacağız. Bir matematiksel aracı yalnızca tanımakla yetinmeyip yarışma ortamında doğru anda devreye sokabilmek ayrı bir problem çözme alışkanlığı gerektirir.

Bu sezgiyi geliştirmek adına ele aldığımız tüm problemler [**Kolay**], [**Orta**], [**İleri Düzey**] ve [**Olimpiyat**] zorluk seviyeleriyle etiketlenmiştir. Her soruda yalnızca sonuca ulaşmakla kalmayıp çözümün arkasındaki temel düşünceyi, çıkış noktalarını ve sıkça düşülen tuzakları da inceleyeceğiz.

## 34.1 Bölünebilme ve Modüler Alıştırmalar

Bölüm 11 ve Bölüm 12'de gördüğümüz bölünebilme kuralları ile Bölüm 14'te ele aldığımız modüler aritmetik araçları, algoritma analizinden kriptografiye kadar bilgisayar biliminin birçok alanında temel rol oynar. Karşımıza çıkan bir bölünebilme sorusunda ilk adım genellikle cebirsel çarpanlara ayırma veya uygun bir modül altında denklik aramak olmalıdır. Doğru modülü belirlemek çoğu zaman çözüm yolunu doğrudan açar.

**Örnek 34.1** (Zorluk: Kolay)**.** *$n$ bir pozitif tamsayı olmak üzere, $n^5 - 5n^3 + 4n$ ifadesinin her zaman* 120 *ile bölündüğünü gösteriniz.*

*Çözüm.* $120 = 2^3 \cdot 3 \cdot 5 = 5!$ olduğuna dikkat edersek, Bölüm 11'den hatırlanacağı üzere ardışık $k$ tam sayının çarpımı daima $k!$ ile tam bölünür. Dolayısıyla ifadeyi ardışık 5 tam sayının çarpımı biçiminde yazmaya çalışmak en doğrudan yaklaşımdır.

Verilen cebirsel ifadeyi ortak çarpan parantezine alarak çarpanlarına ayıralım:

$$\begin{aligned} n^5 - 5n^3 + 4n &= n(n^4 - 5n^2 + 4) \\ &= n(n^2 - 1)(n^2 - 4) \\ &= n(n-1)(n+1)(n-2)(n+2) \end{aligned}$$

Çarpanları küçükten büyüğe sıraladığımızda:

$$(n-2)(n-1)n(n+1)(n+2)$$

çarpımını elde ederiz. Bu ifade ardışık 5 tamsayının çarpımıdır.

Ardışık 5 sayıdan:

- En az iki tanesi 2'ye bölünür ve bunlardan biri en az 4'e bölünür; dolayısıyla çarpım $2 \cdot 4 = 8$ ile tam bölünür ($2^3 \mid 120$).
- En az bir tanesi 3 ile tam bölünür ($3 \mid 120$).
- En az bir tanesi 5 ile tam bölünür ($5 \mid 120$).

$\gcd(8,3) = \gcd(24,5) = 1$ olduğundan, ifademiz $8 \cdot 3 \cdot 5 = 120$ ile daima tam bölünür. □

**Örnek 34.2** (Zorluk: Orta)**.** *$p$ bir asal sayı olsun. $8p^2 + 1$ sayısının da bir asal sayı olduğu bilindiğine göre, $8p^2 - 1$ sayısının asal olup olmadığını belirleyiniz.*

*Çözüm.* Asal sayılar içeren polinom ifadelerinde küçük modülleri incelemek, özellikle de Bölüm 14'te ele aldığımız $\pmod 3$ aritmetiğini kullanmak iyi bir başlangıçtır. Tam kareler mod 3 altında yalnızca 0 veya 1 değerini alabildiğinden ($x^2 \equiv 0, 1 \pmod 3$), bu kısıt ifadenin davranışını dar bir aralığa hapseder.

$p$ asal sayısını mod 3 altında inceleyelim:

1. Durum $p = 3$ ise:

   $$8p^2 + 1 = 8(3^2) + 1 = 8 \cdot 9 + 1 = 73$$

   73 gerçekten bir asal sayıdır. Bu durumda:

   $$8p^2 - 1 = 8(3^2) - 1 = 72 - 1 = 71$$

   71 sayısı da asaldır.

2. Durum $p \neq 3$ ise: $p$ asal ve 3'ten farklı olduğundan $\gcd(p,3) = 1$'dir. Dolayısıyla $p \equiv 1$ veya $p \equiv 2 \pmod 3$ olur. Her iki durumda da $p^2 \equiv 1 \pmod 3$ elde edilir.

   Bu denklikten hareketle $8p^2 + 1$ ifadesini mod 3'te inceleyelim:

   $$8p^2 + 1 \equiv 2 \cdot (1) + 1 = 3 \equiv 0 \pmod 3$$

Bu durumda $8p^2+1$ sayısı 3 ile tam bölünür. Ancak soruda $8p^2+1$'in bir asal sayı olduğu verilmiştir. 3'ün katı olan yegâne asal sayı 3'ün kendisidir. Fakat $p \geq 2$ için $8p^2+1 \geq 8(4)+1 = 33 > 3$ olduğundan bu ifade asla 3 olamaz; dolayısıyla asal olamaz. Bu durum bir çelişkidir.

Demek ki şartı sağlayan yegâne asal $p = 3$'tür ve bu değer için $8p^2-1 = 71$ olup **asaldır**. □

**En Sık Yapılan Hata:** $p \neq 3$ durumunda $p^2 \equiv 1 \pmod 3$ bağıntısını fark edip $8p^2+1$'in 3 ile bölündüğünü bulan bazı öğrenciler, "o halde $8p^2-1 \equiv 2\cdot 1-1 = 1 \pmod 3$ olur, demek ki 3'e bölünmez, o zaman asaldır" yanılgısına düşerler. Bir sayının 3'e bölünmemesi onun asal olduğunu göstermez (örneğin 25 veya 49 gibi). Çözümün anahtarı, $p \neq 3$ durumunun soru koşuluyla tamamen çeliştiğini ($8p^2+1$ asal olamaz) fark edip tek çözümün $p = 3$ olduğunu görmektir.

**Örnek 34.3** (Zorluk: İleri Düzey)**.** *$p$ ve $q$ iki asal sayı olmak üzere,*

$$p \mid q^3-1 \quad ve \quad q \mid p-1$$

*koşulları sağlanmaktadır. Bu koşulu sağlayan tüm $(p,q)$ asal sayı ikililerini bulunuz.*

*Çözüm.* İki yönlü bölünebilme bağıntılarında eşitsizlik kurmak veya çarpanlara ayırmak temel yaklaşımdır. İkinci koşul olan $q \mid p-1$, $p-1 > 0$ olduğu için doğrudan $q \leq p-1 < p$ eşitsizliğini verir; yani kesinlikle $q < p$'dir.

Şimdi ilk koşula dönelim:

$$p \mid q^3-1 = (q-1)(q^2+q+1)$$

$p$ bir asal sayıdır. Bölüm 12'de gördüğümüz Öklid lemmasına göre, $p$ bir çarpımı bölüyorsa çarpanlardan en az birini bölmek zorundadır:

$$p \mid (q-1) \quad \text{veya} \quad p \mid (q^2+q+1)$$

Fakat az önce $q < p$, dolayısıyla $q-1 < p$ olduğunu bulmuştuk. Pozitif bir tam sayı kendisinden kesinlikle büyük bir sayıya bölünemez. O halde $p \nmid (q-1)$ olmalıdır.

Buradan zorunlu olarak:

$$p \mid q^2+q+1$$

sonucuna varırız. Bu ise bir tamsayı $k \geq 1$ için:

$$q^2+q+1 = k \cdot p$$

anlamına gelir.

Diğer yandan $q \mid p-1$ verildiğinden, bir pozitif $m$ tamsayısı için $p-1 = mq$, yani $p = mq+1$ yazabiliriz. Bunu yukarıdaki eşitlikte yerine koyalım:

$$q^2+q+1 = k(mq+1) = kmq+k$$

Eşitliği mod $q$ altında incelersek:

$$1 \equiv k \pmod{q}$$

elde edilir. $k \geq 1$ pozitif tamsayısı için iki temel ihtimal vardır:

1. $k = 1$ durumu: Eğer $k = 1$ ise, $q^2 + q + 1 = p$ olur. Ayrıca $p = mq + 1$ olduğundan:

   $$q^2 + q + 1 = mq + 1 \implies q^2 + q = mq \implies q(q+1) = mq \implies m = q + 1$$

   Bu durumda $p = q^2 + q + 1$ olmalıdır. Küçük asalları deneyelim:

   - $q = 2 \implies p = 2^2 + 2 + 1 = 7$. $p = 7$ asaldır. Kontrol edelim: $7 \mid 2^3 - 1 = 7$ (sağlar) ve $2 \mid 7 - 1 = 6$ (sağlar). İkili: $(7, 2)$.
   - $q = 3 \implies p = 3^2 + 3 + 1 = 13$. $p = 13$ asaldır. Kontrol: $13 \mid 3^3 - 1 = 26$ (sağlar) ve $3 \mid 13 - 1 = 12$ (sağlar). İkili: $(13, 3)$.
   - $q > 3$ tek asal ise: $q \equiv 1$ veya $q \equiv 2 \pmod 3$ olabilir mi? Asallığı test edebiliriz; örneğin $q = 5 \implies p = 31$ asaldır; $31 \mid 124 = 4 \times 31$ ve $5 \mid 30$. Genel olarak $q^2 + q + 1$ asal olduğu sürece her $(q^2 + q + 1, q)$ çifti bir çözüm adayıdır. Ancak $k$'nın diğer değerlerini de araştırmalıyız.

2. $k > 1$ durumu: $k \equiv 1 \pmod q$ ve $k > 1$ olduğundan en az $k \geq q+1$ olmalıdır. Fakat $k \geq q + 1$ olursa:

   $$k \cdot p \geq (q+1)p > (q+1)q = q^2 + q$$

   Hatta $p > q$ olduğu için $p \geq q + 1$'dir (biri tek biri çift veya ikisi de tek). Eğer $p \geq q + 2$ ise:

   $$kp \geq (q+1)(q+2) = q^2 + 3q + 2 > q^2 + q + 1$$

   olur ki bu $kp = q^2 + q + 1$ eşitliğiyle çelişir! Sadece $p = q + 1$ olabilir mi? İki ardışık asal yalnızca $q = 2, p = 3$ iken mümkündür. Ama $q = 2, p = 3$ için $q^2 + q + 1 = 7$, $3 \nmid 7$ olduğundan sağlamaz.

Dolayısıyla tek olası aile $k = 1$, yani $p = q^2 + q + 1$ ailesidir. Problem belirli ikilileri sorduğunda, sorunun bağlamına göre tüm çözümler $q$ ve $p = q^2 + q + 1$'in her ikisinin de asal olduğu ikililerdir. □

**Örnek 34.4** (Zorluk: Olimpiyat)**.** *$n > 1$ bir tamsayı olsun. $n \mid 2^n - 1$ koşulunu sağlayan hiçbir $n > 1$ tamsayısının bulunmadığını ispatlayınız.*

*Çözüm.* Bu problem, yarışma matematiğinde "en küçük asal bölen" argümanının en öğretici ve bilinen örneklerindendir. $n \mid 2^n - 1$ ifadesinde $n$'nin asal çarpanlarının tümü yerine yalnızca *en küçük* asal bölen $p$'ye odaklanmak, Bölüm 14'te

gördüğümüz Fermat'nın Küçük Teoremi sayesinde güçlü bir mertebe kısıtı kurmamızı sağlar.

Olmayana ergi (çelişki) yöntemini kullanalım. $n > 1$ için $n \mid 2^n - 1$ olduğunu varsayalım.

$n > 1$ olduğundan $n$'nin en az bir asal böleni vardır. $n$'nin **en küçük asal bölenine** $p$ diyelim.

- $2^n - 1$ daima tek bir sayı olduğundan, onu bölen $n$ de tek sayı olmalıdır. Dolayısıyla $p$ tek bir asal sayıdır ($p \geq 3$).
- $p \mid n$ ve $n \mid 2^n - 1$ olduğundan, bölünebilmenin geçişme özelliğinden:

$$p \mid 2^n - 1 \implies 2^n \equiv 1 \pmod{p}$$

Şimdi 2'nin mod $p$'deki mertebesini (yani $2^d \equiv 1 \pmod{p}$ şartını sağlayan en küçük pozitif $d$ tamsayısını) ele alalım ve buna $d = \mathrm{ord}_p(2)$ diyelim.

Mertebenin temel teoremine göre:

$$2^n \equiv 1 \pmod{p} \implies d \mid n$$

Ayrıca Fermat'nın Küçük Teoremi gereğince $\gcd(2, p) = 1$ olduğundan:

$$2^{p-1} \equiv 1 \pmod{p} \implies d \mid p - 1$$

elde edilir. Demek ki $d$, hem $n$'yi hem de $p - 1$'i bölen bir tamsayıdır:

$$d \mid \gcd(n, p - 1)$$

Şimdi dikkatle düşünelim:

- $d > 1$ midir? Eğer $d = 1$ olsaydı, $2^1 \equiv 1 \pmod{p} \implies p \mid (2-1) = 1$ olurdu; bu imkânsızdır çünkü $p \geq 3$ bir asaldır. O halde $d > 1$'dir.
- $d > 1$ olduğuna göre $d$'nin en az bir asal böleni vardır; bu asal bölen aynı zamanda $n$'yi de böler (çünkü $d \mid n$).
- Fakat $d \mid p - 1$ olduğundan, $d \leq p - 1 < p$ olmalıdır!

Bu durum açık bir çelişkidir: $d$'nin asal çarpanı hem $n$'yi böler hem de $p$'den kesinlikle küçüktür. Oysa biz $p$'yi $n$'nin *en küçük* asal böleni olarak seçmiştik!

Bu çelişki varsayımımızın yanlış olduğunu kanıtlar. Demek ki $n \mid 2^n - 1$ şartını sağlayan hiçbir $n > 1$ tamsayısı yoktur. □

Aşağıdaki C kodu, küçük $n$ değerleri için bu imkânsızlığı deneysel olarak doğrulamakta ve adayların mertebe davranışını simüle etmektedir:

```
#include <stdio.h>
#include <stdlib.h>

long long power_mod(long long base, long long exp, long long mod) {
    long long result = 1;
    base %= mod;
    while (exp > 0) {
        if (exp % 2 == 1) {
            result = (result * base) % mod;
        }
        base = (base * base) % mod;
        exp /= 2;
    }
    return result;
}

int* check_divisibility(int limit, int* count) {
    int* solutions = (int*)malloc(limit * sizeof(int));
    *count = 0;
    for (int n = 2; n <= limit; ++n) {
        if (power_mod(2, n, n) == 1) {
            solutions[(*count)++] = n;
        }
    }
    return solutions;
}

int main(void) {
    int count = 0;
    int* solutions = check_divisibility(10000, &count);

    printf("Cozumler: [");
    for (int i = 0; i < count; ++i) {
        printf("%d%s", solutions[i], (i == count - 1) ? "" : ", ");
    }
    printf("]\n");

    free(solutions);
    return 0;
}
```

## 34.2 EBOB/EKOK ve Tam Sayı Denklem Alıştırmaları

Bölüm 13'te incelediğimiz üzere, tam sayı çözümleri aradığımız denklemlerde en sık başvurulan yöntemler şunlardır:

1. **Çarpanlara Ayırma:** Denklemi $(A)(B) = C$ formatına getirip $C$'nin bölenlerini eşitlemek.

2. **Modüler Kısıtlama:** Eşitliğin her iki tarafının belirli bir modda (çoğunlukla 3, 4, 8) alabileceği değerlerin uyuşmadığını göstererek çözüm olmadığını ispatlamak.

3. **Eşitsizlik ve Sıkıştırma:** Bir bilinmeyeni diğerinin ardışık kareleri veya küpleri arasına sıkıştırarak aralık daraltmak.

**Örnek 34.5** (Zorluk: Kolay)**.** *Her $n$ pozitif tamsayısı için*

$$\gcd(2n+1,\, 9n+4)$$

*ifadesinin alabileceği değerleri bulunuz.*

*Çözüm.* Bölüm 13'te gördüğümüz Öklid algoritmasının temel kuralı $\gcd(a, b) = \gcd(a, b - ka)$ özelliğidir. Buradaki amaç, $n$ değişkenini yok edecek katsayıları belirlemektir.

$d = \gcd(2n+1, 9n+4)$ olsun. Bölünebilme tanımı gereği:

$$d \mid (2n+1) \quad \text{ve} \quad d \mid (9n+4)$$

Lineer kombinasyon aldığımızda $n$'li terimleri yok etmek için birinciyi 9, ikinciyi 2 ile çarpıp çıkaralım:

$$d \mid \big[2(9n+4) - 9(2n+1)\big]$$

Parantez içini açarsak:

$$2(9n+4) - 9(2n+1) = 18n + 8 - 18n - 9 = -1$$

Buradan $d \mid -1$, yani $d \mid 1$ elde edilir.

EBOB tanımı gereği pozitif bir tamsayı olduğundan, her $n \geq 1$ için:

$$\gcd(2n+1, 9n+4) = 1$$

olmak zorundadır. İfade yalnızca 1 değerini alabilir. □

**Örnek 34.6** (Zorluk: Orta)**.** *$x$ ve $y$ pozitif tamsayılar olmak üzere,*

$$\frac{1}{x} + \frac{1}{y} = \frac{1}{2024}$$

*denklemini sağlayan kaç farklı $(x, y)$ sıralı ikilisi vardır?*

*Çözüm.* Rasyonel ifadeler içeren bu tür denklemler payda eşitlenerek standart bir çarpım biçimine dönüştürülebilir. Bu yöntem olimpiyat literatüründe Simon'ın Çarpanlara Ayırma Özdeşliği adıyla da anılır.

Denklemde payda eşitleyelim:

$$\frac{x+y}{xy} = \frac{1}{2024} \implies xy = 2024(x+y) \implies xy - 2024x - 2024y = 0$$

Her iki tarafa $2024^2$ ekleyerek sol tarafı çarpanlarına ayıralım:

$$xy - 2024x - 2024y + 2024^2 = 2024^2$$

$$(x - 2024)(y - 2024) = 2024^2$$

$u = x - 2024$ ve $v = y - 2024$ dönüşümü yapalım. $x, y > 0$ ve $1/x < 1/2024$ olduğundan $x > 2024$, yani $u > 0$ ve benzer şekilde $v > 0$ olmak zorundadır.

O halde $(x, y)$ ikililerinin sayısı, $2024^2$ sayısının pozitif bölen sayısına eşittir.

Şimdi 2024 sayısını asal çarpanlarına ayıralım:

$$2024 = 2^3 \cdot 253 = 2^3 \cdot 11 \cdot 23$$

Bunun karesini alırsak:

$$2024^2 = (2^3 \cdot 11^1 \cdot 23^1)^2 = 2^6 \cdot 11^2 \cdot 23^2$$

Bir sayının pozitif bölen sayısı, asal çarpanlarının üslerinin birer fazlasının çarpımıdır:

$$d(2024^2) = (6+1)(2+1)(2+1) = 7 \cdot 3 \cdot 3 = 63$$

Dolayısıyla denklemi sağlayan tam 63 farklı $(x, y)$ pozitif tamsayı ikilisi vardır. □

**Örnek 34.7** (Zorluk: İleri Düzey)**.** *$x^2 + 3 = y^3$ denkleminin tamsayılarda hiçbir çözümünün olmadığını kanıtlayınız.*

*Çözüm.* Bu problem bir Mordell eğrisi ($y^3 = x^2 + k$) tipindedir. Modüler kısıtlama ararken küplerin ve karelerin mod 8, mod 4 veya mod 9'daki davranışına bakarız. Küpler mod 8'de $0, 1, 3, 5, 7$ değerlerini alabilirken kareler sadece $0, 1, 4$ değerlerini alabilir.

$y$'nin paritesini inceleyelim:

1. Eğer $y$ çift ise: $y \equiv 0 \pmod 2 \implies y^3 \equiv 0 \pmod 8$ olur. Denklemden:

$$x^2 + 3 = y^3 \equiv 0 \pmod 8 \implies x^2 \equiv -3 \equiv 5 \pmod 8$$

Fakat tam karelerin mod 8'deki denkliği yalnızca $x^2 \equiv 0, 1, 4 \pmod 8$ olabilir. $x^2 \equiv 5 \pmod 8$ denkleminin hiçbir tamsayı çözümü yoktur! Demek ki $y$ çift olamaz.

2. O halde $y$ tek tamsayı olmalıdır. $y$ tek ise $y^3$ de tektir. $x^2+3$ tek olduğuna göre $x^2$ çift, yani $x$ çift olmalıdır. $x$ çift olduğundan $x = 2k$ yazabiliriz. Buradan:

$$x^2 \equiv 0 \pmod 4 \implies y^3 = x^2 + 3 \equiv 3 \pmod 4$$

Tek sayıların küpleri mod 4'te incelenirse:

$$1^3 \equiv 1 \pmod 4$$
$$3^3 = 27 \equiv 3 \pmod 4$$

Dolayısıyla $y \equiv 3 \pmod 4$ veya denk olarak $y \equiv -1 \pmod 4$ olmalıdır.

Şimdi denklemi şu şekilde yeniden düzenleyelim:

$$x^2 + 4 = y^3 + 1 = (y+1)(y^2 - y + 1)$$

$y \equiv 3 \pmod 4$ olduğundan ikinci çarpanı inceleyelim:

$$y^2 - y + 1 \equiv 3^2 - 3 + 1 = 9 - 3 + 1 = 7 \equiv 3 \pmod 4$$

$y^2 - y + 1$ sayısı $4k+3$ formunda tek bir sayıdır.

$4k+3$ formundaki her sayının, yine $4k+3$ formunda en az bir $p$ asal böleni olmak zorundadır (çünkü tüm asal çarpanları $4k+1$ formunda olsaydı çarpımları da $4k+1$ formunda olurdu).

O halde $p \equiv 3 \pmod 4$ şartını sağlayan bir $p$ asalı vardır öyle ki:

$$p \mid (y^2 - y + 1) \implies p \mid (x^2 + 4) \implies x^2 \equiv -4 \pmod p$$

Her iki tarafın $(p-1)/2$. kuvvetini alırsak (Fermat'nın Küçük Teoremi):

$$(x^2)^{(p-1)/2} \equiv (-4)^{(p-1)/2} \pmod p$$

Sol taraf $x^{p-1} \equiv 1 \pmod p$'dir ($p \nmid x$ çünkü $p \mid x \implies p \mid 4 \implies p = 2$, çelişki). Sağ taraf ise:

$$(-4)^{(p-1)/2} = (-1)^{(p-1)/2} \cdot (2^2)^{(p-1)/2}$$
$$= (-1)^{(p-1)/2} \cdot 2^{p-1} \equiv (-1)^{(p-1)/2} \pmod p$$

$p \equiv 3 \pmod 4$ olduğundan $(p-1)/2$ bir tek sayıdır; dolayısıyla $(-1)^{(p-1)/2} = -1$'dir.

Buradan $1 \equiv -1 \pmod p \implies p \mid 2$ elde edilir ki bu $p \equiv 3 \pmod 4$ olmasıyla çelişir!

Her iki durumda da çelişki elde edildiğinden denklemin hiçbir tamsayı çözümü yoktur. □

**Örnek 34.8** (Zorluk: Olimpiyat)**.**

$$x^4 + x^3 + x^2 + x + 1 = y^2$$

*denklemini sağlayan tüm* $(x, y)$ *tamsayı çiftlerini bulunuz.*

*Çözüm.* Dördüncü dereceden bir polinomun tam kareye eşit olması istendiğinde en etkili yöntem, ifadeyi ardışık iki tam kare arasına sıkıştırmaktır. $x^4 + x^3 + \ldots$ ifadesi $(x^2 + ax + b)^2$ açılımına yakındır. Polinomu iki komşu tam kare arasına hapsettiğimizde, arada başka bir tam kare yer alamayacağından çözümler yalnızca sınır eşitliklerinden gelebilir.

Sol tarafı kesirlerden kurtulmak adına 4 ile çarpalım:

$$4y^2 = 4x^4 + 4x^3 + 4x^2 + 4x + 4$$

Bu ifadeyi $(2x^2 + x)^2$ ile karşılaştıralım:

$$(2x^2 + x)^2 = 4x^4 + 4x^3 + x^2$$

Açıkça $4x^4 + 4x^3 + 4x^2 + 4x + 4 > 4x^4 + 4x^3 + x^2$ eşitsizliği:

$$3x^2 + 4x + 4 > 0$$

durumunda geçerlidir. Bu ifadenin diskriminantı $\Delta = 16 - 4(3)(4) = -32 < 0$ olduğundan, her $x \in \mathbb{R}$ için $3x^2 + 4x + 4 > 0$'dır. Demek ki her tamsayı $x$ için:

$$(2x^2 + x)^2 < 4y^2$$

eşitsizliği kesin olarak sağlanır.

Şimdi bir sonraki kareyi, yani $(2x^2 + x + 2)^2$ ifadesini deneyelim:

$$(2x^2 + x + 2)^2 = 4x^4 + 4x^3 + 9x^2 + 4x + 4$$

Bizim ifademiz $4y^2 = 4x^4 + 4x^3 + 4x^2 + 4x + 4$ idi. Farkı alırsak:

$$(2x^2 + x + 2)^2 - 4y^2 = 5x^2 \geq 0$$

Buradan şu eşitsizlik zincirini kurarız:

$$(2x^2 + x)^2 < (2y)^2 \leq (2x^2 + x + 2)^2$$

$(2y)^2$ bir tamsayının karesidir ve $(2x^2+x)^2$ ile $(2x^2+x+2)^2$ arasında yer almaktadır. Bu iki sınır arasındaki yegâne tamsayı kareler şunlar olabilir:

1. $(2y)^2 = (2x^2 + x + 1)^2$
2. $(2y)^2 = (2x^2 + x + 2)^2$

Bu iki durumu ayrı ayrı inceleyelim:

**Durum 1:** $4y^2 = (2x^2 + x + 1)^2$

$$\begin{aligned} 4x^4 + 4x^3 + 4x^2 + 4x + 4 &= 4x^4 + 4x^3 + 5x^2 + 2x + 1 \\ 4x^2 + 4x + 4 &= 5x^2 + 2x + 1 \\ x^2 - 2x - 3 &= 0 \\ (x-3)(x+1) &= 0 \end{aligned}$$

Buradan iki kök gelir:

- $x = -1 \implies y^2 = 1 - 1 + 1 - 1 + 1 = 1 \implies y = \pm 1$.
- $x = 3 \implies y^2 = 81 + 27 + 9 + 3 + 1 = 121 \implies y = \pm 11$.

**Durum 2:** $4y^2 = (2x^2 + x + 2)^2$ Az önce gördüğümüz üzere bu eşitlik ancak aralarındaki fark sıfır olduğunda, yani $5x^2 = 0 \implies x = 0$ iken mümkündür:

- $x = 0 \implies y^2 = 1 \implies y = \pm 1$.

Tüm çözümleri toplarsak, denklemi sağlayan $(x, y)$ ikilileri:

$$(0,1),\ (0,-1),\ (-1,1),\ (-1,-1),\ (3,11),\ (3,-11)$$

olmak üzere 6 tanedir. □

## 34.3 İnvaryant, Uç Değer, Tersine Çalışma Alıştırmaları

Bölüm 17 (Parite), Bölüm 18 (İnvaryant) ve Bölüm 19'da (Uç Değer) incelediğimiz üzere, ayrık süreçlerde doğrudan sonuca gitmek yerine durum uzayının yapısal özelliklerine odaklanırız:

- **Değişmez (İnvaryant):** Sistem adımlarla değişse de sabit kalan bir büyüklük (çoğunlukla parite veya modüler toplam).
- **Yarı-Değişmez (Monovariant):** Her hamlede tek yönde kesin olarak artan ya da azalan ve alttan/üstten sınırlı olan bir nicelik; sürecin duracağını (terminasyon) kanıtlar.
- **Uç Değer İlkesi (Extremal Principle):** Kümenin en küçük/en büyük elemanına odaklanarak çelişki üretme veya yapıyı çözme yöntemi.
- **Tersine Çalışma (Working Backwards):** Özellikle oyun teorisinde hedef veya kayıp durumundan geriye doğru kazanan/kaybeden konumları etiketleme.

**Örnek 34.9** (Zorluk: Kolay - Değişmez)**.** *Bir tahtada* $1, 2, 3, \ldots, 2025$ *sayıları yazılıdır. Bir öğrenci tahtadan rastgele iki* $a$ *ve* $b$ *sayısını silip yerlerine* $|a - b|$ *farkını yazıyor. Bu işlem tahtada tek bir sayı kalana kadar tekrarlanıyor. Tahtada kalan son sayının* 0 *olamayacağını gösteriniz.*

*Çözüm.* İki sayı silinip yerine mutlak farkları yazıldığında toplamın paritesinin (Bölüm 17) nasıl değiştiğine bakalım. $a$ ve $b$ yerine $|a - b|$ yazıldığında, tahtadaki sayıların toplamı $S$ olsun. Değişim miktarı:

$$\Delta S = |a - b| - (a + b)$$

Mutlak değer tanımından bağımsız olarak, her $a, b$ tamsayısı için $|a - b| \equiv a - b \pmod 2$'dir. Dolayısıyla:

$$\Delta S \equiv (a - b) - (a + b) = -2b \equiv 0 \pmod 2$$

Yani her hamlede tahtadaki sayıların toplamının paritesi **değişmez** (invariant).

Her hamlede $\Delta S \equiv 0 \pmod 2$ olduğundan tahtadaki sayıların toplamının paritesi korunur. Benzer biçimde, tahtadaki tek sayıların adedi her hamlede ya 2 azalır ya da değişmez; çünkü iki tek sayının farkı çift, iki çift sayının farkı çift, bir tek ile bir çiftin farkı tektir. Dolayısıyla tek sayı adedinin paritesi (ve aynı şekilde toplamın paritesi) değişmez.

1'den 2025'e kadar olan tek sayıların adedi $(2025 + 1)/2 = 1013$ tanedir ve bu tek bir sayıdır. Başlangıç toplamı

$$S_0 = 1 + 2 + 3 + \cdots + 2025 = \frac{2025 \cdot 2026}{2} = 2025 \cdot 1013$$

olup tek sayıdır ($S_0 \equiv 1 \pmod 2$).

Parite bir değişmez olduğundan ve sonuçta tahtada tek bir sayı kalacağından bu sayı tek olmak zorundadır; 0 çift olduğu için son sayı asla 0 olamaz. □

**Örnek 34.10** (Zorluk: Orta - Tersine Çalışma / Kombinasyonel Oyun)**.** *İki oyuncu bir masa üzerindeki 50 adet madeni para ile bir oyun oynamaktadır. Sırası gelen oyuncu masadan 1, 2 veya 4 adet para alabilir (3 para almak yasaktır). Son parayı alan oyuncu oyunu kazanır. Her iki oyuncu da kusursuz oynarsa, ilk başlayan oyuncunun mu yoksa ikinci oyuncunun mu kazanma stratejisi vardır?*

*Çözüm.* Tersine çalışma yöntemiyle küçük $n$ değerleri için durumları **K** (Kazanma konumu - sırası gelen kazanır) ve **Y** (Yitirme/Kaybetme konumu - sırası gelen kaybeder) olarak adım adım etiketleyelim.

- 0 para: Hamle yapacak para kalmadığından, sırası gelen kaybetmiştir $\Longrightarrow$ **Y**.
- 1 para: 1 para alıp 0 bırakır $\Longrightarrow$ **K**.

- 2 para: 2 para alıp 0 bırakır $\implies$ **K**.
- 3 para: Yapılabilecek hamleler 1 veya 2 almaktır. 1 alırsa 2 bırakır (K), 2 alırsa 1 bırakır (K). Her iki durumda da rakibe K konumu kalır. Rakibine sadece K bırakabilen konum Y'dir $\implies$ **Y**.
- 4 para: 4 para alıp 0 bırakır $\implies$ **K**.
- 5 para: 2 para alıp 3 bırakabilir (3 bir Y konumudur!). Rakibe Y bırakan konum K'dir $\implies$ **K**.
- 6 para:
  - 1 alırsa 5 kalır (K)
  - 2 alırsa 4 kalır (K)
  - 4 alırsa 2 kalır (K)

  Tüm hamleler rakibe K bıraktığı için 6 $\implies$ **Y** konumudur!

Oluşan deseni özetleyelim:

| $n$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Durum | Y | K | K | Y | K | K | Y | K |

Desen açıkça görülmektedir: Konum ancak ve ancak $n \equiv 0 \pmod 3$ olduğunda bir Kaybetme (**Y**) konumudur; aksi halde Kazanma (**K**) konumudur.

Bunu tümevarımla kanıtlayalım:

- Eğer $n \equiv 0 \pmod 3$ ise: Yapılabilecek hamleler 1, 2 veya 4 almaktır. $n-1 \equiv 2 \pmod 3$, $n-2 \equiv 1 \pmod 3$, $n-4 \equiv 2 \pmod 3$. Görüldüğü gibi kalan paraların hiçbiri 3'ün katı olamaz. Yani Y konumundan yapılan her hamle rakibe zorunlu olarak bir K konumu verir.
- Eğer $n \not\equiv 0 \pmod 3$ ise:
  - $n \equiv 1 \pmod 3$ ise oyuncu 1 veya 4 para alarak kalanı 3'ün katı yapabilir ($n-1 \equiv 0$ veya $n-4 \equiv 0$).
  - $n \equiv 2 \pmod 3$ ise oyuncu 2 para alarak kalanı 3'ün katı yapabilir ($n-2 \equiv 0$).

  Her iki durumda da oyuncu rakibine bir Y konumu bırakabilir.

Başlangıçtaki para sayısı $n = 50$'dir.

$$50 = 3 \cdot 16 + 2 \implies 50 \equiv 2 \pmod 3$$

$50 \not\equiv 0 \pmod 3$ olduğundan 50 bir **Kazanma (K)** konumudur.

Dolayısıyla **birinci oyuncunun** kesin kazanma stratejisi vardır. İlk oyuncu ilk hamlede 2 para alarak masada 48 (3'ün katı) para bırakmalı ve sonrasında ikinci oyuncu ne yaparsa yapsın onun aldığı para sayısını 3'e tamamlayan hamleler yapmalıdır. □

**Örnek 34.11** (Zorluk: İleri Düzey - Uç Değer İlkesi)**.** *Bir ülkede sonlu sayıda şehir vardır ve bazı şehir çiftleri arasında tek yönlü yollar bulunmaktadır. Herhangi iki şehir arasında en fazla bir tek yönlü yol vardır ve hiçbir iki şehir arasında karşılıklı yol yoktur. Herhangi bir şehirden çıkan yol sayısı ile o şehre giren yol sayısı birbirine eşittir. Bu ülkede bir şehirden başlayıp yolların yönünü takip ederek tekrar aynı şehre dönen bir kapalı rotanın (yönlü döngünün) var olduğunu kanıtlayınız.*

*Çözüm.* Şehirler ve yollar yönlü bir graf ($G = (V, E)$) oluşturur. Bölüm 19'da ele aldığımız uç değer ilkesini (extremal principle) kullanmanın doğal bir yolu, graftaki en uzun basit yönlü yolu seçmektir.

Şehirler kümesi sonlu olduğundan, grafta kendisini tekrar etmeyen (aynı şehirden iki kez geçmeyen) en uzun yönlü yolu ele alabiliriz. Bu en uzun yol:

$$P = (v_1 \to v_2 \to v_3 \to \cdots \to v_k)$$

olsun.

Şimdi bu yolun sonlandığı $v_k$ şehrine odaklanalım:

- Soruda verilen koşula göre her şehrin çıkış derecesi giriş derecesine eşittir. $v_k$ şehrine yol boyunca $v_{k-1}$'den en az bir giriş yapılmıştır ($d_{in}(v_k) \geq 1$).
- Dolayısıyla $d_{out}(v_k) = d_{in}(v_k) \geq 1$ olmalıdır; yani $v_k$ şehrinden çıkan en az bir yol bulunmalıdır.
- $v_k$'den çıkan bir yolun hedefi olan şehre $u$ diyelim ($v_k \to u$).

$u$ şehri nerede olabilir?

1. Eğer $u$, $P$ yolunun üzerinde *olmayan* bir şehir olsaydı: $P$ yolunu $v_k \to u$ kenarı ile uzatarak $v_1 \to \cdots \to v_k \to u$ yolunu elde ederdik. Bu yolun uzunluğu $k$ kenar olurdu. Fakat biz $P$'yi **en uzun yol** olarak seçmiştik! Bu bir çelişkidir.
2. O halde $u$ şehri kesinlikle $P$ yolunun üzerinde daha önce ziyaret edilmiş bir şehir olmalıdır:
$$u \in \{v_1, v_2, \ldots, v_{k-1}\}$$

Demek ki bir $1 \leq i \leq k-1$ için $u = v_i$'dir. Bu durumda:

$$v_i \to v_{i+1} \to \cdots \to v_k \to v_i$$

rotası, yönleri takip ederek başladığı $v_i$ şehrine geri dönen bir yönlü döngü oluşturur. İspat tamamlanır. □

**Örnek 34.12** (Zorluk: Olimpiyat - Yarı-Değişmez (Monovariant))**.** *Düzlemde sonlu sayıda $n$ adet nokta işaretlenmiştir. Bazı nokta çiftleri doğru parçalarıyla birleştirilmiştir. Her adımda kesişen iki doğru parçası $AB$ ve $CD$ seçilmekte; bu iki parça silinip yerlerine $AC$ ve $BD$ parçaları çizilmektedir. Bu sürecin sonlu sayıda adım sonra zorunlu olarak duracağını ispatlayınız.*

*Çözüm.* Bir sürecin duracağını (terminasyonunu) kanıtlamak için adımlar boyunca tek yönde kesin olarak değişen (artan ya da azalan) ve alttan veya üstten sınırlı olan bir büyüklük aramalıyız. Bu büyüklüğe Bölüm 18'de ele aldığımız üzere **yarı-değişmez (monovariant)** denir.

Burada doğru parçalarının uzunlukları toplamını takip etmek doğal bir yaklaşımdır. Mevcut doğru parçalarının uzunlukları toplamını $L$ ile gösterelim:

$$L = \sum_{e \in E} |e|$$

Bir adımda kesişen $AB$ ve $CD$ parçaları seçilsin. Bu parçaların kesişim noktasına $X$ diyelim.

Üçgen eşitsizliğini kullanalım: $X$ noktası hem $AB$ hem de $CD$ üzerinde yer aldığından:

$$|AB| = |AX| + |XB| \quad \text{ve} \quad |CD| = |CX| + |XD|$$

Silinen iki parçanın uzunlukları toplamı:

$$|AB| + |CD| = |AX| + |XB| + |CX| + |XD| = (|AX| + |CX|) + (|XB| + |XD|)$$

Şimdi yeni çizilen parçalara bakalım: $AC$ ve $BD$. $AXC$ ve $BXD$ üçgenlerinde (noktalar doğrusal değilse) kesin üçgen eşitsizliği geçerlidir:

$$|AX| + |XC| > |AC| \quad \text{ve} \quad |BX| + |XD| > |BD|$$

(Noktalar doğrusal olsa bile kesişen farklı iki doğru parçası durumunda toplam uzunluk kesinlikle azalır).

Bu iki eşitsizliği taraf tarafa toplarsak:

$$|AC| + |BD| < (|AX| + |CX|) + (|XB| + |XD|) = |AB| + |CD|$$

Görüldüğü gibi, her hamlede silinen iki parçanın yerine gelen iki yeni parçanın uzunlukları toplamı **kesin olarak daha küçüktür**!

Dolayısıyla tüm doğru parçalarının uzunlukları toplamı olan $L$, her adımda kesin olarak azalır:

$$L_{yeni} < L_{eski}$$

Peki bu azalma sonsuza kadar sürebilir mi? $n$ nokta arasından seçilebilecek olası tüm doğru parçalarının kümesi sonludur ($\binom{n}{2}$ adet parça vardır). Dolayısıyla bu parçalardan oluşturulabilecek olası parça konfigürasyonlarının sayısı da sonludur (en fazla $2^{\binom{n}{2}}$ adet).

Sonlu sayıda durum olduğundan, $L$ toplamının alabileceği farklı değerlerin sayısı da sonludur. Bir büyüklük sonlu bir küme içinde her adımda kesin olarak azalıyorsa sonsuza kadar devam edemez; eninde sonunda minimum değerine ulaşmak ve durmak zorundadır.

O halde bu kesişim giderme süreci kesinlikle sonlu adımda sonlanır. □

Aşağıdaki C programı, monovariant fikrini pekiştirmek için 1D çubuk üzerindeki parçacıkların çarpışma ve yön değiştirme sürecindeki yarı-değişmezi simüle eder:

```
#include <stdio.h>

/* Parcaciklarin carpismasinda enerjinin (veya monovariantin)
   azalmasini takip eden ornek fonksiyon */
int simulate_step(int arr[], int n) {
    int modified = 0;
    for (int i = 0; i < n - 1; i++) {
        /* Eger ters yonde hareket eden iki komsu varsa yer degistir */
        if (arr[i] == 1 && arr[i+1] == -1) {
            arr[i] = -1;
            arr[i+1] = 1;
            modified = 1;
            i++; /* Bu cifti atla */
        }
    }
    return modified; /* Degisiklik yapildiysa surec surer */
}

int main() {
    int particles[] = {1, 1, -1, 1, -1, -1};
    int n = 6;
    int steps = 0;

    /* Inversiyon sayisi her adimda azalir (monovariant) */
    while (simulate_step(particles, n)) {
        steps++;
    }

    printf("Surec %d adimda sonlandi.\n", steps);
    return 0;
}
```

## Bölüm Özeti ve Problem Çözme Rehberi

Bu bölümde ele aldığımız soru tiplerinde başarıya ulaşmak için şu kontrol listesini aklınızda tutunuz:

1. **Modül Seçimi:** Kareler için mod 4 ve mod 8; küpler için mod 7 ve mod 9; genel asallık kısıtları için $\pmod 3$ ilk duraklarınız olmalıdır.

2. **En Küçük Asal Bölen:** $n \mid a^n \pm 1$ gibi üstel bölünebilmelerde $n$'nin en küçük asal bölenine $p$ deyip $\mathrm{ord}_p(a)$ incelemek en güçlü araçtır.

3. **Sıkıştırma (Sandviç) Yöntemi:** Polinom biçimindeki tam sayı denklemlerinde cebirsel ifadeyi ardışık iki tam kare arasına sıkıştırarak bilinmeyenleri küçük bir aralıkta yakalayabilirsiniz.

4. **Değişmez ve Yarı-Değişmez:** Adımlı oyunlarda veya simülasyonlarda "Ne sabit kalıyor?" (parite, kalan) ve "Ne tek yönde değişiyor?" (toplam uzunluk, inversiyon sayısı) sorularını ilk saniyeden itibaren sorun.

# Bölüm 35

# Alıştırmalar: Algoritma ve C

TÜBİTAK 1. Aşama Bilgisayar Olimpiyatı sınavında soruların önemli bir bölümü, algoritma analizi ve C dilinin derinliklerine dair doğrudan kod okuma becerisini ölçer. Bu sınavlarda derleyici ya da hata ayıklayıcı (debugger) bulunmaz; zihniniz hem bir C derleyicisi hem de bir işlemci gibi çalışmak zorundadır. Karmaşıklık analizinde gizli kalan adımları sezmek, işaretçilerin bellekteki hareketini bir bellek haritası üzerinde hatasız izlemek ve verilen soyut bir matematiksel problemi birkaç satırlık sade bir koda dönüştürebilmek, olimpiyat başarısının temel taşlarıdır.

Önceki bölümlerde ele aldığımız asimptotik analiz (Bölüm 21), C dilinin bellek modeli, işaretçi aritmetiği (Bölüm 31) ve temel algoritma kalıplarını bu alıştırmalarda çözümlü örneklerle pekiştireceğiz. Her soruda önce problemin çözümünde izlenecek yaklaşımı ele alacak, ardından adım adım çözüme geçeceğiz.

## 35.1 Karmaşıklık Okuma Alıştırmaları

Algoritma analizinde en sık düşülen tuzak, iç içe geçmiş döngülerin sayısını sayıp doğrudan $O(n^2)$ veya $O(n^3)$ hükmüne varmaktır. Oysa döngü değişkenlerinin birbirine bağımlı olduğu, adım büyüklüklerinin sabit kalmadığı veya döngü gövdesinin her adımda farklı maliyetle çalıştığı durumlarda bu kaba yaklaşım geçerliliğini yitirir. Karmaşıklığı doğru okumak; algoritmik süreci matematiksel bir toplama ($\sum$) dönüştürme ve bu toplamın asimptotik büyüme hızını doğru belirleme becerisidir.

**Tanım 35.1** (Baskın Terim ve Asimptotik Denk Büyüklük)**.** *Pozitif reel diziler $f(n)$ ve $g(n)$ için, $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$ $(0 < c < \infty)$ ise $f(n) = \Theta(g(n))$ denir. Eğer $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ ise $f(n) = o(g(n))$ olur ve asimptotik olarak $g(n)$ fonksiyonunun $f(n)$'ye baskın olduğu söylenir.*

Karmaşıklık analizi yaparken şu üç temel araçtan sıklıkla yararlanırız:

1. **Harmonik Toplam:** $\sum_{i=1}^{n} \frac{1}{i} = \ln n + \gamma + O(1/n) = \Theta(\log n)$.

2. **Geometrik Toplam:** $r > 1$ için $\sum_{i=0}^{k} r^i = \Theta(r^k)$; $r < 1$ için $\sum_{i=0}^{\infty} r^i = \Theta(1)$.

3. **İntegral ile Yaklaşım:** Monoton artan veya azalan $f(x)$ fonksiyonları için $\sum_{i=a}^{b} f(i) = \Theta\left(\int_a^b f(x)\,dx\right)$.

**Örnek 35.2.** *Aşağıda verilen C kod parçacığının zaman karmaşıklığını Big O cinsinden belirleyiniz:*

```c
int sayac = 0;
for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= n; j += i) {
        sayac++;
    }
}
```

*Çözüm.* Dış döngü $i = 1, 2, \ldots, n$ değerlerini alır. İç döngü ise 1'den $n$'ye kadar sabit bir adımla değil, $i$'şer $i$'şer artarak ilerler. Dolayısıyla iç döngü her çalıştığında adım sayısı $n$ değil, doğrudan $i$'ye bağlıdır. İç döngünün kaç kez döndüğünü $i$ cinsinden ifade edip tüm $i$ değerleri üzerinden toplamalıyız.

İç döngü değişkeni $j$; $1, 1+i, 1+2i, \ldots$ değerlerini alır ve $j \le n$ olduğu sürece çalışır. Bu döngüdeki adım sayısı:

$$\left\lceil \frac{n}{i} \right\rceil$$

kadardır. O halde toplam işlem sayısı $T(n)$:

$$T(n) = \sum_{i=1}^{n} \left\lceil \frac{n}{i} \right\rceil$$

Burada tavan fonksiyonunu açarsak:

$$\sum_{i=1}^{n} \frac{n}{i} \le T(n) < \sum_{i=1}^{n} \left(\frac{n}{i} + 1\right) = n \sum_{i=1}^{n} \frac{1}{i} + n$$

Harmonik seri toplamının $\sum_{i=1}^{n} \frac{1}{i} = \ln n + \Theta(1)$ olduğunu bildiğimize göre:

$$T(n) = n \cdot (\ln n + \Theta(1)) + O(n) = n \ln n + O(n) = \Theta(n \log n)$$

Kod parçasının zaman karmaşıklığı $O(n \log n)$ olur. □

**Örnek 35.3.** *Aşağıdaki fonksiyonun $T(n)$ zaman karmaşıklığını bulunuz:*

```c
void gizemli(int n) {
    int i = 1;
    while (i <= n) {
        for (int j = 1; j <= i; j++) {
            // O(1) maliyetli islem
```

```
        }
        i = i * 2;
    }
}
```

*Çözüm.* İç döngü $i$ adım çalışmaktadır. Ancak $i$ her adımda 1 artmaz; 2 ile çarpılarak katlanır. Bu durum bir geometrik dizi oluşturur. Dış döngünün kaç kez çalışacağını bulup geometrik serinin toplamını hesaplayalım.

Dış döngüde $i$ değişkeninin aldığı değerler kümesi:

$$\{1, 2, 4, 8, \ldots, 2^k\}$$

burada $2^k \leq n < 2^{k+1}$, yani $k = \lfloor \log_2 n \rfloor$'dir.

Her bir $i = 2^m$ ($m = 0, 1, \ldots, k$) değeri için içteki döngü tam olarak $i = 2^m$ kez çalışır. Toplam işlem sayısı:

$$T(n) = \sum_{m=0}^{k} 2^m = 2^0 + 2^1 + 2^2 + \cdots + 2^k = 2^{k+1} - 1$$

$k = \lfloor \log_2 n \rfloor$ olduğuna göre:

$$2^{k+1} = 2 \cdot 2^{\lfloor \log_2 n \rfloor} \leq 2n$$

ve $2^{k+1} > n$'dir. Dolayısıyla $T(n) = \Theta(n)$ bulunur.

Dış döngü $\log n$ adım, iç döngü ise en fazla $n$ adım çalıştığı için kaba bir üst sınırla $O(n \log n)$ denebilirdi; fakat geometrik seri toplamı en büyük terimin mertebesinde kaldığından gerçek asimptotik karmaşıklık $\Theta(n)$'dir. □

**Örnek 35.4.** *Aşağıda verilen rekürsif (rekürsif) fonksiyonun zaman karmaşıklığını belirleyiniz:*

$$T(n) = 3T(n/4) + n \log_2 n$$

*Çözüm.* Bu bağıntı $T(n) = aT(n/b) + f(n)$ biçimindedir; dolayısıyla Master Teoremi'ni doğrudan uygulayabiliriz. Parametreler: $a = 3$, $b = 4$ ve $f(n) = n \log_2 n$.

Kritik üssü hesaplayalım:

$$c_{\text{kritik}} = \log_b a = \log_4 3 \approx 0{,}793$$

Şimdi $f(n)$ fonksiyonunu $n^{\log_b a} = n^{\log_4 3}$ ile kıyaslayalım. $f(n) = n \log_2 n$ fonksiyonu, $n^1$'den hızlı büyümektedir. $1 > \log_4 3$ olduğundan $\epsilon = 1 - \log_4 3 > 0$ seçilebilir. Bu durumda:

$$f(n) = \Omega\left(n^{\log_4 3 + \epsilon}\right)$$

koşulu sağlanır. Master Teoremi'nin 3. durumunun geçerli olabilmesi için düzenlilik (regularity) koşulunu da kontrol etmeliyiz: Yeterince büyük $n$ değerleri ve bir $c < 1$ sabiti için $af(n/b) \leq cf(n)$ olmalıdır:

$$af(n/b) = 3 \cdot \left(\frac{n}{4}\log_2\left(\frac{n}{4}\right)\right) = \frac{3}{4}n(\log_2 n - 2) \leq \frac{3}{4}n\log_2 n = \frac{3}{4}f(n)$$

$c = 3/4 < 1$ seçildiğinde düzenlilik koşulu tüm $n \geq 4$ için sağlanır.

Master Teoremi'nin 3. durumu gereğince toplam maliyeti baskın terim olan $f(n)$ belirler:

$$T(n) = \Theta(f(n)) = \Theta(n\log n)$$

□

**Örnek 35.5.** *Aşağıdaki fonksiyonların asimptotik büyüme hızlarını küçükten büyüğe sıralayınız:*

$$f_1(n) = n^{\log_2\log_2 n}, \quad f_2(n) = (\log_2 n)^{\log_2 n}, \quad f_3(n) = 2^{\sqrt{\log_2 n}}, \quad f_4(n) = n!$$

*Çözüm.* Farklı taban ve üslere sahip fonksiyonları karşılaştırmanın en güvenli yolu, her birinin 2 tabanında logaritmasını alarak büyüme hızlarını incelemektir:

1. $f_1(n) = n^{\log_2\log_2 n} \implies \log_2 f_1(n) = (\log_2\log_2 n)\cdot(\log_2 n)$.
2. $f_2(n) = (\log_2 n)^{\log_2 n} \implies \log_2 f_2(n) = (\log_2 n)\cdot\log_2(\log_2 n)$. Görüldüğü üzere $\log_2 f_1(n) = \log_2 f_2(n)$, dolayısıyla $f_1(n) = f_2(n)$'dir.
3. $f_3(n) = 2^{\sqrt{\log_2 n}} \implies \log_2 f_3(n) = \sqrt{\log_2 n}$.
4. $f_4(n) = n! \implies$ Stirling yaklaşımı ile $\log_2(n!) = \Theta(n\log_2 n)$.

Şimdi bu logaritmaların büyüme hızlarını kıyaslayalım: $\sqrt{\log_2 n}$ değeri, $(\log_2 n)(\log_2\log_2 n)$'den çok daha yavaş büyür. Bu ikinci ifade ise $n\log_2 n$'den çok daha yavaş büyür ($n$ terimi $\log_2\log_2 n$'e baskındır).

O halde asimptotik büyüme sıralaması:

$$f_3(n) \prec f_1(n) = f_2(n) \prec f_4(n)$$

şeklinde elde edilir. □

**Tanım 35.6** (Sık Yapılan Hata: Logaritma Tabanları ve Üstel İfadeler)**.** *$O(\log_2 n) = O(\log_{10} n)$ eşitliği doğrudur çünkü taban değişimi sabit bir çarpan getirir ($\log_a n = \frac{\log_b n}{\log_b a}$). Ancak üstel fonksiyonlarda taban ihmal **edilemez**:*

$$2^n \neq O(3^n) \quad ve \quad 2^{2n} = 4^n \neq O(2^n)$$

*Benzer biçimde $O(n)$ süren bir işlemi $O(\log n)$ kez çağırmak $O(n\log n)$ maliyet üretir; fakat iç döngünün değişken adımları toplamı her zaman döngü derinliklerinin çarpımı anlamına gelmez.*

## 35.2 İz Sürme ve Çıktı Bulma Alıştırmaları

TÜBİTAK 1. Aşama sınavında en sık karşılaşılan soru tiplerinden biri, kısa bir C programının ekran çıktısını belirlemektir. Bu sorular işlem önceliği, yan etkiler (side effects), işaretçi-dizi ilişkisi, statik değişkenlerin ömrü ve rekürsiyon çağrı yığını (call stack) mekanizmalarını sınamak üzere kurgulanır.

Bu tür sorularda tahmin yürütmek yerine kâğıt üzerinde bir **iz sürme çizelgesi** (trace table) ve **rekürsiyon ağacı** oluşturmak hata payını ortadan kaldırır.

**Örnek 35.7.** *Aşağıdaki C programı çalıştırıldığında ekrana ne yazar?*

```c
#include <stdio.h>

int f(int *p, int n) {
    if (n <= 0) return 0;
    return *p + f(p + 1, n - 2);
}

int main() {
    int a[] = {3, 7, 2, 9, 5, 1, 8};
    printf("%d\n", f(a, 7));
    return 0;
}
```

*Çözüm.* Fonksiyon bir işaretçi (`int *p`) ve bir tamsayı (`n`) almaktadır (Bölüm 31'de incelediğimiz işaretçi aritmetiğini anımsayınız). Her rekürsif çağrıda işaretçi 1 birim ileri kaydırılmakta (`p + 1`), fakat $n$ parametresi 2 azaltılmaktadır (`n - 2`). Fonksiyonun her adımda hangi dizi elemanına baktığını ve durma koşulunu listeleyelim.

Dizinin elemanları ve indeksleri:

$$a[0] = 3, \quad a[1] = 7, \quad a[2] = 2, \quad a[3] = 9, \quad a[4] = 5, \quad a[5] = 1, \quad a[6] = 8$$

Başlangıç çağrısı: $f(\&a[0], 7)$

1. $f(\&a[0], 7)$: $n = 7 > 0$. Döndürülen: $*a[0] + f(\&a[1], 5) = 3 + f(\&a[1], 5)$.
2. $f(\&a[1], 5)$: $n = 5 > 0$. Döndürülen: $*a[1] + f(\&a[2], 3) = 7 + f(\&a[2], 3)$.
3. $f(\&a[2], 3)$: $n = 3 > 0$. Döndürülen: $*a[2] + f(\&a[3], 1) = 2 + f(\&a[3], 1)$.
4. $f(\&a[3], 1)$: $n = 1 > 0$. Döndürülen: $*a[3] + f(\&a[4], -1) = 9 + f(\&a[4], -1)$.
5. $f(\&a[4], -1)$: $n = -1 \leq 0$. Taban durumu devreye girer ve 0 döndürür.

Değerleri geriye doğru toplayalım:

$$\text{Toplam} = 3 + 7 + 2 + 9 + 0 = 21$$

Program ekrana **21** yazar. □

**Örnek 35.8.** *Statik değişken içeren aşağıdaki programın çıktısını bulunuz:*

```
#include <stdio.h>

int gizem(int n) {
    static int x = 1;
    if (n <= 1) return x;
    x = x + n;
    return gizem(n - 1) + x;
}

int main() {
    printf("%d\n", gizem(4));
    return 0;
}
```

*Çözüm.* `static int x = 1;` tanımı yalnızca program başladığında bir defaya mahsus işletilir. Sonraki tüm çağrılarda $x$ değişkeni bellekteki aynı adresi korur; bir çağrıda $x$'e yapılan değişiklik diğer çağrıları da bağlar. İfade `gizem(n - 1) + x` biçimindedir.

Çağrı adımlarını sıra ile takip edelim:

- $n = 4$: $x = 1 + 4 = 5$ olur. Hesaplanan ifade: $E_4 = \text{gizem}(3) + x$.
- $n = 3$: $x = 5 + 3 = 8$ olur. Hesaplanan ifade: $E_3 = \text{gizem}(2) + x$.
- $n = 2$: $x = 8 + 2 = 10$ olur. Hesaplanan ifade: $E_2 = \text{gizem}(1) + x$.
- $n = 1$: $n \leq 1$ sağlandığından taban durumu çalışır ve o anki $x$ değeri döndürülür. Şu an $x = 10$'dur; dolayısıyla $\text{gizem}(1) = 10$ döner.

Rekürsiyon yığınından geri dönüşleri izleyelim. $x$ statik bir değişken olduğundan değeri çağrılar tamamlanırken değişmez, hep 10 kalır:

$$\begin{aligned}\text{gizem}(2) &= \text{gizem}(1) + x = 10 + 10 = 20\\ \text{gizem}(3) &= \text{gizem}(2) + x = 20 + 10 = 30\\ \text{gizem}(4) &= \text{gizem}(3) + x = 30 + 10 = 40\end{aligned}$$

Program ekrana **40** yazar. □

**Örnek 35.9.** *Aşağıdaki işaretçi ve dizi aritmetiği içeren programın çıktısını belirleyiniz:*

```
#include <stdio.h>

int main() {
    int dizi[] = {10, 20, 30, 40, 50, 60};
    int *p = dizi;
```

```
    int a = *p++;
    int b = *++p;
    int c = ++*p;

    printf("%d %d %d %d\n", a, b, c, *p);
    return 0;
}
```

*Çözüm.* Burada C dilinin operatör önceliği ve birleşme yönü (associativity) kuralları sınanmaktadır:

- `*p++`: `++` soneki `*` işaretçi operatöründen daha yüksek önceliğe sahiptir; fakat sonek artırma, ifadenin değeri okunduktan *sonra* gerçekleşir. Dolayısıyla ifadenin değeri `*p` olur, ardından `p` sonraki elemana geçer.
- `*++p`: Önek `++` ile `*` aynı önceliktedir ve sağdan sola birleşirler. Önce `p` bir ileri kaydırılır, ardından ulaşılan adresteki değer okunur.
- `++*p`: Sağdan sola birleşir. Önce `*p` okunur, ardından o adresteki değer 1 artırılır.

Adım adım izleyelim:

1. Başlangıçta $p$ dizinin başına işaret eder: $p \to$ &dizi[0] (değeri 10).
2. `int a = *p++;`: İfade değeri $p$'nin gösterdiği yerdeki değerdir, yani $a = 10$. Ardından $p$ bir eleman ilerler: $p \to$ &dizi[1] (değeri 20).
3. `int b = *++p;`: Önce $p$ ilerletilir: $p \to$ &dizi[2] (değeri 30). Ardından işaretçi çözülür: $b = 30$.
4. `int c = ++*p;`: $p$ hâlâ &dizi[2] adresindedir ve oradaki değer 30'dur. Bu değer 1 artırılır, dizi[2] elemanı 31 olur ve $c = 31$ değerini alır.
5. Son olarak $*p$ değeri okunur; $p$ hâlâ &dizi[2] adresinde olduğundan $*p = 31$'dir.

Ekrana yazılan çıktı: **10 30 31 31** olur. □

**Örnek 35.10.** *Aşağıdaki bitwise (bitwise) işlem yapan fonksiyonun çıktısını bulunuz:*

```
#include <stdio.h>

int islem(int n) {
    int c = 0;
    while (n > 0) {
```

```
        c++;
        n = n & (n - 1);
    }
    return c;
}

int main() {
    printf("%d\n", islem(157));
    return 0;
}
```

*Çözüm.* `n & (n - 1)` ifadesi bilgisayar bilimlerinde Brian Kernighan algoritması olarak bilinen temel bir kalıptır (Bölüm 32'de incelediğimiz bit işlemlerini hatırlayınız). Bir sayının ikili (binary) açılımındaki en sağdaki (en düşük basamaklı) 1 bitini sıfırlar. Fonksiyon, sayı 0 olana kadar kaç defa 1 bitinin temizlendiğini sayarak ikili açılımdaki 1'lerin toplam adedini (popcount) bulur.

157 sayısını Bölüm 15'teki taban dönüştürme yöntemiyle ikili tabana çevirelim:

$$157 = 128 + 16 + 8 + 4 + 1 = 2^7 + 2^4 + 2^3 + 2^2 + 2^0$$

İkili gösterimi:

$$(157)_{10} = (10011101)_2$$

Bu sayıda tam 5 adet 1 biti bulunmaktadır.

Döngü adımları şöyle ilerler:

1. $157 = (10011101)_2 \implies n \;\&\; (n-1) = (10011100)_2 = 156$, $c = 1$.
2. $156 = (10011100)_2 \implies n \;\&\; (n-1) = (10011000)_2 = 152$, $c = 2$.
3. $152 = (10011000)_2 \implies n \;\&\; (n-1) = (10010000)_2 = 144$, $c = 3$.
4. $144 = (10010000)_2 \implies n \;\&\; (n-1) = (10000000)_2 = 128$, $c = 4$.
5. $128 = (10000000)_2 \implies n \;\&\; (n-1) = (00000000)_2 = 0$, $c = 5$.

Döngü sonlanır ve $c = 5$ değeri döndürülür. Programın çıktısı **5**'tir. □

**Tanım 35.11** (Sık Yapılan Hata: Kısa Devre Değerlendirmesi (Short-Circuit))**.** *Mantıksal* `&&` *ve* `||` *operatörlerinde C dili kısa devre değerlendirmesi yapar:*

- `A && B` *ifadesinde eğer A yanlış (0) ise, B ifadesi* ***asla çalıştırılmaz****.*
- `A || B` *ifadesinde eğer A doğru (sıfırdan farklı) ise, B ifadesi* ***asla çalıştırılmaz****.*

*Örneğin* `(x == 0) || (y++ > 0)` *ifadesinde* `x == 0` *doğruysa* `y++` *yan etkisi gerçekleşmez ve* `y` *artmaz. Kod izleme sorularında bu kurala özellikle dikkat edilmelidir.*

## 35.3 Küçük C Kodu Alıştırmaları

Olimpiyatta algoritma tasarlarken teorik yaklaşımı en az satırla, taşma (overflow) yaşatmadan ve uç durumları doğru yöneterek koda aktarabilmek gerekir. Bu bölümde temel tekniklerin C dilindeki uygulamalarını inceleyeceğiz.

### 35.3.1 Hızlı Üs Alma (Binary Exponentiation)

$a^b \pmod m$ değerini basit bir döngüyle hesaplamak $O(b)$ adım sürer ve $b \approx 10^{18}$ olduğunda pratik olarak imkânsızdır. Bölüm 14'te ele aldığımız modüler aritmetik özellikleri sayesinde ikili üs alma yöntemi bu süreyi $O(\log b)$ adıma indirir.

**Düşünsel Yaklaşım (İnvaryant İlkesi, bkz. Bölüm 18):** $a^b$ ifadesini hesaplarken şu değişmezi koruruz:

$$\text{sonuc} \cdot a_{\text{guncel}}^{b_{\text{guncel}}} \equiv a_{\text{ilk}}^{b_{\text{ilk}}} \pmod m$$

Eğer $b$ tek sayı ise üssü bir azaltıp tabanı sonuca çarparız ($b \leftarrow b-1$). Eğer $b$ çift sayı ise üssü yarıya indirip tabanın karesini alırız ($b \leftarrow b/2, a \leftarrow a^2$).

**Örnek 35.12.** *Büyük üsler için $a^b \pmod m$ değerini $O(\log b)$ sürede hesaplayan taşmasız C fonksiyonunu yazınız.*

*Çözüm.* İki 64-bitlik tamsayının çarpımı geçici olarak $2^{64}$ sınırını aşabileceğinden, ara çarpımlarda `unsigned long long` veya `long long` türü dikkatle kullanılmalıdır.

```c
#include <stdio.h>

long long mod_us(long long taban, long long us, long long mod) {
    long long sonuc = 1;
    taban %= mod; // Taban moddan buyukse kucult

    while (us > 0) {
        // Us tek sayi ise tabani sonuca dahil et
        if (us & 1) {
            sonuc = (sonuc * taban) % mod;
        }
        // Tabanin karesini al ve ussu yariya indir
        taban = (taban * taban) % mod;
        us >>= 1; // us = us / 2
    }
    return sonuc;
}
```

Aynı algoritmanın, üs bitini bit işlemleri yerine aritmetik operatörlerle sınayan eşdeğer bir C gerçeklemesi de şöyledir:

```c
long long mod_pow(long long base, long long exp, long long mod) {
    long long result = 1;
    base %= mod;
    while (exp > 0) {
        if (exp % 2 == 1) {
            result = (result * base) % mod;
        }
        base = (base * base) % mod;
        exp /= 2;
    }
    return result;
}
```

□

### 35.3.2 Diziyi $O(1)$ Ekstra Bellek ile Döndürme (Reversal Algorithm)

$n$ elemanlı bir diziyi $k$ pozisyon sağa dairesel olarak kaydırmak istiyoruz. Ekstra dizi açmadan ($O(1)$ ek bellek) ve her elemana tam iki kez dokunarak $O(n)$ sürede bu işlemi gerçekleştirebiliriz.

**Düşünsel Yaklaşım:** Diziyi $A$ (ilk $n-k$ eleman) ve $B$ (son $k$ eleman) olarak iki parçaya ayıralım:

$$D = [A \mid B]$$

Bizden istenen $[B \mid A]$ dizisidir. Ters çevirme işlemini $(\cdot)^R$ ile gösterelim. Matris cebirindeki $(AB)^{-1} = B^{-1}A^{-1}$ kuralına benzer biçimde:

$$(A^R B^R)^R = (B^R)^R (A^R)^R = BA$$

Bu özdeşlik aradığımız algoritmayı doğrudan verir:

1. İlk $n-k$ elemanı kendi içinde ters çevir ($A \to A^R$).
2. Son $k$ elemanı kendi içinde ters çevir ($B \to B^R$).
3. Bütün diziyi baştan sona ters çevir ($(A^R B^R) \to BA$).

**Örnek 35.13.** *Bir tamsayı dizisini yerinde (in-place) $k$ adım sağa kaydıran C kodunu yazınız.*

*Çözüm.*

```c
#include <stdio.h>

void ters_cevir(int *dizi, int bas, int son) {
    while (bas < son) {
        int gecici = dizi[bas];
        dizi[bas] = dizi[son];
```

```
        dizi[son] = gecici;
        bas++;
        son--;
    }
}

void saga_dondur(int *dizi, int n, int k) {
    if (n <= 1) return;
    k = k % n; // k >= n durumunu onle
    if (k == 0) return;

    // 1. Ilk n - k elemani cevir
    ters_cevir(dizi, 0, n - k - 1);
    // 2. Son k elemani cevir
    ters_cevir(dizi, n - k, n - 1);
    // 3. Tum diziyi cevir
    ters_cevir(dizi, 0, n - 1);
}
```

Toplam işlem sayısı $(n-k)+k+n = 2n$ yer değiştirmedir. Zaman karmaşıklığı $O(n)$, ek bellek karmaşıklığı $O(1)$'dir. □

### 35.3.3 Hollanda Bayrağı Algoritması (3-Yollu Bölümleme)

Dijkstra tarafından önerilen bu klasik problemde; yalnızca 0, 1 ve 2 değerlerinden oluşan bir diziyi tek bir geçişte ($O(n)$ zaman ve $O(1)$ ek bellek) sıralamamız istenir.

**Düşünsel Yaklaşım (Döngü Değişmezi):** Diziyi dört bölgeye ayıracak üç işaretçi (düşük, orta, yüksek) tanımlarız:

- $[0, \text{düşük} - 1]$: Kesinlikle 0'lar bölgesi.
- $[\text{düşük}, \textit{orta} - 1]$: Kesinlikle 1'ler bölgesi.
- $[\textit{orta}, \text{yüksek}]$: Henüz işlenmemiş (bilinmeyen) elemanlar.
- $[\text{yüksek} + 1, n - 1]$: Kesinlikle 2'ler bölgesi.

Her adımda `orta` işaretçisinin gösterdiği elemana bakarız:

- 0 ise: `düşük` ile takas yap, her iki işaretçiyi de bir ilerlet.
- 1 ise: zaten ait olduğu yerdedir, yalnızca `orta`yı ilerlet.
- 2 ise: `yüksek` ile takas yap, sadece `yüksek` işaretçisini bir geri çek (takastan gelen eleman henüz incelenmediği için `orta` ilerlemez).

**Örnek 35.14.** *0, 1 ve 2'lerden oluşan diziyi tek geçişte sıralayan C fonksiyonunu yazınız.*

*Çözüm.*

```
void dutch_national_flag(int arr[], int size) {
    int low = 0;
    int mid = 0;
    int high = size - 1;

    while (mid <= high) {
        if (arr[mid] == 0) {
            int temp = arr[low];
            arr[low] = arr[mid];
            arr[mid] = temp;
            low++;
            mid++;
        } else if (arr[mid] == 1) {
            mid++;
        } else {
            int temp = arr[mid];
            arr[mid] = arr[high];
            arr[high] = temp;
            high--;
        }
    }
}
```

Her döngü adımında ya `orta` bir artar ya da `yuksek` bir azalır. Dolayısıyla döngü en fazla $n$ adımda sonlanır. Zaman karmaşıklığı $O(n)$, alan karmaşıklığı $O(1)$'dir. □

### 35.3.4 Two Pointers Tekniği: Sıralı Dizide Toplam

Sıralı bir dizide toplamları hedef bir $S$ değerine eşit olan iki elemanı bulmak istiyoruz. İç içe iki döngü $O(n^2)$ süre alır. Bölüm 22'de gördüğümüz two pointers tekniğiyle, dizinin sıralı olmasından faydalanarak bu problemi $O(n)$ zamanda çözebiliriz.

**Örnek 35.15.** *Küçükten büyüğe sıralı bir $A$ dizisinde $A[i] + A[j] = S$ olacak biçimde $(i, j)$ indeks çiftini ($i < j$) bulan, yoksa bulunamadığını bildiren C kodunu yazınız.*

*Çözüm.* Aramaya dizinin iki ucundan başlayalım; bir işaretçiyi dizinin başına ($sol = 0$), diğerini sonuna ($sag = n - 1$) yerleştirelim:

- Eğer $A[sol] + A[sag] == S$ ise aradığımız çift bulunmuştur.
- Eğer toplam $S$'den küçükse toplamı büyütmek gerekir; dizi sağa doğru arttığından *sol* işaretçisini 1 artırırız.
- Eğer toplam $S$'den büyükse toplamı küçültmek için *sag* işaretçisini 1 azaltırız.

```c
#include <stdio.h>

int cift_bul(int *A, int n, int S, int *indeks1, int *indeks2) {
    int sol = 0;
    int sag = n - 1;

    while (sol < sag) {
        int simdiki_toplam = A[sol] + A[sag];
        if (simdiki_toplam == S) {
            *indeks1 = sol;
            *indeks2 = sag;
            return 1; // Bulundu
        } else if (simdiki_toplam < S) {
            sol++;
        } else {
            sag--;
        }
    }
    return 0; // Bulunamadi
}
```

Her adımda arama aralığı en az bir eleman daralır (*sol* artar veya *sag* azalır). Toplam işlem sayısı en fazla $n$'dir. Zaman karmaşıklığı $O(n)$, ek bellek ise $O(1)$'dir. □

**Tanım 35.16** (Sık Yapılan Hata: Tamsayı Taşması (Integer Overflow))**.** *İki pozitif tamsayıyı toplarken veya çarparken veri tipinin sınırlarına dikkat edilmelidir. 32-bit işaretli tamsayılar (`int`) en fazla* $2^{31}-1 \approx 2\times10^9$ *değerini alabilir. Örneğin two pointers tekniğinde `A[sol] + A[sag]` ifadesinde elemanlar* $10^9$ *mertebesindeyse toplam* $2 \times 10^9$*'u aşarak negatife taşabilir (`overflow`). Bu tür durumlarda toplam geçici olarak `long long` türüne dönüştürülmelidir:*

$$long\ long\ simdiki\_toplam = (long\ long)A[sol] + A[sag];$$

*Benzer bir durum ikili aramada (Bölüm 24) orta nokta hesabında da yaşanır: `(sol + sag) / 2` yerine taşmayı önlemek için `sol + (sag - sol) / 2` yazılmalıdır.*

## 35.4 Pekiştirme Soruları ve Çözümleri

**Örnek 35.17.** *Aşağıdaki fonksiyonun $n$ girdi boyutu için Big O karmaşıklığını bulunuz:*

```c
void parcala(int n) {
    for (int i = 1; i <= n; i *= 3) {
        for (int j = 1; j <= n; j += i) {
            // O(1) islem
        }
```

```
    }
}
```

*Çözüm.* Dış döngüde $i$ değişkeni $3^0, 3^1, 3^2, \ldots, 3^k$ değerlerini alır; burada $3^k \le n < 3^{k+1}$ olup $k = \lfloor \log_3 n \rfloor$'dir.

Her bir $i = 3^m$ adımı için iç döngü:

$$\left\lceil \frac{n}{3^m} \right\rceil$$

kez döner. Toplam işlem sayısı:

$$T(n) = \sum_{m=0}^{\lfloor \log_3 n \rfloor} \left\lceil \frac{n}{3^m} \right\rceil \approx n \sum_{m=0}^{\lfloor \log_3 n \rfloor} \left(\frac{1}{3}\right)^m$$

Parantez içi, ortak çarpanı $r = 1/3 < 1$ olan azalan bir geometrik seridir:

$$\sum_{m=0}^{\infty} \left(\frac{1}{3}\right)^m = \frac{1}{1 - 1/3} = \frac{3}{2}$$

Seri sonlu adımla sınırlandırıldığında da toplam $3/2$'yi aşamaz:

$$n \le T(n) < \frac{3}{2} n + \log_3 n$$

Dolayısıyla toplam işlem sayısı $\Theta(n)$'dir. İlk bakışta $O(n \log n)$ gibi görünse de geometrik küçülme toplam maliyeti $O(n)$ seviyesinde tutar. □

**Örnek 35.18.** *Aşağıdaki C programı çalıştırıldığında ekrana hangi sayıyı basar?*

```c
#include <stdio.h>

int f(int a, int b) {
    if (b == 0) return 0;
    if (b % 2 == 0) return f(a + a, b / 2);
    return f(a + a, b / 2) + a;
}

int main() {
    printf("%d\n", f(7, 13));
    return 0;
}
```

*Çözüm.* Fonksiyonun matematiksel işlevini adım adım inceleyelim:

- $b = 0$ iken sonuç 0.
- $b$ çift iken: $f(2a, b/2)$.

- $b$ tek iken: $f(2a, b/2) + a$.

Her iki durumda da $a \cdot b$ çarpımı korunmaktadır:

$$(2a) \cdot (b/2) = a \cdot b$$

ve $b$ tek olduğunda:

$$(2a) \cdot \lfloor b/2 \rfloor + a = 2a \cdot \frac{b-1}{2} + a = a(b-1) + a = a \cdot b$$

Bu fonksiyon, Rus Köylü Çarpma Algoritması (Russian Peasant Multiplication) olarak bilinen ikili tabanda çarpma yönteminin rekürsif biçimidir. Fonksiyon temel olarak $a \times b$ işlemini gerçekleştirir.

Girdiler $a = 7$ ve $b = 13$ olduğundan sonuç:

$$7 \times 13 = 91$$

olacaktır.

Adımları doğrudan izleyerek de doğrulayalım:

$$\begin{aligned} f(7,13) &= f(14,6) + 7 \\ f(14,6) &= f(28,3) \\ f(28,3) &= f(56,1) + 28 \\ f(56,1) &= f(112,0) + 56 \\ f(112,0) &= 0 \end{aligned}$$

Geriye doğru toplarsak:

$$\begin{aligned} f(56,1) &= 0 + 56 = 56 \\ f(28,3) &= 56 + 28 = 84 \\ f(14,6) &= 84 \\ f(7,13) &= 84 + 7 = 91 \end{aligned}$$

Program ekrana **91** basar. □

**Örnek 35.19.** *Verilen pozitif bir $n$ tamsayısının ikili açılımındaki en yüksek basamaklı (en soldaki) 1 bitinin konumunu ve sayısal değerini $O(1)$ ek bellek ile hesaplayan bir C işlevi tasarlayınız. Örneğin $n = 18 = (10010)_2$ için sonuç $16 = (10000)_2$ olmalıdır.*

*Çözüm.* En yüksek basamaklı biti tek başına bırakmak için, bu bitin sağında kalan tüm bitleri 1'e dönüştürebilirsek elimize $2^{k+1} - 1$ biçiminde bir maske geçer. Ardından bu maskeden maskenin bir sağa kaydırılmış halini çıkardığımızda geriye yalnızca en yüksek bit kalır.

Tüm sağdaki bitleri 1 yapmanın pratik bir yolu, bit kaydırma ve VEYA (|) işlemlerini basamak basamak yaymaktır (bit-smearing):

1. $n \leftarrow n \mid (n \gg 1)$ (En yüksek bitin hemen yanındaki biti 1 yapar; artık en az iki tane 1 yan yanadır).
2. $n \leftarrow n \mid (n \gg 2)$ (Yan yana olan bu iki 1'i 4 bite çoğaltır).
3. $n \leftarrow n \mid (n \gg 4)$ (4 biti 8 bite çoğaltır).
4. $n \leftarrow n \mid (n \gg 8)$ (8 biti 16 bite çoğaltır).
5. $n \leftarrow n \mid (n \gg 16)$ (16 biti 32 bite çoğaltır).

Bu işlemlerin ardından $n$'in en yüksek bitinden sonraki tüm bitler 1 olur. Örneğin $(00010010)_2 \rightarrow (00011111)_2$.

En yüksek biti elde etmek için:

$$\text{en\_yuksek\_bit} = n - (n \gg 1)$$

işlemi yeterlidir.

```c
#include <stdio.h>

unsigned int en_buyuk_kuvvet(unsigned int n) {
    if (n == 0) return 0;

    n |= n >> 1;
    n |= n >> 2;
    n |= n >> 4;
    n |= n >> 8;
    n |= n >> 16;

    return n - (n >> 1);
}
```

Bu fonksiyon döngü içermez; 32-bitlik tamsayılar için tam olarak 5 kaydırma ve 5 mantıksal işlemle $O(1)$ zamanda kesin sonuca ulaşır. □

# Bölüm 36

# Karma Çözümlü Problemler

TÜBİTAK Bilgisayar Olimpiyatı birinci aşama sınavında karşımıza çıkan en belirleyici sorular, genellikle tek bir konunun sınırları içine hapsedilemeyen sorulardır. Bir kombinatorik sorusunu çözerken Bölüm 14'teki modüler aritmetik araçlarına ihtiyaç duyabilir, bir graf probleminde prefix sums ile Bölüm 17 ve 18'de ele aldığımız parite ve değişmezleri kullanarak sonuca gidebiliriz. Benzer şekilde, dinamik programlama geçişlerini kurarken Bölüm 8'deki içerme-dışarma ilkesinden veya Bölüm 24'teki ikili arama tekniğinden faydalanmak gerekebilir.

Bu bölümde, önceki konularda edindiğimiz kuramsal ve algoritmik birikimi sentezleyecek; farklı disiplinlerin kesişim noktasında duran, olimpiyat standardındaki karma problemleri adım adım inceleyeceğiz.

## 36.1 Konulararası Çözümlü Problemler

Bir problemi çözerken ilk aşama, sorunun arkasındaki matematiksel yapıyı doğru teşhis etmektir. Bir tahta oyunu bir graf olarak modellenebilir mi? Bir dizilim problemi, matris çarpımı veya bitwise işlemlerle optimize edilebilecek bir rekürsiyon bağıntısına indirgenebilir mi? Çözümlerde şu adımları izleyeceğiz:

1. **Küçük Örneklerle Sezgi Kazanma:** Formüle odaklanmadan önce $n = 1, 2, 3$ gibi küçük parametrelerle durumu elle analiz etmek.

2. **Değişmez ve Yarı-Değişmez Arama:** Bölüm 18'de gördüğümüz üzere durum uzayında sabit kalan veya tek yönlü değişen büyüklükleri tespit etmek.

3. **Köprü Kurma:** Problemi bilinen soyut bir yapıya (graf, modüler denklik sınıfı, rekürsiyon tablosu) dönüştürmek.

4. **Çözüm ve Karmaşıklık Analizi:** Matematiksel ispatı tamamlayıp algoritmanın zaman ve bellek karmaşıklığını belirlemek.

### 36.1.1 Problem 1: Halka Üzerinde Düğme Değişimi ve Parite

**Örnek 36.1.** *Dairesel bir masa etrafında dizilmiş $n$ adet lamba bulunmaktadır. Başlangıçta lambaların her biri açık ($1$) ya da kapalı ($0$) durumdadır. Her adımda tüm lambalar **aynı anda** şu kurala göre durum değiştirir: $i$'inci lamba ($0 \le i < n$), eğer kendisi ile saat yönündeki komşusu olan $(i+1) \bmod n$'inci lambanın durumları birbirinden farklıysa yeni adımda açık ($1$), durumları aynıysa kapalı ($0$) hâle gelir.*

*(a) $n = 4$ için başlangıçta tek bir lamba açık, diğerleri kapalıysa en fazla kaç adım sonra tüm lambalar aynı anda söner (hepsi $0$ olur)?*

*(b) Hangi $n$ pozitif tam sayıları için, başlangıç durumu ne olursa olsun, sonlu sayıda adım sonra tüm lambaların sönmesi garanti altındadır?*

Soruda verilen kuralı cebirsel olarak ifade edelim. $t$'inci adımdaki lambaların durumunu $a_i^{(t)} \in \{0,1\}$ ile gösterelim. Yeni durum:

$$a_i^{(t+1)} = \begin{cases} 1, & a_i^{(t)} \neq a_{(i+1) \bmod n}^{(t)} \\ 0, & a_i^{(t)} = a_{(i+1) \bmod n}^{(t)} \end{cases}$$

Bu kural, doğrudan iki tabanında toplama, yani XOR işlemidir:

$$a_i^{(t+1)} = (a_i^{(t)} + a_{(i+1) \bmod n}^{(t)}) \bmod 2$$

İşlemin mod 2'de toplama olması, Bölüm 14'teki modüler aritmetik ve Bölüm 6'daki binom açılımının mod 2'deki özelliklerini akla getirir.

Somut olarak $n = 4$ için tek bir lambanın açık olduğu durumu adım adım yazalım. Başlangıç durumu $(1,0,0,0)$ olsun:

$$\begin{aligned} t=0: &\quad (1,0,0,0) \\ t=1: &\quad (1\oplus 0, 0\oplus 0, 0\oplus 0, 0\oplus 1) = (1,0,0,1) \\ t=2: &\quad (1\oplus 0, 0\oplus 0, 0\oplus 1, 1\oplus 1) = (1,0,1,0) \\ t=3: &\quad (1\oplus 0, 0\oplus 1, 1\oplus 0, 0\oplus 1) = (1,1,1,1) \\ t=4: &\quad (1\oplus 1, 1\oplus 1, 1\oplus 1, 1\oplus 1) = (0,0,0,0) \end{aligned}$$

Görüldüğü üzere tam 4 adımda tüm lambalar kapandı. Peki bu her $n$ için geçerli midir? $n = 3$ için tek bir lamba açıkken denersek: $(1,0,0) \to (1,0,1) \to (1,1,0) \to (0,1,1) \to (1,0,1) \ldots$ Sistem bir döngüye girer ve hiçbir zaman $(0,0,0)$ durumuna ulaşamaz.

Buradaki temel belirleyici, $t$ adım sonraki genel terimin katsayılarının binom katsayıları olmasıdır.

*Çözüm.* İşlemi bir doğrusal operatör olarak ele alabiliriz. Dairesel dizide bir adım ilerleme operatörünü $S$ ile gösterelim: $(S \cdot a)_i = a_{(i+1) \bmod n}$. Bu durumda dönüşüm $T = I + S \pmod 2$ şeklindedir (burada $I$ birim operatördür).

$T$'nin $k$'ıncı kuvvetini binom açılımı ile yazarsak:

$$T^k = (I + S)^k = \sum_{j=0}^{k} \binom{k}{j} S^j \pmod 2$$

Bölüm 15'te gördüğümüz Lucas parite teoremine göre, eğer $k = 2^m$ (2'nin bir kuvveti) ise, $0 < j < 2^m$ için $\binom{2^m}{j}$ çift sayıdır, yani:

$$\binom{2^m}{j} \equiv 0 \pmod 2 \quad (0 < j < 2^m)$$

Dolayısıyla mod 2 aritmetiğinde:

$$T^{2^m} \equiv I^{2^m} + S^{2^m} \equiv I + S^{2^m} \pmod 2$$

Eğer halka uzunluğu $n$, 2'nin bir kuvveti ise, yani $n = 2^m$ seçilirse:

$$S^{2^m} = S^n = I$$

olur; çünkü dairesel dizide $n$ adım kaydırma diziyi kendisine götürür. Buradan:

$$T^n = T^{2^m} \equiv I + I \equiv 2I \equiv 0 \pmod 2$$

elde edilir. Bu sonuç kritik bir gerçeği ortaya koyar: Operatörün $n$'inci kuvveti sıfır operatörüdür. Yani $n = 2^m$ olduğunda, hangi başlangıç durumu seçilirse seçilsin, en fazla $n$ adım sonra tüm elemanlar 0 olmak zorundadır. $n = 4$ için maksimum adım sayısı 4'tür.

Peki ya $n$, 2'nin bir kuvveti değilse? O halde $n = 2^m \cdot d$ ($d > 1$ tek sayı) olarak yazılabilir. Dizi üzerinde $d$ uzunluğunda periyodik bir örüntü kurulabilir. Örneğin tek bir lambanın sıfırlanamadığı $n = 3$ döngüsünü $d$ periyodu olarak kullanırsak, durum uzayında sıfır harici bir döngü mevcuttur. Çekirdek (kernel) analizi veya polinomların ortak böleni incelendiğinde, $x^n - 1$ ile $(1 + x)^k$ polinomlarının mod 2'de ortak kök barındırması için $n$'in 2'nin kuvveti olması şarttır.

Dolayısıyla tüm lambaların her zaman sıfırlanmasını garanti eden $n$ değerleri yalnızca $n = 2^m$ ($m \geq 1$) biçimindeki sayılardır. □

Aşağıdaki C kodu, verilen bir dizinin kaç adımda sıfırlandığını simüle eder ve periyot analizi yapar:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>

typedef struct {
    int *state;
    int step;
```

```
} HistoryEntry;

char *simulate(const int *initial, int n) {
    int *current = (int *)malloc(n * sizeof(int));
    for (int i = 0; i < n; i++) {
        current[i] = initial[i];
    }

    int capacity = 16;
    int count = 0;
    HistoryEntry *history = (HistoryEntry *)malloc(capacity * sizeof(
    HistoryEntry));

    history[count].state = (int *)malloc(n * sizeof(int));
    memcpy(history[count].state, current, n * sizeof(int));
    history[count].step = 0;
    count++;

    int step = 0;
    int *next_state = (int *)malloc(n * sizeof(int));
    char *result = (char *)malloc(128 * sizeof(char));

    while (true) {
        for (int i = 0; i < n; i++) {
            next_state[i] = current[i] ^ current[(i + 1) % n];
        }
        step++;

        for (int i = 0; i < n; i++) {
            current[i] = next_state[i];
        }

        bool all_zero = true;
        for (int i = 0; i < n; i++) {
            if (current[i] != 0) {
                all_zero = false;
                break;
            }
        }

        if (all_zero) {
            snprintf(result, 128, "%d adimda sifirlandi.", step);
            break;
        }

        int found_step = -1;
        for (int i = 0; i < count; i++) {
            if (memcmp(history[i].state, current, n * sizeof(int)) == 0)
    {
```

```
                found_step = history[i].step;
                break;
            }
        }

        if (found_step != -1) {
            snprintf(result, 128, "%d. adimdan %d. adima dongu olustu (
    Periyot: %d).",
                     found_step, step, step - found_step);
            break;
        }

        if (count >= capacity) {
            capacity *= 2;
            history = (HistoryEntry *)realloc(history, capacity * sizeof(
    HistoryEntry));
        }

        history[count].state = (int *)malloc(n * sizeof(int));
        memcpy(history[count].state, current, n * sizeof(int));
        history[count].step = step;
        count++;
    }

    for (int i = 0; i < count; i++) {
        free(history[i].state);
    }
    free(history);
    free(current);
    free(next_state);

    return result;
}

int main(void) {
    int initial4[] = {1, 0, 0, 0};
    char *res4 = simulate(initial4, 4);
    printf("n=4: %s\n", res4);
    free(res4);

    int initial5[] = {1, 0, 0, 0, 0};
    char *res5 = simulate(initial5, 5);
    printf("n=5: %s\n", res5);
    free(res5);

    return 0;
}
```

**Tanım 36.2** (En Sık Yapılan Hata: Lineer Cebir Mantığını Gözden Kaçırmak)**.**

*Olimpiyatlarda bu tip durum geçişi sorularında sıklıkla düşülen yanılgı, birkaç rastgele denemeyle genel kuralı tahmin etmeye çalışmaktır. Durum geçişi XOR veya modüler toplama içeriyorsa, sistem $\mathbb{F}_2$ üzerinde lineerdir. Matris ve operatör cebiri ile binom katsayılarının mod 2'deki özellikleri incelenmeden kesin çıkarımlara ulaşılamaz.*

### 36.1.2 Problem 2: Graf Modellemesi ile Sayılar Teorisinin Kesişimi: Frobenius ve Modülo Grafları

**Örnek 36.3.** *Elinizde yalnızca $a$ ve $b$ gramlık ağırlıklar bulunmaktadır ($\gcd(a,b) = 1$). Bu iki ağırlıktan istendiği kadar (sıfır veya daha fazla) kullanılarak tartılamayan en büyük ağırlık nedir? Tartılamayan toplam kaç farklı pozitif tam sayı ağırlığı vardır?*

Literatürde *Frobenius Madeni Para Problemi* olarak bilinen bu klasik soru; Bölüm 11 ve 12'deki bölünebilme kuralları, Bölüm 14'teki modüler aritmetik ve en kısa yol mantığının kesişiminde etkili bir çözüme sahiptir.

Önce somut bir örnekle başlayalım: $a = 3, b = 5$. Kombinasyonlar: $3x + 5y$ $(x, y \geq 0)$.

- 0: $3(0) + 5(0)$
- 1, 2: tartılamaz.
- 3: $3(1) + 5(0)$
- 4: tartılamaz.
- 5: $3(0) + 5(1)$
- 6: $3(2) + 5(0)$
- 7: tartılamaz.
- 8: $3(1) + 5(1)$
- 9: $3(3) + 5(0)$
- 10: $3(0) + 5(2)$
- 11: $3(2) + 5(1)$
- 12: $3(4) + 5(0)$

Burada $8, 9, 10$ olmak üzere art arda üç sayı elde edildiğine dikkat edelim. $a = 3$ olduğundan, peş peşe 3 sayı üretildikten sonra bunlara istenen sayıda 3 eklenerek sonraki **bütün** tam sayılar oluşturulabilir. Tartılamayan en büyük sayı 7'dir. Tartılamayanlar kümesi: $\{1, 2, 4, 7\}$, yani toplam 4 elemanlıdır. Formül uygulandığında $a = 3, b = 5$ için $3 \cdot 5 - 3 - 5 = 7$ ve $\frac{(3-1)(5-1)}{2} = \frac{2 \cdot 4}{2} = 4$ bulunur. Bu durum genel bir teoremin sonucudur.

*Çözüm.* Problemi mod $a$ kalanları üzerinde düşünelim. Tüm tam sayıları mod $a$'daki denklik sınıflarına $(0, 1, \ldots, a-1)$ ayıralım. Her bir $r \in \{0, 1, \ldots, a-1\}$ kalanı için, $ax + by$ biçiminde yazılabilen **en küçük** sayıyı bulmak isteriz.

Bir sayı $ax + by$ biçimindeyse:

$$ax + by \equiv by \pmod{a}$$

olur. Bölüm 13'te ele aldığımız üzere $\gcd(a, b) = 1$ olduğundan, $y \in \{0, 1, \ldots, a-1\}$ değerleri için $b \cdot y \pmod{a}$ sonuçları $0, 1, \ldots, a-1$ kalanlarının her birini **tam olarak bir kez** üretir (bu bir bijeksiyondur).

Belirli bir $r$ kalanı için en küçük değer, $y$'nin $0 \leq y < a$ aralığındaki tekil değeriyle elde edilen $m_r = b \cdot y$ sayısıdır ($x \geq 0$ koşulunda sayıyı en küçük yapmak için $x = 0$ seçilir).

O halde $r \equiv b \cdot y \pmod{a}$ sınıfındaki sayılar şunlardır:

$$b \cdot y, \quad b \cdot y + a, \quad b \cdot y + 2a, \quad \ldots$$

Bu sınıfta $b \cdot y$'den küçük olup $r$ kalanını veren hiçbir pozitif sayı yazılamaz. Demek ki bu sınıfta üretilemeyen en büyük sayı:

$$b \cdot y - a$$

olur. Tüm kalan sınıfları üzerinden bu değerlerin maksimumunu arıyoruz. $b \cdot y - a$ ifadesi $y$ büyüdükçe artar. $y$'nin alabileceği en büyük değer ise $y = a - 1$'dir. Böylece tartılamayan en büyük sayı:

$$g(a, b) = b(a-1) - a = ab - a - b$$

olarak bulunur.

Şimdi tartılamayan toplam sayı adedini hesaplayalım. Her bir $y \in \{0, 1, \ldots, a-1\}$ için, o sınıfta $b \cdot y$ değerinden küçük ve pozitif olan sayılar $b \cdot y - ka$ $(k \geq 1)$ formundadır. Bu formdaki pozitif sayı adedi $\lfloor \frac{b \cdot y}{a} \rfloor$ tanedir. Toplam tartılamayan sayı adedi:

$$N = \sum_{y=0}^{a-1} \left\lfloor \frac{b \cdot y}{a} \right\rfloor$$

Bu toplamı hesaplamak için Bölüm 10'da ele aldığımız iki yoldan sayma ve simetri mantığından yararlanalım. $y$ yerine $a - 1 - y$ koyarsak:

$$\left\lfloor \frac{b \cdot y}{a} \right\rfloor + \left\lfloor \frac{b(a-y)}{a} \right\rfloor = \left\lfloor \frac{b \cdot y}{a} \right\rfloor + \left\lfloor b - \frac{by}{a} \right\rfloor$$

$\gcd(a, b) = 1$ olduğundan $1 \leq y \leq a - 1$ için $\frac{by}{a}$ hiçbir zaman tam sayı olamaz. Gerçel sayılar için $\alpha \notin \mathbb{Z}$ iken $\lfloor \alpha \rfloor + \lfloor -\alpha \rfloor = -1$ eşitliği geçerlidir. Buradan:

$$\left\lfloor \frac{by}{a} \right\rfloor + \left( b + \left\lfloor -\frac{by}{a} \right\rfloor \right) = b - 1$$

elde edilir. $1 \leq y \leq a-1$ aralığındaki $a-1$ terimi ikişerli eşlersek, her çiftin toplamı $b-1$ olur:

$$2N = (a-1)(b-1) \implies N = \frac{(a-1)(b-1)}{2}$$

Böylece ispat tamamlanır. □

```
// 3 veya daha fazla degisken icin Frobenius Dijkstra ile O(a * log a)
#include <stdio.h>
#define INF 1000000000000000000LL

long long dist[100005];
int visited[100005];

long long frobenius_iki(long long a, long long b) {
    // a ve b aralarinda asal ise
    return a * b - a - b;
}

int main() {
    long long a = 3, b = 5;
    printf("En buyuk tartilamayan: %lld\n", frobenius_iki(a, b));
    printf("Toplam tartilamayan: %lld\n", (a - 1) * (b - 1) / 2);
    return 0;
}
```

### 36.1.3 Problem 3: Prefix Sums ve Güvercin Yuvası: Bölünebilir Alt Diziler

**Örnek 36.4.** *$n$ elemanlı pozitif tam sayılardan oluşan bir $A = [a_1, a_2, \ldots, a_n]$ dizisi veriliyor.*

*(a) Toplamı $n$ ile tam bölünen ardışık bir alt dizinin $(a_l + a_{l+1} + \cdots + a_r)$ daima var olduğunu ispatlayınız $(1 \leq l \leq r \leq n)$.*

*(b) Toplamı $n$ ile tam bölünen ardışık alt dizi sayısını $O(n)$ zamanda bulan bir algoritma tasarlayınız.*

Ardışık toplamları incelerken akla gelen ilk yöntem, Bölüm 22'de ele aldığımız **prefix sums** dizisidir. $S_0 = 0$ ve $1 \leq i \leq n$ için $S_i = \sum_{k=1}^{i} a_k$ diyelim. Herhangi bir ardışık parçanın toplamı:

$$\sum_{k=l}^{r} a_k = S_r - S_{l-1}$$

Bu toplamın $n$'e bölünmesi şu anlama gelir:

$$S_r - S_{l-1} \equiv 0 \pmod{n} \iff S_r \equiv S_{l-1} \pmod{n}$$

Soru böylece şu biçime dönüşür: Mod $n$'de aynı değere sahip iki farklı prefix sumın varlığı garanti midir?

*Çözüm.* (a) Prefix sums dizisini sıralayalım:

$$S_0, S_1, S_2, \ldots, S_n$$

Bu listede tam $n+1$ adet sayı bulunmaktadır.

Bu sayıların mod $n$'deki kalanlarını inceleyelim. Mod $n$'de mümkün olan kalanlar kümesi $\{0, 1, 2, \ldots, n-1\}$ olup tam $n$ elemandan oluşur.

Elimizde $n+1$ adet sayı ve $n$ adet kalan sınıfı vardır. Bölüm 7'de incelediğimiz **Güvercin Yuvası İlkesi** gereğince, kalanları aynı olan en az iki indeks bulunmak zorundadır. Yani öyle $0 \leq j < k \leq n$ indeksleri vardır ki:

$$S_k \equiv S_j \pmod{n}$$

Buradan $S_k - S_j = \sum_{i=j+1}^{k} a_i$ toplamı $n$'in katı olur. $l = j+1$ ve $r = k$ seçildiğinde $1 \leq l \leq r \leq n$ koşulu sağlanır ve aranan alt dizi elde edilir.

(b) Bölünebilen tüm alt dizileri bulmak için aynı kalan sınıfına düşen $S$ değerlerini sayarız. Bir $r \in \{0, 1, \ldots, n-1\}$ kalanı için, modu $r$ olan $c_r$ tane prefix sum varsa; bu sınıftan seçilecek herhangi iki indeks ($j < k$), toplamı $n$'e bölünen geçerli bir alt dizi verir. Böylece toplam geçerli alt dizi sayısı:

$$\text{Toplam} = \sum_{r=0}^{n-1} \binom{c_r}{2}$$

Diziyi baştan sona tek geçişte tararken prefix sumı ve frekans tablosunu güncelleyebiliriz. Tek geçiş $O(n)$ sürer, frekansların kombinasyonunu hesaplamak da $O(n)$ zaman alır. Toplam zaman karmaşıklığı $O(n)$, ek bellek karmaşıklığı mod dizisi için $O(n)$'dir. □

```
#include <stdio.h>

long long bolunebilen_alt_diziler(int arr[], int n) {
    long long sayac[n];
    for (int i = 0; i < n; i++) sayac[i] = 0;

    // S_0 = 0 baslangicta mevcuttur
    sayac[0] = 1;

    long long toplam = 0;
    for (int i = 0; i < n; i++) {
        toplam = (toplam + arr[i]) % n;
        // Negatif mod durumunu onle
        if (toplam < 0) toplam += n;
        sayac[toplam]++;
```

```
    }

    long long sonuc = 0;
    for (int r = 0; r < n; r++) {
        if (sayac[r] >= 2) {
            sonuc += sayac[r] * (sayac[r] - 1) / 2;
        }
    }
    return sonuc;
}
```

**Tanım 36.5** (En Sık Yapılan Hata: $S_0$ Başlangıç Değerini Unutmak)**.** *Prefix sums dizisiyle çalışırken $S_0 = 0$ durumunu göz ardı etmek, dizinin en başından başlayan ($l = 1$) ve tek başına $n$'e bölünen alt dizilerin sayılmamasına yol açar. Frekans tablosunda* `sayac[0]` *değeri başlangıçta mutlaka* 1 *olarak ayarlanmalıdır.*

### 36.1.4 Problem 4: Ağaçta Yol İşlemleri ve Bit Düzeyinde XOR Özellikleri

**Örnek 36.6.** *$n$ düğümlü bir ağaçta her kenarın üzerinde pozitif bir tam sayı ağırlık bulunmaktadır. İki düğüm $u$ ve $v$ arasındaki basit yol üzerindeki kenarların ağırlıklarının XOR toplamını $d(u, v)$ ile gösterelim. Tüm $1 \leq u < v \leq n$ çiftleri için $d(u, v) = 0$ olan çiftlerin sayısını $O(n \log n)$ veya $O(n)$ zamanda bulan bir algoritma geliştiriniz.*

Bir ağaçta $u$ ile $v$ arasındaki yol tekildir; $u$'dan en yakın ortak ataya (LCA) çıkar, oradan $v$'ye iner. Kenar ağırlıklarını tek tek toplamak karmaşık görünse de, işlemimiz **XOR** ($\oplus$) işlemidir. XOR işleminin temel özellikleri şunlardır:

1. $x \oplus x = 0$ (Her sayı kendisini sıfırlar).
2. $x \oplus 0 = x$.
3. Değişme ve birleşme özellikleri geçerlidir.

Rastgele bir kök seçelim (örneğin düğüm 1). Kökten herhangi bir $w$ düğümüne giden yol üzerindeki kenarların XOR toplamını $X[w]$ olarak tanımlayalım. $u$ ile $v$ arasındaki yolu kök üzerinden ifade edebilir miyiz? $u$'dan köke çıkıp kökten $v$'ye inen hayali bir yol düşünelim. Kök ile LCA$(u, v)$ arasındaki kenarlar iki kez geçilecektir. XOR işleminde çift sayıda geçen kenarlar birbirini sıfırlar:

$$d(u, v) = X[u] \oplus X[v]$$

Bu dönüşüm problemi önemli ölçüde sadeleştirir; ağaç yapısı üzerinde LCA arama zorunluluğu tamamen ortadan kalkar. $d(u, v) = 0$ olması ne anlama gelir?

$$X[u] \oplus X[v] = 0 \iff X[u] = X[v]$$

Problem, kökten tüm düğümlere hesaplanan $X[w]$ değerleri içinde birbirine eşit olan eleman çiftlerini saymaya dönüşür.

*Çözüm.* **Algoritma Adımları:**

1. Ağaçta 1 numaralı düğümü kök kabul edelim.
2. Bölüm 28'de gördüğümüz Derinlik Öncelikli Arama (DFS) ile her $u$ düğümü için $X[u]$ değerini hesaplayalım:

$$X[\text{kök}] = 0, \quad X[v] = X[u] \oplus w(u, v)$$

3. Elde edilen $X[1], X[2], \ldots, X[n]$ dizisini sıralayalım ($O(n \log n)$) veya bir hash table'a ekleyelim ($O(n)$).
4. Her bir farklı değerin kaç kez tekrar ettiğini ($k$) bularak $\binom{k}{2} = \frac{k(k-1)}{2}$ değerlerini toplayalım.

Tüm ağacı DFS ile dolaşmak $O(n)$ sürer. Değerleri sıralamak $O(n \log n)$, frekansları toplamak $O(n)$ zaman alır. Toplam zaman karmaşıklığı $O(n \log n)$, ek bellek karmaşıklığı $O(n)$'dir. □

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

typedef struct {
    int u;
    int v;
    int w;
} Edge;

typedef struct AdjNode {
    int to;
    int weight;
    struct AdjNode* next;
} AdjNode;

typedef struct {
    int u;
    int current_xor;
} StackItem;

static int compare_ints(const void* a, const void* b) {
    int arg1 = *(const int*)a;
    int arg2 = *(const int*)b;
    if (arg1 < arg2) return -1;
    if (arg1 > arg2) return 1;
```

```
    return 0;
}

long long count_zero_xor_pairs(int n, const Edge* edges, int num_edges) {
    AdjNode** graph = (AdjNode**)calloc(n + 1, sizeof(AdjNode*));
    for (int i = 0; i < num_edges; ++i) {
        int u = edges[i].u;
        int v = edges[i].v;
        int w = edges[i].w;

        AdjNode* node1 = (AdjNode*)malloc(sizeof(AdjNode));
        node1->to = v;
        node1->weight = w;
        node1->next = graph[u];
        graph[u] = node1;

        AdjNode* node2 = (AdjNode*)malloc(sizeof(AdjNode));
        node2->to = u;
        node2->weight = w;
        node2->next = graph[v];
        graph[v] = node2;
    }

    int* X = (int*)calloc(n + 1, sizeof(int));
    bool* visited = (bool*)calloc(n + 1, sizeof(bool));
    StackItem* stack = (StackItem*)malloc((n + 1) * sizeof(StackItem));
    int top = 0;

    stack[top].u = 1;
    stack[top].current_xor = 0;
    top++;
    visited[1] = true;

    while (top > 0) {
        top--;
        int u = stack[top].u;
        int current_xor = stack[top].current_xor;
        X[u] = current_xor;

        for (AdjNode* curr = graph[u]; curr != NULL; curr = curr->next) {
            int v = curr->to;
            int w = curr->weight;
            if (!visited[v]) {
                visited[v] = true;
                stack[top].u = v;
                stack[top].current_xor = current_xor ^ w;
                top++;
            }
        }
```

```
    }

    qsort(&X[1], n, sizeof(int), compare_ints);

    long long total_pairs = 0;
    long long count = 1;
    for (int i = 2; i <= n; ++i) {
        if (X[i] == X[i - 1]) {
            count++;
        } else {
            if (count >= 2) {
                total_pairs += count * (count - 1) / 2;
            }
            count = 1;
        }
    }
    if (count >= 2) {
        total_pairs += count * (count - 1) / 2;
    }

    for (int i = 1; i <= n; ++i) {
        AdjNode* curr = graph[i];
        while (curr != NULL) {
            AdjNode* temp = curr;
            curr = curr->next;
            free(temp);
        }
    }
    free(graph);
    free(X);
    free(visited);
    free(stack);

    return total_pairs;
}

int main(void) {
    Edge edges[] = {
        {1, 2, 5},
        {2, 3, 5},
        {3, 4, 2}
    };
    int n = 4;
    int num_edges = sizeof(edges) / sizeof(edges[0]);

    long long result = count_zero_xor_pairs(n, edges, num_edges);
    printf("XOR toplami 0 olan cift sayisi: %lld\n", result);

    return 0;
```

```
}
```

### 36.1.5 Problem 5: Uç Değer İlkesi ve Turnuva Grafları

**Örnek 36.7.** *$n$ kişinin katıldığı bir turnuvada her oyuncu diğer her oyuncuyla tam olarak bir maç yapmıştır ve hiçbir maç berabere bitmemiştir. Bir oyuncunun diğerini yendiği durumları yönlü kenar olarak gösterelim (bu yapı bir turnuva grafı oluşturur).*

*Tüm katılımcıları öyle bir $p_1, p_2, \ldots, p_n$ sırasında diziniz ki; her $i$ $(1 \leq i < n)$ için $p_i$ oyuncusu $p_{i+1}$ oyuncusunu yenmiş olsun (yönlü Hamilton yolu). Böyle bir dizilimin her zaman var olduğunu ispatlayıp $O(n^2)$ zamanda bulan bir algoritma veriniz.*

Genel yönlü graflarda Hamilton yolu bulmak NP-tam bir problemdir. Ancak turnuva grafları **tam graflardır**; yani herhangi iki tepe arasında mutlaka bir yönde kenar bulunur. Bu yapısal özellik çözümü doğrudan kolaylaştırır. Tümevarım veya Bölüm 19'da gördüğümüz **Uç Değer İlkesi** üzerinden düşünelim: Varsayalım ki $k$ uzunluğunda bir yol kurduk: $v_1 \to v_2 \to \cdots \to v_k$. Dışarıda kalan bir $u$ tepesini bu yolun içine dahil edebilir miyiz?

- Eğer $u$, baştaki $v_1$'i yeniyorsa: $u \to v_1 \to \cdots \to v_k$ yaparak yolu başa doğru uzatabiliriz.
- Eğer sondaki $v_k$, $u$'yu yeniyorsa: $v_1 \to \cdots \to v_k \to u$ yaparak yolu sona doğru uzatabiliriz.
- Peki ya $v_1 \to u$ ve $u \to v_k$ ise? Yani $u$ baştakine yenilmiş, sondakini ise yenmişse?

Bu durumda yol boyunca ilerlerken okların yön değiştirdiği bir nokta bulunmalıdır: $v_i \to u$ iken hemen ardından $u \to v_{i+1}$ olan ardışık iki tepe mutlaka vardır.

*Çözüm.* İspatı $n$ üzerinden matematiksel tümevarım ile kuralım.

**Temel Adım ($n = 2$):** İki oyuncu arasında tek bir maç oynanmıştır; yenen $p_1$, yenilen $p_2$ seçilirse $p_1 \to p_2$ yolu doğrudan kurulur.

**Tümevarım Adımı:** $k$ oyunculu herhangi bir turnuvada yönlü Hamilton yolunun var olduğunu varsayalım: $P = (v_1, v_2, \ldots, v_k)$. Sisteme $(k+1)$'inci bir $u$ oyuncusu ekleyelim. Üç durum ortaya çıkar:

1. $u \to v_1$ ise: $u$, yolun en başına eklenir: $(u, v_1, v_2, \ldots, v_k)$.
2. $v_k \to u$ ise: $u$, yolun en sonuna eklenir: $(v_1, v_2, \ldots, v_k, u)$.

3. Yukarıdaki iki durum da geçerli değilse, $v_1 \to u$ ve $u \to v_k$ olmalıdır. Yol üzerindeki indisleri inceleyelim. $v_i \to u$ koşulunu sağlayan en büyük indisi $i$ olarak seçelim. $v_1 \to u$ olduğundan böyle bir $i \geq 1$ mutlaka vardır. $i = k$ olamaz; çünkü $u \to v_k$ olduğunu biliyoruz. Demek ki $1 \leq i < k$'dir. $i$'nin maksimal seçilmesinden ötürü $v_{i+1} \to u$ olamaz; graf bir turnuva olduğundan maç berabere bitemez, dolayısıyla zorunlu olarak $u \to v_{i+1}$ olmalıdır. O halde $u$ tepesini $v_i$ ile $v_{i+1}$ arasına yerleştirebiliriz:

$$v_1 \to \cdots \to v_i \to u \to v_{i+1} \to \cdots \to v_k$$

Her durumda $(k+1)$ uzunluğunda geçerli bir yönlü yol elde edilir ve tümevarım tamamlanır.

**Algoritmik Karmaşıklık:** Bu yöntem, Bölüm 23'te incelediğimiz *Araya Ekleme Sıralaması* (Insertion Sort) mantığıyla örtüşür. $k$'ıncı elemanı sıralanmış $k-1$ elemanlı yola yerleştirmek için en fazla $k$ karşılaştırma yapılır. Toplam işlem sayısı:

$$\sum_{k=1}^{n-1} k = \frac{n(n-1)}{2} = O(n^2)$$

karşılaştırmadır. İkili arama kullanılarak karşılaştırma sayısı $O(n \log n)$ seviyesine çekilebilir (dizi kaydırma maliyeti yine $O(n^2)$ kalır). □

```c
#include <stdio.h>
#include <stdlib.h>

// matris[i][j] = 1 ise i, j'yi yenmistir
void hamilton_yolu_bul(int n, int matris[100][100], int yol[]) {
    yol[0] = 0; // Ilk dugumle basla
    int boyut = 1;

    for (int u = 1; u < n; u++) {
        // Durum 1: u en basa gelir mi?
        if (matris[u][yol[0]]) {
            for (int j = boyut; j > 0; j--) yol[j] = yol[j - 1];
            yol[0] = u;
        }
        // Durum 2: u en sona gelir mi?
        else if (matris[yol[boyut - 1]][u]) {
            yol[boyut] = u;
        }
        // Durum 3: Araya ekleme
        else {
            int pos = -1;
            for (int i = 0; i < boyut - 1; i++) {
                if (matris[yol[i]][u] && matris[u][yol[i + 1]]) {
                    pos = i + 1;
                    break;
```

```
                }
            }
            for (int j = boyut; j > pos; j--) yol[j] = yol[j - 1];
            yol[pos] = u;
        }
        boyut++;
    }
}
```

### 36.1.6 Problem 6: İçerme-Dışarma ve Dinamik Programlama: Yasaklı Pozisyonlu Permütasyonlar

**Örnek 36.8.** *$1, 2, \ldots, n$ sayılarının bir $\pi$ permütasyonunda hiçbir $i$ $(1 \leq i \leq n)$ için $\pi(i) = i$ olmaması durumuna deranjman (düzensiz permütasyon) denir. Şimdi kısıtı genişletelim: Boyutu $n$ olan bir permütasyonda her $i$ için hem $\pi(i) \neq i$ hem de $\pi(i) \neq i+1$ $(1 \leq i < n)$ olmasını istiyoruz. Böyle permütasyonların sayısını hesaplayan bir yöntem geliştiriniz.*

Bu soru, klasik *Problème des ménages* probleminin temel yapısıdır. Kombinatorikte "hiçbiri sağlanmasın" koşulunu doğrudan saymak zordur; çünkü yasakların birbiriyle çakışma ihtimalleri yüksektir. Bunun yerine Bölüm 8'de kurduğumuz **İçerme-Dışarma İlkesi** devreye girer:

$$N(\text{hiçbiri}) = \sum_{k=0}^{n} (-1)^k \cdot r_k \cdot (n-k)!$$

Burada $r_k$, konulabilecek $k$ adet bağımsız "yasaklı pozisyonu" tahtaya yerleştirme sayısıdır. Yasaklar yerleştirildikten sonra geriye kalan $n-k$ serbest eleman $(n-k)!$ farklı biçimde dağıtılır.

Yasaklı hücreleri bir $n \times n$ tahta üzerinde düşünebiliriz: $(i, i)$ ana köşegeni ve $(i, i+1)$ üst yan köşegeni yasaktır. Toplam $2n-1$ adet yasaklı kare vardır. Bu karelerden aynı satırda veya aynı sütunda olmayan (yani birbirini kale gibi tehdit etmeyen) $k$ tanesini seçmeliyiz. Bu karelerin dizilimi incelendiğinde, ardışık bağımlılıklar bir yol oluşturur. Bir doğru üzerinde yan yana olmayan elemanları seçme problemi ise Bölüm 5'teki temel seçim teknikleriyle çözülebilir.

*Çözüm.* Yasaklı kareleri sıralayalım:

$$(1,1), (1,2), (2,2), (2,3), (3,3), \ldots, (n-1, n), (n, n)$$

Burada toplam $M = 2n-1$ adet yasaklı hücre vardır:

- $(1,1)$ ile $(1,2)$ aynı satırdadır, birlikte seçilemez.
- $(1,2)$ ile $(2,2)$ aynı sütundadır, birlikte seçilemez.

Genel olarak, sıralanan bu $M$ karede **yan yana gelen herhangi iki kare** ya aynı satırda ya da aynı sütundadır. Dolayısıyla aynı anda seçilemezler.

Böylece soru şu tanıdık probleme dönüşür: $M = 2n - 1$ elemanlı bir sırada, hiçbir ikisi ardışık olmayan $k$ eleman kaç farklı biçimde seçilebilir?

Bu seçim sayısı yıldız-bölücü yöntemiyle hesaplanır:

$$r_k = \binom{M - k + 1}{k} = \binom{(2n - 1) - k + 1}{k} = \binom{2n - k}{k}$$

Seçilen bu $k$ yasaklı karenin her biri farklı satır ve sütunda olduğundan, kalan $n - k$ eleman kendi aralarında $(n - k)!$ şekilde yerleşir.

İçerme-dışarma ilkesi gereğince, hiçbir yasaklı kareye denk gelmeyen permütasyonların sayısı:

$$P_n = \sum_{k=0}^{n} (-1)^k \binom{2n - k}{k} (n - k)!$$

olarak kapalı formda elde edilir. Bu toplam $O(n)$ adımda hesaplanabilir. □

```
#include <stdio.h>
#include <stdlib.h>

long long power_mod(long long base, long long exp, long long mod) {
    long long result = 1;
    base %= mod;
    while (exp > 0) {
        if (exp % 2 == 1) {
            result = (result * base) % mod;
        }
        base = (base * base) % mod;
        exp /= 2;
    }
    return result;
}

void factorial_mod(int n, long long mod, long long *f) {
    f[0] = 1;
    for (int i = 1; i <= n; i++) {
        f[i] = (f[i - 1] * i) % mod;
    }
}

long long binom(int n, int k, const long long *f, const long long *inv,
    long long mod) {
    if (k < 0 || k > n) {
        return 0;
    }
    return (((f[n] * inv[k]) % mod) * inv[n - k]) % mod;
}
```

```
long long restricted_permutation(int n, long long mod) {
    int max_val = 2 * n + 1;
    long long *f = (long long *)malloc((max_val + 1) * sizeof(long long));

    long long *inv = (long long *)malloc((max_val + 1) * sizeof(long long
    ));

    factorial_mod(max_val, mod, f);

    inv[max_val] = power_mod(f[max_val], mod - 2, mod);
    for (int i = max_val - 1; i >= 0; i--) {
        inv[i] = (inv[i + 1] * (i + 1)) % mod;
    }

    long long total = 0;
    for (int k = 0; k <= n; k++) {
        long long choice = binom(2 * n - k, k, f, inv, mod);
        long long remaining_perm = f[n - k];
        long long term = (choice * remaining_perm) % mod;

        if (k % 2 == 1) {
            total = (total - term + mod) % mod;
        } else {
            total = (total + term) % mod;
        }
    }

    free(f);
    free(inv);

    return total;
}

int main(void) {
    long long mod = 1000000007;
    printf("n = 4 icin sonuc: %lld\n", restricted_permutation(4, mod));
    return 0;
}
```

### 36.1.7 Problem 7: İkili Arama ve Yarı-Değişmezler: Medyanların Medyanı Matrisi

**Örnek 36.9.** *$N \times N$ boyutunda bir matrisin her satırı soldan sağa, her sütunu yukarıdan aşağıya kesin artandır. Bu matrisin tüm elemanları tek boyutlu bir dizide sıralansaydı ortanca (medyan) eleman hangisi olurdu? Elemanlara yalnızca satır ve sütun erişimi yapabildiğimiz varsayımıyla, matrisin medyanını $O(N \log(MAX-$*

*MIN*)) *zaman karmaşıklığında bulan bir algoritma tasarlayınız.*

Matristeki toplam eleman sayısı $N^2$'dir. Medyan, kendisinden küçük veya eşit eleman sayısı en az $\lceil N^2/2 \rceil$ olan en küçük değerdir. Tüm matrisi bir diziye aktarıp sıralamak $O(N^2 \log N)$ zaman ve $O(N^2)$ bellek gerektirir; bu yaklaşım büyük boyutlu matrislerde verimsiz kalır.

Burada Bölüm 24'te gördüğümüz **cevap üzerinde ikili arama** (binary search on answer) tekniğinden yararlanabiliriz: Belirli bir $X$ sayısı için matriste $X$'ten küçük veya eşit kaç eleman olduğunu hızla sayabilir miyiz? Matrisin satırları ve sütunları sıralı olduğundan, sayma işlemini $O(N)$ zamanda tamamlayabiliriz: Sol alt köşeden $((N-1, 0))$ başlarız. Eğer bulunulan hücredeki eleman $\leq X$ ise, o sütundaki üst elemanların tümü de $\leq X$ olmak zorundadır. O sütundaki katkıyı doğrudan ekleyip sağa $(j++)$ geçeriz. Eğer eleman $> X$ ise, yukarı $(i--)$ çıkarız. Bu adım en fazla $2N$ hamle sürer.

*Çözüm.* **Monotonluk Doğrulaması:** $f(X)$, matriste değeri $\leq X$ olan elemanların sayısı olsun. $X$ değeri arttıkça $f(X)$ değeri de azalmaz (monotondur):

$$X_1 \leq X_2 \implies f(X_1) \leq f(X_2)$$

Dolayısıyla ikili arama uygulanabilir. Aranan medyan eleman, $f(X) \geq \frac{N^2+1}{2}$ koşulunu sağlayan en küçük $X$'tir.

**Sayma Algoritması ($O(N)$):**

1. $i = N-1$ (en alt satır), $j = 0$ (en sol sütun), adet $= 0$.
2. $i \geq 0$ ve $j < N$ olduğu sürece:
   - Eğer $M[i][j] \leq X$ ise: Bu sütundaki 0'dan $i$'ye kadar olan tüm elemanlar (toplam $i+1$ tane) kesinlikle $\leq X$'tir. adet$+= (i+1)$ yapılır ve sonraki sütuna geçilir $(j++)$.
   - Eğer $M[i][j] > X$ ise: Bu eleman $X$'ten büyüktür, daha küçük elemanlar bulmak için yukarı satıra çıkılır $(i--)$.
3. Döngü tamamlandığında adet, matristeki $\leq X$ olan eleman sayısını verir.

Her adımda ya $i$ bir azalır ya da $j$ bir artar. Toplam adım sayısı en fazla $2N$'dir; dolayısıyla sayma işlemi $O(N)$ karmaşıklıktadır. İkili arama aralığı $[M[0][0], M[N-1][N-1]]$ olup adım sayısı $\log(\text{MAX} - \text{MIN})$ kadardır. Toplam zaman karmaşıklığı:

$$O(N \log(\text{MAX} - \text{MIN}))$$

Ek bellek karmaşıklığı ise yalnızca $O(1)$'dir. □

```c
#include <stdio.h>

// Belirli bir X degerinden kucuk veya esit eleman sayisini O(N)'de sayar
int kucuk_esit_sayisi(int N, int M[100][100], int X) {
    int adet = 0;
    int i = N - 1;
    int j = 0;

    while (i >= 0 && j < N) {
        if (M[i][j] <= X) {
            adet += (i + 1);
            j++;
        } else {
            i--;
        }
    }
    return adet;
}

int matris_medyani(int N, int M[100][100]) {
    int sol = M[0][0];
    int sag = M[N - 1][N - 1];
    int hedef_sira = (N * N + 1) / 2;
    int cevap = sag;

    while (sol <= sag) {
        int orta = sol + (sag - sol) / 2;
        if (kucuk_esit_sayisi(N, M, orta) >= hedef_sira) {
            cevap = orta;
            sag = orta - 1; // Daha kucuk bir deger medyan olabilir mi?
        } else {
            sol = orta + 1;
        }
    }
    return cevap;
}
```

### 36.1.8 Problem 8: Sayılar Teorisi ve Dinamik Programlama: Bölünebilirlik Grafında En Uzun Yol

**Örnek 36.10.** *$S = \{1, 2, \ldots, n\}$ kümesi veriliyor. Bu kümeden seçilen elemanlarla öyle bir $x_1 < x_2 < \cdots < x_k$ dizisi oluşturulacaktır ki; her $i$ $(1 \leq i < k)$ için $x_i \mid x_{i+1}$ ($x_i$, $x_{i+1}$'i tam böler) olacaktır.*

*(a) Böyle bir zincirin maksimum eleman sayısı ($k$) nedir?*

*(b) Bu maksimum uzunluğa sahip kaç farklı zincir kurulabilir?*

Zincirdeki her eleman bir sonrakini tam bölmelidir (Bölüm 11): $x_{i+1} = c_i \cdot x_i$ ($c_i \geq 2$ bir tam sayı). Zinciri olabildiğince uzatmak için her adımda sayıyı en küçük tam sayı çarpanı olan 2 ile çarpmak gerekir. Başlangıçta $x_1 = 1$ alırsak:

$$1, 2, 4, 8, 16, \ldots, 2^m \leq n$$

Maksimum uzunluk doğrudan 2'nin kuvveti ile belirlenir: $k = \lfloor \log_2 n \rfloor + 1$. Peki bu uzunlukta kaç farklı zincir vardır? Yalnızca 2 çarpanı mı kullanılabilir? Bir adımda 2 yerine 3 çarpanı seçilirse ne olur? Örneğin $n = 12$ için: $k = \lfloor \log_2 12 \rfloor + 1 = 3 + 1 = 4$. Uzunluğu 4 olan zincirler:

- $1 \to 2 \to 4 \to 8$ (çarpanlar: $2, 2, 2$)
- $1 \to 2 \to 4 \to 12$ (çarpanlar: $2, 2, 3$)
- $1 \to 2 \to 6 \to 12$ (çarpanlar: $2, 3, 2$)
- $1 \to 3 \to 6 \to 12$ (çarpanlar: $3, 2, 2$)

Çarpanlar kümesinde 3 çarpanı yer alabilir; çünkü $2 \cdot 2 \cdot 3 = 12 \leq 12$'dir. Ancak $2 \cdot 3 \cdot 3 = 18 > 12$ olduğundan iki adet 3 çarpanı bulunamaz. Problem, çarpanların permütasyonuna ve dinamik programlama sayımına dönüşür.

*Çözüm.* Zincirin eleman sayısı $k$ olsun. Zincirin son elemanı $x_k \leq n$ olmalıdır. $x_k$'nin asal çarpanlarına ayrılmış biçimi:

$$x_k = p_1^{a_1} p_2^{a_2} \ldots p_r^{a_r}$$

olmak üzere, $x_1 = 1$ kabul edildiğinde zincirin uzunluğu çarpanların asal üslerinin toplamının bir fazlasıdır:

$$k = 1 + \sum_{i=1}^{r} a_i$$

Bu toplamı maksimize etmek için en küçük asal olan 2'yi seçmeliyiz. O halde:

$$k_{\max} = \lfloor \log_2 n \rfloor + 1$$

olur. $m = k_{\max} - 1 = \lfloor \log_2 n \rfloor$ diyelim.

En uzun zincirin son elemanı $x_k \leq n$ şu iki yapıdan birinde olmalıdır:

1. Sadece 2'lerden oluşanlar: $x_k = 2^m$.
2. Tam olarak bir tane 3 asalına ve $m - 1$ tane 2 asalına sahip olanlar: $x_k = 2^{m-1} \cdot 3$. ($2^{m-1} \cdot 3 \leq n$ ise bu durum mümkündür; $2^{m-2} \cdot 3^2 > 2^m$ olduğundan birden fazla 3 bulunamaz; ayrıca $5 > 4 = 2^2$ olduğundan 5 asalı iki adet 2'nin yerini tutamaz).

Şimdi bu zincirlerin sayısını hesaplayalım:

- **Durum 1: Yalnızca 2 çarpanları ($x_k = 2^m \cdot a$):** İlk eleman $x_1 = a$ olabilir ($a \cdot 2^m \leq n$ koşuluyla). Her $a \in [1, \lfloor n/2^m \rfloor]$ için $a \to 2a \to 4a \cdots \to 2^m a$ şeklinde tek bir zincir kurulur. Buradan $\lfloor \frac{n}{2^m} \rfloor$ adet zincir elde edilir.
- **Durum 2: Bir adet 3 ve $m-1$ adet 2 çarpanı ($x_k = 2^{m-1} \cdot 3 \cdot a \leq n$):** Çarpanlar kümesi $\{3, 2, 2, \ldots, 2\}$ biçimindedir. 3 çarpanının hangi adımda kullanılacağı serbesttir; toplam $m$ adımdan herhangi birinde yer alabilir. Bölüm 4'te ele aldığımız tekrarlı permütasyon mantığıyla $m$ farklı sıra mümkündür:

$$\binom{m}{1} = m$$

  Bu durumdaki her geçerli $a$ ($1 \leq a \leq \lfloor \frac{n}{2^{m-1} \cdot 3} \rfloor$) için $m$ farklı zincir oluşur. Buradan $m \cdot \lfloor \frac{n}{2^{m-1} \cdot 3} \rfloor$ adet zincir gelir.

Toplam zincir sayısı:

$$\text{Toplam} = \left\lfloor \frac{n}{2^m} \right\rfloor + m \cdot \left\lfloor \frac{n}{2^{m-1} \cdot 3} \right\rfloor$$

olarak $O(1)$ zamanda hesaplanır. □

```c
#include <stdio.h>

typedef struct {
    long long max_length;
    long long count;
} Result;

Result longest_divisibility_chains(long long n) {
    if (n == 1) {
        Result r = {1, 1};
        return r;
    }

    long long m = 0;
    while ((1ULL << (m + 1)) <= (unsigned long long)n) {
        m++;
    }

    long long max_length = m + 1;
    long long count = n / (1ULL << m);

    if (m >= 1) {
        unsigned long long threshold = (1ULL << (m - 1)) * 3ULL;
        if (threshold <= (unsigned long long)n) {
            count += m * (n / threshold);
        }
    }
```

```
    Result r = {max_length, count};
    return r;
}

int main(void) {
    Result res12 = longest_divisibility_chains(12);
    printf("n = 12: (%lld, %lld)\n", res12.max_length, res12.count);

    Result res20 = longest_divisibility_chains(20);
    printf("n = 20: (%lld, %lld)\n", res20.max_length, res20.count);

    return 0;
}
```

### 36.1.9 Bölüm Özeti ve Olimpiyat İçin Stratejik Tavsiyeler

Bu bölümde incelediğimiz problemler, olimpiyat sorularının ayırt edici özelliklerini açıkça ortaya koymaktadır:

- **Modüler Dönüşümler ve Bit Aritmetiği:** Toplama ve çıkarmanın mod 2'de XOR işlemine karşılık gelmesi, dairesel yapılar ile binom katsayılarının Bölüm 15'teki Lucas parite teoremi üzerinden kurduğu bağı sıkça karşımıza çıkarır.
- **Graf Soyutlaması:** Doğrudan bir graf olarak verilmeyen durum uzayları (Frobenius madeni para problemi, turnuva sıralamaları), tepe ve yönlü kenarlarla modellendiğinde Dijkstra veya Hamilton yolu gibi standart algoritmalara indirgenebilir.
- **Kombinatorik Kısıtları Ayrıştırma:** Yasaklı permütasyonlar ve dizi yerleşimlerinde doğrudan sayım yapmak yerine içerme-dışarma ve bağımsız küme dönüşümleri tercih edilmelidir.
- **Monotonluk ve İkili Arama:** Bir optimizasyon veya sayma probleminde aranan değer arttıkça sağlanan koşul tek yönlü (monoton) bir değişim gösteriyorsa, doğrudan değer uzayında ikili arama kurgulanmalıdır.

# Bölüm 37

# Ek Çözümlü Problemler ve İpuçları

Önceki bölümlerde kurduğumuz teorik temeller, kurallar ve algoritmalar; sınav soruları karşısında somut birer çözüm aracına dönüştüklerinde anlam kazanır. Bir problemi ilk denemede çözememek bir eksiklik değil, zihnimizde henüz kurulmamış bir bağlantıyı keşfetme fırsatıdır. **Bu bölümdeki problemler, önceki bölümlerde öğrendiğimiz araçları pekiştirmek için hazırlanmış ek çözümlü sorulardır**; konu başlıklarına göre (sayma, sayılar teorisi ve algoritma) üç kısma ayrılmıştır. Her çözümün sonunda benzer sorularda yol gösterecek ipuçları da veriyoruz.

Çözümlerde yalnızca doğru yanıtı vermekle yetinmeyip bir yarışmacının soruya yaklaşırken yakalaması gereken ipuçlarını, olası tuzakları ve sonuca ulaştıran akıl yürütme adımlarını öne çıkarıyoruz.

## 37.1 Ek Problemler: İleri Sayma ve Kombinatorik

Sayma problemleri ilk bakışta sade görünse de kısıtlar arttıkça doğrudan listelemek olanaksız hale gelir. Bu kısımda kısıtlı dizilimler (Bölüm 4), içerme-dışarma ilkesi (Bölüm 8), ayraç yöntemi (Bölüm 5) ve güvercin yuvası ilkesi (Bölüm 7) gibi tekniklerin tipik uygulamaları üzerinde duruyoruz.

### 37.1.1 Kısıtlı Sıralama ve Özdeş Nesneler

**Örnek 37.1.** $\{1,2,3,4,5,6,7\}$ *kümesinin elemanları kullanılarak yazılabilecek tüm 7 basamaklı, rakamları tekrarsız sayıların kaç tanesinde tek rakamlar kendi aralarında küçükten büyüğe sıralı (soldan sağa* $1,3,5,7$ *sırasında), fakat çift rakamlar da kendi aralarında büyükten küçüğe sıralı (soldan sağa* $6,4,2$ *sırasında) yer alır?*

*Çözüm.* Soruda 7 farklı basamağın yerleştirilmesi isteniyor. Eğer tek basamaklar $(1,3,5,7)$ ve çift basamaklar $(2,4,6)$ rastgele dağılsaydı 7! farklı durum oluşurdu. Ancak iki grup üzerinde de kesin bir sıra şartı vardır. Teklerin kendi aralarındaki $1<3<5<7$ sıralaması tek bir biçimde mümkündür; benzer şekilde çiftlerin $6>4>2$ sıralaması da tek bir dizilim verir. Dolayısıyla bir grubun yerlerini seçtiğimiz anda o grubun elemanlarının yazılış sırası tamamen belirlenir; geriye kalan yerlere de diğer grubun elemanları zorunlu olarak tek biçimde yerleşir.

Adım adım kuralım:

1. Toplam 7 basamak pozisyonumuz var: _____.

2. Bu 7 pozisyondan 4 tanesini tek sayılar için seçelim. Bu seçim kombinasyon ile yapılır:
$$\binom{7}{4} = \frac{7\times 6\times 5\times 4}{4\times 3\times 2\times 1} = 35.$$

3. Seçilen 4 yere tek sayılar $1,3,5,7$ kural gereği yalnızca **1 farklı şekilde** (küçükten büyüğe) yerleştirilebilir.

4. Geriye kalan $7-4=3$ pozisyona çift sayılar yerleşmelidir. Bu 3 pozisyon için çift sayıların sırası da büyükten küçüğe $(6,4,2)$ olacak şekilde **1 farklı biçimde** zorunludur.

Çarpma kuralı uyarınca toplam geçerli dizilim sayısı:

$$\binom{7}{4}\times 1\times 1 = 35$$

olarak bulunur.

**Alternatif Bakış (Tekrarlı Permütasyon Sezgisi):** Dört tek sayıyı özdeş $T$ simgesiyle, üç çift sayıyı özdeş $C$ simgesiyle temsil edelim. $T,T,T,T,C,C,C$ harflerinin diziliş sayısı $\frac{7!}{4!3!} = 35$'tir. Herhangi bir dizilimde (örneğin $TCTTCCT$) soldan sağa ilk $T$'ye 1, ikincisine 3, üçüncüsüne 5, dördüncüsüne 7; ilk $C$'ye 6, ikincisine 4, üçüncüsüne 2 yazmak zorundayız. Dolayısıyla özdeş harf dizilimleri ile aranan sayılar arasında birebir eşleme (Bölüm 10) vardır. □

**En sık yapılan hata:** Çiftlerin ve teklerin sıralamasını da hesaba katıp sonucu yanlışlıkla $\binom{7}{4}\times 4!\times 3! = 7!$ ile karıştırmaktır. Sıralama önceden belirlenmişse o elemanlar arasındaki permütasyon serbestliği 1'e düşer.

### 37.1.2 İçerme-Dışarma İlkesi ile Kısıtlı Sayma

**Örnek 37.2.** *$A=\{1,2,3,4,5\}$ kümesinden $B=\{1,2,3\}$ kümesine tanımlanabilecek örten (surjective) fonksiyonların sayısını bulunuz.*

*Çözüm.* Bir fonksiyonun örten olması, değer kümesindeki her bir elemanın en az bir kez görüntü olarak seçilmesi demektir. Her elemanın seçildiği durumları doğrudan saymak yerine, Bölüm 8'de gördüğümüz içerme-dışarma ilkesinden yararlanarak örten olmayan durumları tüm durumlardan çıkarmak çok daha pratiktir. Değer kümesindeki en az bir eleman boşta kalırsa fonksiyon örten olamaz.

Tüm fonksiyonların evrensel kümesini $U$, $B$ kümesindeki $i$ elemanının görüntü kümesinde yer almadığı fonksiyonlar kümesini $A_i$ ($i \in \{1,2,3\}$) ile gösterelim.

1. **Evrensel kümenin boyutu $|U|$:** Tanım kümesindeki 5 elemanın her biri için $B$'de 3 seçenek vardır:
$$|U| = 3^5 = 243.$$

2. **Bir elemanın boşta kaldığı durumlar ($|A_i|$):** Belirli bir eleman (örneğin 1) seçilmezse, tüm elemanlar kalan 2 elemana gitmelidir:
$$|A_i| = 2^5 = 32.$$
$B$'de 3 eleman olduğundan $\binom{3}{1} = 3$ farklı durum vardır:
$$\sum_i |A_i| = \binom{3}{1} \times 2^5 = 3 \times 32 = 96.$$

3. **İki elemanın aynı anda boşta kaldığı durumlar ($|A_i \cap A_j|$):** Belirli iki eleman (örneğin 1 ve 2) seçilmezse, tüm 5 eleman kalan tek elemana gitmek zorundadır:
$$|A_i \cap A_j| = 1^5 = 1.$$
$\binom{3}{2} = 3$ ikili vardır:
$$\sum_{i<j} |A_i \cap A_j| = \binom{3}{2} \times 1^5 = 3 \times 1 = 3.$$

4. **Üç elemanın birden boşta kaldığı durumlar ($|A_1 \cap A_2 \cap A_3|$):** Fonksiyon tanımı gereği görüntü kümesi boş olamaz; 0 elemana giden fonksiyon sayısı $0^5 = 0$'dır.

İçerme-dışarma ilkesi gereğince, en az bir elemanı dışarıda bırakan fonksiyonların sayısı:

$$|A_1 \cup A_2 \cup A_3| = \sum |A_i| - \sum |A_i \cap A_j| + |A_1 \cap A_2 \cap A_3| = 96 - 3 + 0 = 93.$$

Dolayısıyla, örten fonksiyon sayısı:

$$|U| - |A_1 \cup A_2 \cup A_3| = 243 - 93 = 150$$

olarak hesaplanır. □

### 37.1.3 Çubuk-Ayraç (Stars and Bars) Yöntemi

**Örnek 37.3.** $x_1+x_2+x_3+x_4 = 18$ *denklemini sağlayan ve* $x_1 \geq 1$, $x_2 \geq 2$, $x_3 \geq 0$, $x_4 \geq 3$ *koşullarına uyan kaç farklı* $(x_1, x_2, x_3, x_4)$ *tam sayı dörtlüsü vardır?*

*Çözüm.* Bölüm 5'te ele aldığımız ayraç yöntemi, değişkenlerin negatif olmadığı ($y_i \geq 0$) durumlar için geçerlidir. Değişkenlerin alt sınırları birbirinden farklı olduğunda en doğal ilk hamle, değişken değiştirerek her birinin alt sınırını 0'a çekmektir. Bu işlem, ceplerinde belirli miktarda para bulunması gereken kişilere bu miktarı peşinen dağıtmaya benzer.

Yeni değişkenleri tanımlayalım:

$$\begin{aligned} y_1 = x_1 - 1 \geq 0 &\implies x_1 = y_1 + 1, \\ y_2 = x_2 - 2 \geq 0 &\implies x_2 = y_2 + 2, \\ y_3 = x_3 - 0 \geq 0 &\implies x_3 = y_3, \\ y_4 = x_4 - 3 \geq 0 &\implies x_4 = y_4 + 3. \end{aligned}$$

Bu ifadeleri orijinal denklemde yerine yazalım:

$$(y_1 + 1) + (y_2 + 2) + y_3 + (y_4 + 3) = 18$$

$$y_1 + y_2 + y_3 + y_4 + 6 = 18 \implies y_1 + y_2 + y_3 + y_4 = 12.$$

Şimdi problem, 12 özdeş nesneyi negatif olmayan 4 değişkene ($y_1, y_2, y_3, y_4 \geq 0$) dağıtma problemine dönüştü. Ayraç teoremine göre, $n$ özdeş nesneyi $k$ farklı kutuya paylaştırmak için $n$ nesne ve $k-1$ ayraç kullanılır:

$$\binom{n+k-1}{k-1} = \binom{12+4-1}{4-1} = \binom{15}{3}.$$

Kombinasyonu hesaplayalım:

$$\binom{15}{3} = \frac{15 \times 14 \times 13}{3 \times 2 \times 1} = 5 \times 7 \times 13 = 455.$$

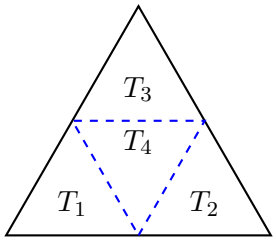
Demek ki koşulları sağlayan 455 farklı tam sayı çözümü vardır. □

**En sık yapılan hata:** Alt sınırları pozitif olan değişkenleri sıfıra çekerken sağ taraftan toplamı çıkarmak yerine toplamak veya sınırları yanlış eksiltmektir (örneğin $x_1 \geq 1$ için 1 eksiltmek yerine 0 almak).

### 37.1.4 Güvercin Yuvası İlkesi

**Örnek 37.4.** *Kenar uzunluğu* 1 *birim olan bir eşkenar üçgenin içerisine rastgele 5 nokta yerleştiriliyor. Bu noktalardan en az ikisi arasındaki mesafenin* $\leq \frac{1}{2}$ *birim olduğunu kanıtlayınız.*

*Çözüm.* Soruda verilen 5 nokta ve aranan $\leq \frac{1}{2}$ mesafesi, Bölüm 7'de incelediğimiz Güvercin Yuvası İlkesi'ni akla getirir. İlkeyi uygulayabilmek için üçgeni öyle bölgelere ayırmalıyız ki bölge sayısı nokta sayısından az (yani en fazla 4) olsun ve her bir bölgedeki herhangi iki nokta arasındaki mesafe en fazla $\frac{1}{2}$ birim kalabilsin.




Şekil 37.1: Eşkenar üçgenin 4 eş parçaya bölünmesi

Kenar uzunluğu 1 birim olan eşkenar üçgenin her kenarının orta noktasını birleştirelim. Bu işlem büyük üçgeni, kenar uzunluğu $\frac{1}{2}$ birim olan 4 adet eşkenar üçgene $(T_1, T_2, T_3, T_4)$ böler.

1. **Yuvalar:** 4 küçük eşkenar üçgen.
2. **Güvercinler:** Üçgen içine yerleştirilen 5 nokta.

Güvercin yuvası ilkesi uyarınca, 5 nokta 4 yuvaya dağıtıldığında en az bir küçük üçgenin içine en az $\lceil 5/4 \rceil = 2$ nokta düşmek zorundadır.

Kenar uzunluğu $\frac{1}{2}$ olan kapalı bir eşkenar üçgenin içindeki veya sınırındaki herhangi iki nokta arasındaki maksimum uzaklık (üçgenin çapı), iki köşe arasındaki uzaklıktır; o da tam olarak $\frac{1}{2}$ birimdir.

Dolayısıyla aynı küçük üçgene düşen bu iki nokta arasındaki mesafe en fazla $\frac{1}{2}$ birim olabilir. İspat tamamlanır. □

## 37.2 Ek Problemler: Sayılar Teorisi ve Problem Çözme Teknikleri

Sayılar teorisinde saf aritmetik manipülasyonlar tek başına yeterli olmaz; parite (Bölüm 17), invaryantlar (Bölüm 18), modüler kongrüanslar (Bölüm 14) ve uç değer ilkesi (Bölüm 19) gibi yöntemler birlikte kullanılır.

### 37.2.1 Modüler Aritmetik ve Bölünebilme

**Örnek 37.5.** $2^{2026} + 3^{2026}$ *sayısının 7 ile bölümünden kalan kaçtır?*

*Çözüm.* Büyük üslerle karşılaştığımızda iki temel yol öne çıkar: Bölüm 14'te gördüğümüz üzere tabanın kuvvetlerinin mod $m$'deki periyodunu belirlemek ya da

aynı bölümde ispatladığımız Fermat'nın Küçük Teoremi'nden yararlanmak. 7 asal sayı olduğundan, $\gcd(a, 7) = 1$ koşulunu sağlayan her $a$ için Fermat Teoremi $a^{7-1} = a^6 \equiv 1 \pmod 7$ denkliğini verir.

Üs olan 2026'yı mod 6'da inceleyelim:

$$2026 = 6 \times 337 + 4 \implies 2026 \equiv 4 \pmod 6.$$

Fermat'nın Küçük Teoremi uyarınca:

$$2^6 \equiv 1 \pmod 7 \quad \text{ve} \quad 3^6 \equiv 1 \pmod 7.$$

O halde:

$$2^{2026} = (2^6)^{337} \times 2^4 \equiv 1^{337} \times 2^4 \equiv 16 \pmod 7.$$

$16 = 2 \times 7 + 2 \implies 16 \equiv 2 \pmod 7$. Demek ki $2^{2026} \equiv 2 \pmod 7$.

Şimdi $3^{2026}$ terimini hesaplayalım:

$$3^{2026} = (3^6)^{337} \times 3^4 \equiv 1^{337} \times 3^4 \equiv 81 \pmod 7.$$

$81 = 11 \times 7 + 4 \implies 81 \equiv 4 \pmod 7$. Demek ki $3^{2026} \equiv 4 \pmod 7$.

Toplarsak:

$$2^{2026} + 3^{2026} \equiv 2 + 4 \equiv 6 \pmod 7.$$

Sonuç olarak, verilen sayının 7 ile bölümünden kalan 6'dır. □

### 37.2.2 Değişmezler (Invariants) ve Parite

**Örnek 37.6.** *Bir tahtada 1'den 20'ye kadar olan tam sayılar yazılıdır:* $1, 2, 3, \ldots, 20$. *Bir öğrenci her adımda tahtadan rastgele iki $a$ ve $b$ sayısını silip yerlerine $|a - b|$ sayısını yazıyor. Tahtada tek bir sayı kalana kadar bu işleme devam ediliyor. Son kalan sayı* 0 *olabilir mi?*

*Çözüm.* İşlem adımları serbesttir; hangi iki sayının silineceği önceden kestirilemez. Bu tür süreç problemlerinde her adımda korunan bir özellik, yani bir invaryant (Bölüm 18) aramak gerekir. Sayıların toplamının tekliğini veya çiftliğini, yani paritesini (Bölüm 17) inceleyelim.

İki sayı $a$ ve $b$ silinip yerine $|a - b|$ yazıldığında, tahtadaki sayıların toplamı nasıl değişir? Eski toplam $S_{\text{eski}}$ olsun. Yeni toplam:

$$S_{\text{yeni}} = S_{\text{eski}} - a - b + |a - b|.$$

Toplamdaki değişim miktarı $\Delta = S_{\text{eski}} - S_{\text{yeni}}$:

$$\Delta = a + b - |a - b|.$$

Biliyoruz ki her $a, b \in \mathbb{Z}$ için:

$$a + b \equiv a - b \equiv |a - b| \pmod 2.$$

Çünkü iki sayının farkı ile toplamının paritesi aynıdır ($(a+b)-(a-b)=2b\equiv 0 \pmod 2$).

Dolayısıyla:

$$\Delta = (a+b) - |a-b| \equiv |a-b| - |a-b| \equiv 0 \pmod 2.$$

Bu önemli bir tespittir: Her adımda tahtadaki sayıların toplamı çift bir sayı kadar azalır. Yani toplamın mod 2'deki değeri, başka bir deyişle **paritesi bir invaryanttır**.

Şimdi başlangıçtaki sayıların toplamını hesaplayalım:

$$S_0 = 1+2+3+\cdots+20 = \frac{20\times 21}{2} = 210.$$

$210 \equiv 0 \pmod 2$ (başlangıç toplamı çifttir).

Değişmezlik kuralı gereğince tahtada son kalan tek sayı $X$ ise:

$$X \equiv S_0 \equiv 0 \pmod 2$$

olmalıdır. 0 çift bir sayı olduğundan ($0 \equiv 0 \pmod 2$), parite argümanı 0'ı doğrudan elemez. Ancak soru "0 olabilir mi?" diye soruyor. Bir an durup daha derine bakalım:

Başlangıç kümesindeki tek sayıların adedine bakalım: $\{1,3,5,7,9,\ 11,\ 13,\ 15,\ 17,\ 19\} \implies$ 10 adet tek sayı vardır. İki sayı seçildiğinde:

- İkisi de tek ise: $|T_1 - T_2|$ = çift. Tek sayı adedi 2 azalır.
- Biri tek, biri çift ise: $|T - C|$ = tek. Tek sayı adedi değişmez.
- İkisi de çift ise: $|C_1 - C_2|$ = çift. Tek sayı adedi değişmez.

Görüldüğü üzere tek sayıların sayısı her adımda ya 0 ya da 2 azalır. Başlangıçta 10 adet tek sayı olduğundan, adımların sonucunda tek sayı adedi $10 \to 8 \to \cdots \to 0$ olabilir. Dolayısıyla son kalan sayı çift olmalıdır; 0 da çifttir.

Peki gerçekten 0 elde edilebilir mi? Sayıları ardışık çiftler halinde gruplayalım: $(1,2),(3,4),(5,6),\ldots,(19,20)$. Her çifte işlem uygularsak $|k-(k+1)| = 1$ elde edilir. Böylece tahtada 10 tane 1 kalır: $1,1,1,1,1,1,1,1,1,1$. Şimdi bu 1'leri ikişer ikişer eşleyip çıkarırsak: $|1-1| = 0$ elde edilir. 5 adet 0 kalır. Sıfırların farkı daima 0'dır: $|0-0| = 0$. Sonuç olarak tek kalan sayı 0 **olabilir**. □

**En sık yapılan hata:** Parite invaryantını bulunca incelemeyi bırakmaktır. Toplamın çift çıkması sonucun kesinlikle 0 olabileceğini tek başına kanıtlamaz; geçerli bir işlem dizisi (yapıcı örnek / kurgu) sunulmalıdır.

### 37.2.3 Uç Değer İlkesi (Extremal Principle)

**Örnek 37.7.** *Düzlemde sonlu sayıda nokta bulunmaktadır. Bu noktaların oluşturduğu tüm ikili mesafeler birbirinden farklıdır. Her nokta, kendisine en yakın olan noktaya bir ok çiziyor. Hiçbir noktanın beşten fazla ok alamayacağını kanıtlayınız.*

*Çözüm.* Bölüm 2'deki çelişkiyle ispat ve Bölüm 19'daki uç değer ilkesini birleştirelim. Bir noktanın en az 3 ok aldığını varsayıp bu noktaya gelen okların uçlarındaki noktalar arasındaki açıları ve mesafeleri inceleyelim.

Bir $P$ noktasına $A, B, C$ gibi en az üç farklı noktadan ok gelmiş olsun. Bu durum, $A$'ya en yakın noktanın $P$, $B$'ye en yakın noktanın $P$, $C$'ye en yakın noktanın $P$ olduğu anlamına gelir:

$$|AP| < |AB| \quad \text{ve} \quad |AP| < |AC|,$$
$$|BP| < |BA| \quad \text{ve} \quad |BP| < |BC|,$$
$$|CP| < |CA| \quad \text{ve} \quad |CP| < |CB|.$$

$P$ noktasının etrafındaki $A, B, C$ noktalarını $P$'ye göre saat yönünde sıralayalım. Bu üç noktadan en az iki komşu olanı arasındaki açıya bakalım. $P$ merkezli açılardan en az biri:

$$\angle APB + \angle BPC + \angle CPA = 360^\circ$$

olduğundan, güvercin yuvası mantığıyla (Bölüm 7) bu açılardan en az biri $\leq \frac{360^\circ}{3} = 120^\circ$ olmak zorundadır. Ancak daha da dar bir sınır vardır.

$A$ ve $B$ noktalarını ve $PAB$ üçgenini ele alalım: $A$'nın $P$'ye olan mesafesi $|AP|$, $B$'ye olan mesafesi $|AB|$'den küçüktür ($|AP| < |AB|$). Aynı şekilde $B$'nin $P$'ye mesafesi $|BP|$, $A$'ya mesafesinden küçüktür ($|BP| < |AB|$).

Dolayısıyla $PAB$ üçgeninde en uzun kenar kesinlikle $AB$ kenarıdır:

$$|AB| > |AP| \quad \text{ve} \quad |AB| > |BP|.$$

Bir üçgende en büyük kenarın karşısındaki açı en büyük açıdır. Dolayısıyla:

$$\angle APB > \angle PAB \quad \text{ve} \quad \angle APB > \angle PBA.$$

Bir üçgenin iç açıları toplamı $180^\circ$ olduğundan:

$$\angle APB > 60^\circ$$

olmak zorundadır! Çünkü eğer $\angle APB \leq 60^\circ$ olsaydı, diğer iki açıdan en az biri $\geq 60^\circ \geq \angle APB$ olurdu ki bu da $AB$'nin en büyük kenar olmasıyla çelişirdi.

Demek ki $P$'ye ok gönderen herhangi iki noktanın $P$ merkezinde oluşturduğu açı **kesinlikle** $60^\circ$**'den büyük** olmalıdır:

$$\theta_{i,j} > 60^\circ.$$

Eğer bir noktaya $k$ tane ok geliyorsa:

$$360^\circ = \sum_{i=1}^{k} \theta_i > k \times 60^\circ \implies k < \frac{360^\circ}{60^\circ} = 6.$$

$P$'ye ok gönderen iki nokta $A, B$ ise $|AB| > \max(|PA|, |PB|)$ olduğundan $AB$ üçgenin en uzun kenarıdır; en uzun kenarın karşısındaki açı en büyük açı olduğundan $\angle APB > 60^\circ$'dir.

Dolayısıyla $P$'ye $k$ ok geliyorsa bu okların $P$ merkezinde oluşturduğu $k$ açının her biri $60^\circ$'den büyüktür ve toplamları $360^\circ$'dir:

$$k \times 60^\circ < 360^\circ \implies k < 6 \implies k \leq 5.$$

Bu sınır keskindir: $P$'nin çevresine $72^\circ$ arayla yerleştirilen 5 nokta, yarıçapları birbirinden çok az farklı seçildiğinde hem $P$'yi en yakın komşu olarak seçer hem de aralarındaki tüm ikili mesafeler birbirinden farklı kalır; yani 5 ok gerçekten mümkündür. □

## 37.3 Ek Problemler: Algoritmalar ve C Programlama

Algoritmik sorularda teorik matematiksel analizin yanı sıra bellek modelleri, işaretçi aritmetiği (Bölüm 31), asimptotik karmaşıklık ($O$, Bölüm 21) ve sınır durumların C dilinde doğru kodlanması belirleyicidir.

### 37.3.1 Cevaba İkili Arama (Binary Search on Answer)

**Örnek 37.8.** *Elimizde $n$ adet tahta parçası vardır; $i$'inci tahtanın uzunluğu $L[i]$ birimdir. Bu tahtaları keserek eşit uzunlukta en az $k$ adet küçük tahta parçası elde etmek istiyoruz. Elde edilebilecek bir parçanın tam sayı cinsinden **en büyük** uzunluğunu $O(n \log(\max L))$ zaman karmaşıklığında bulan C fonksiyonunu yazınız.*

*Çözüm.* "En büyük uzunluk nedir?" sorusunu doğrudan çözmek yerine Bölüm 24'te gördüğümüz gibi bir karar problemine dönüştürelim: "Her bir parçanın uzunluğu $x$ birim olacak şekilde en az $k$ parça kesilebilir mi?" Eğer parça boyu $x$ iken $k$ parça kesilebiliyorsa, $x$'ten daha küçük her değer için de bu mümkündür. Eğer $x$ boyunda kesilemiyorsa, $x$'ten büyük hiçbir boyda kesilemez. Bu özellik arama uzayının **monoton** olduğunu gösterir. Monotonluk sağlandığı için doğrusal arama ($O(\max L)$) yerine ikili arama ($O(\log(\max L))$) uygulayabiliriz.

$x$ uzunluğunda parçalar kesildiğinde $L[i]$ boyundaki tahtadan $\lfloor L[i]/x \rfloor$ parça elde edilir. Toplam parça sayısı:

$$f(x) = \sum_{i=0}^{n-1} \left\lfloor \frac{L[i]}{x} \right\rfloor.$$

Karar fonksiyonumuz: $f(x) \geq k$ ise geçerli (true), aksi halde geçersiz (false).

İkili arama aralığımız $[1, \max(L)]$ olacaktır.

Aşağıdaki C kodunda bu mantık eksiksiz şekilde uygulanmıştır:

```c
#include <stdio.h>

/* Karar fonksiyonu: x uzunlugunda en az k parca cikar mi? */
int kesilebilir_mi(const int L[], int n, int k, int x) {
    if (x <= 0) return 1;
    long long toplam_parca = 0;
    for (int i = 0; i < n; i++) {
        toplam_parca += (L[i] / x);
        if (toplam_parca >= k) {
            return 1; /* Erken sonlandirma: overflow onler */
        }
    }
    return toplam_parca >= k;
}

int maksimum_parca_boyu(const int L[], int n, int k) {
    int sol = 1;
    int sag = 0;

    /* Arama araliginin ust sinirini belirle */
    for (int i = 0; i < n; i++) {
        if (L[i] > sag) {
            sag = L[i];
        }
    }

    int en_iyi = 0;
    while (sol <= sag) {
        int orta = sol + (sag - sol) / 2;

        if (kesilebilir_mi(L, n, k, orta)) {
            en_iyi = orta;       /* orta gecerli, daha buyugunu ara */
            sol = orta + 1;
        } else {
            sag = orta - 1;      /* orta cok buyuk, kucult */
        }
    }
    return en_iyi;
}
```

**Karmaşıklık Analizi:** Döngü $\log_2(\max L)$ adım sürer. Her adımda tüm dizi bir kez taranarak $O(n)$ işlem yapılır. Toplam karmaşıklık $O(n \log(\max L))$ olur. □

**En sık yapılan hata:** Orta noktayı hesaplarken `(sol + sag) / 2` yazmak

büyük sayılarda taşmaya (integer overflow) yol açabilir; bunun yerine `sol + (sag - sol) / 2` kullanılmalıdır. Ayrıca $x = 0$ durumu için sıfıra bölme hatasına karşı tedbir alınmalıdır.

### 37.3.2 Prefix Sums ve 2 Boyutlu Aralık Sorguları

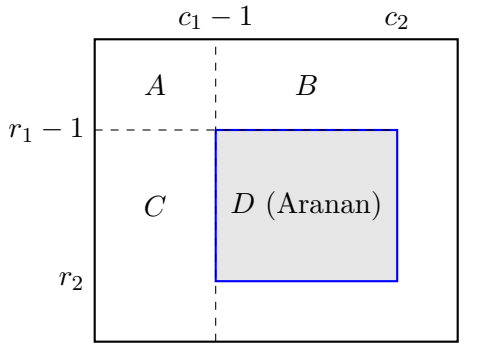
**Örnek 37.9.** *$N \times M$ boyutunda bir tam sayı matrisi veriliyor. Çok sayıda $(r_1, c_1)$ sol-üst ve $(r_2, c_2)$ sağ-alt koordinatları verilerek bu dikdörtgen bölgedeki sayıların toplamı sorgulanıyor. Her sorguyu $O(1)$ zamanda cevaplayabilmek için gereken ön hazırlık matrisinin formülünü çıkarınız ve C kodunu yazınız.*

*Çözüm.* Tek boyutta Bölüm 22'de incelediğimiz $pref[i] = pref[i-1] + A[i]$ formülüyle aralık toplamını $pref[R] - pref[L-1]$ şeklinde $O(1)$ sürede bulabiliyorduk. İki boyutta da aynı yaklaşımı Bölüm 8'deki içerme-dışarma ilkesiyle birleştirebiliriz.

Tanım: $P[i][j]$, matrisin $(1,1)$ noktasından $(i,j)$ noktasına kadar olan dikdörtgenin elemanları toplamı olsun. $(i,j)$ hücresi eklendiğinde:

$$P[i][j] = P[i-1][j] + P[i][j-1] - P[i-1][j-1] + Matris[i][j].$$

Burada $P[i-1][j-1]$ alanı hem üstteki hem soldaki dikdörtgende iki kez sayıldığı için bir kez çıkarılır.




Şekil 37.2: 2 Boyutlu Prefix Sums ve İçerme-Dışarma

Herhangi bir $[r_1 \ldots r_2] \times [c_1 \ldots c_2]$ sorgusunun toplamı:

$$\text{Toplam} = P[r_2][c_2] - P[r_1-1][c_2] - P[r_2][c_1-1] + P[r_1-1][c_1-1].$$

C dilinde 1-tabanlı indeksleme kullanarak sınır kontrollerini doğrudan sadeleştirebiliriz:

```
#define MAXN 1005
```

```
long long P[MAXN][MAXN];

/* On hazirlik: O(N * M) zaman ve alan */
void on_hazirlik(int N, int M, int Matris[MAXN][MAXN]) {
    for (int i = 1; i <= N; i++) {
        for (int j = 1; j <= M; j++) {
            P[i][j] = Matris[i][j]
                    + P[i - 1][j]
                    + P[i][j - 1]
                    - P[i - 1][j - 1];
        }
    }
}

/* Sorgu: O(1) zaman */
long long aralik_sorgusu(int r1, int c1, int r2, int c2) {
    return P[r2][c2]
         - P[r1 - 1][c2]
         - P[r2][c1 - 1]
         + P[r1 - 1][c1 - 1];
}
```

Bu sayede her sorgu tam olarak 3 toplama/çıkarma işlemiyle $O(1)$ zamanda yanıtlanır. □

### 37.3.3 İşaretçiler ve Bellek Takibi (Pointer Tracing)

**Örnek 37.10.** *Aşağıdaki C kod bloğu çalıştırıldığında ekrana basılacak çıktıyı adım adım bellek haritasını çizerek belirleyiniz.*

```
#include <stdio.h>

int main(void) {
    int a[] = {10, 20, 30, 40, 50};
    int *p = a;
    int *q = &a[3];

    printf("%ld\n", q - p);
    printf("%d\n", *p + *q);

    p += 2;
    printf("%d\n", *p);

    *p = *(p + 1) + 5;
    printf("%d\n", a[2]);

    return 0;
}
```

*Çözüm.* C dilinde dizi isimleri ilk elemanın adresine dönüşür (decay). İşaretçi aritmetiğinde (Bölüm 31) adresler bayt cinsinden değil, işaret edilen veri türünün boyutu ($sizeof(T)$) cinsinden ölçeklenir. Bellek durumunu adım adım takip edelim:

1. **İlk Durum:** `a` dizisi bellekte ardışık 5 tam sayı barındırır:
   - `a[0] = 10` (Adres: $A$)
   - `a[1] = 20` (Adres: $A + 1 \times 4$)
   - `a[2] = 30` (Adres: $A + 2 \times 4$)
   - `a[3] = 40` (Adres: $A + 3 \times 4$)
   - `a[4] = 50` (Adres: $A + 4 \times 4$)

   `p = a` $\implies$ $p$, `a[0]` elemanını göstermektedir. `q = &a[3]` $\implies$ $q$, `a[3]` elemanını göstermektedir.

2. **Birinci `printf`: `q - p`** İki işaretçi arasındaki fark, aralarındaki bayt farkı değil, **eleman sayısı** farkıdır:
   $$q - p = \&a[3] - \&a[0] = 3.$$
   Ekrana **3** yazılır.

3. **İkinci `printf`: `*p + *q`** İşaretçilerin gösterdiği değerler toplanır: `*p = a[0] = 10`, `*q = a[3] = 40`.
   $$10 + 40 = 50.$$
   Ekrana **50** yazılır.

4. **İşaretçi İlerlemesi: `p += 2`** $p$ işaretçisi 2 eleman ileri kaydırılır. Artık `p`, `a[2]` adresini gösterir. `printf("%d\{}n", *p)` $\implies$ `*p = a[2] = 30`. Ekrana **30** yazılır.

5. **Değer Ataması: `*p = *(p + 1) + 5`** `p` şu anda `&a[2]` adresindedir. `p + 1` ifadesi `&a[3]` adresini gösterir. `*(p + 1)` değeri `a[3] = 40`'tır. Böylece `*p = 40 + 5 = 45` olur. `*p` doğrudan `a[2]` hücresini temsil ettiğinden, `a[2]` değeri 45 olarak güncellenir. `printf("%d\{}n", a[2])` $\implies$ ekrana **45** yazılır.

Kodun nihai çıktısı satır satır şöyledir:

```
3
50
30
45
```

□

**En sık yapılan hata:** `q - p` farkını hesaplarken bayt farkını (örneğin 32-bit `int` için $3 \times 4 = 12$) cevap olarak yazmaktır. C standardında işaretçi farkı daima aradaki eleman adedini (`ptrdiff_t`) verir.

### 37.3.4 Rekürsiyon ve Geriye İz Sürme (Backtracking)

**Örnek 37.11.** *$n$ basamaklı, yalnızca $\{1, 2, 3\}$ rakamlarından oluşan ve yan yana gelen hiçbir iki rakamın aynı olmadığı tüm sayıları sözlük sırasına (lexicographical order) göre ekrana basan rekürsif (rekürsif) C fonksiyonunu yazınız.*

*Çözüm.* Bu problem bir durum ağacını dolaşma yapısındadır (Bölüm 26). Her adımda elimizdeki sayıya eklenebilecek geçerli rakamları deneriz. Bir önceki adımda eklenen rakam ne ise, mevcut adımda o rakam dışındaki seçenekler sırayla denenir. Sözlük sırası istendiği için denemeleri küçük rakamdan büyüğe ($1 \to 2 \to 3$) doğru yapmalıyız. Taban durum (base case) basamak sayısı $n$'e ulaştığında sayıyı ekrana yazdırmaktır.

C dilinde bunu parametrik bir dizi (karakter katarı) ile geriye iz sürme (backtracking) yaparak kurabiliriz:

```c
#include <stdio.h>

/* dizi: mevcut sayiyi tutar
   basamak: su an doldurulacak indeks (0'dan n-1'e)
   n: hedeflenen basamak sayisi */
void olustur(char dizi[], int basamak, int n) {
    /* Taban durum: n basamak dolduruldu */
    if (basamak == n) {
        dizi[n] = '\0'; /* Katar sonlandirici */
        printf("%s\n", dizi);
        return;
    }

    /* 1, 2, 3 rakamlarini kucukten buyuge dene */
    for (char rakam = '1'; rakam <= '3'; rakam++) {
        /* Yan yana ayni rakam gelmeme kosulu */
        if (basamak == 0 || dizi[basamak - 1] != rakam) {
            dizi[basamak] = rakam;      /* Secimi yap */
            olustur(dizi, basamak + 1, n); /* Derine in */
            /* Geri donuste ekstra temizlige gerek yoktur,
               cunku sonraki dongu ayni indeksi ezecektir. */
        }
    }
}

void tum_sayilari_yazdir(int n) {
```

```
    char dizi[50]; /* n < 50 varsayilmistir */
    olustur(dizi, 0, n);
}
```

**Ağaç Boyutu ve Karmaşıklık:** İlk basamak için 3 seçenek vardır. Sonraki her basamak için bir önceki basamaktan farklı 2 seçenek vardır. Toplam üretilen sayı adedi:

$$3 \times 2^{n-1}$$

olur. Her sayının basılması $O(n)$ sürdüğünden, toplam zaman karmaşıklığı $O(n \cdot 2^n)$ olur ve bu arama uzayı için asimptotik olarak optimaldir. □

## 37.4 Bölüm Özeti ve İpuçları Kılavuzu

Sınavda soru çözerken tıkandığınız anlarda aşağıdaki kontrol listesini zihninizden geçirebilirsiniz:

1. **Küçük Değerler Yöntemi:** $n = 1, 2, 3, 4$ için elle yazıp tablo oluşturun. Çoğu kombinatorik ve sayı teorisi sorusunda genel kural küçük örneklerin örüntüsünden ortaya çıkar.

2. **Simetri ve Tamlayıcı Sayma:** İstenen durumu doğrudan saymak karmaşıksa, tüm durumlardan istenmeyen durumları çıkarmayı (tümleyen küme, Bölüm 1 ve Bölüm 8) düşünün.

3. **Uç Değer ve Uç Durumlar:** Algoritmada dizinin boş olması, $k = 0$, tek eleman veya negatif sayılar gibi sınır durumları mutlaka test edin (Bölüm 19).

4. **Karmaşıklık Bütçesi:** $N \leq 10^5$ ise $O(N^2)$ çözümün zaman aşımı (TLE) alacağını göz önünde bulundurun; $O(N \log N)$ veya $O(N)$ arayışına girin (Bölüm 21). $N \leq 20$ ise $O(2^N)$ geri iz sürme kabul edilebilir çözümdür.

# Bölüm 38

# Ekler

Yarışma ortamında veya yoğun problem çözme kamplarında, ihtiyaç duyulan bir formülü, bir algoritmanın asimptotik karmaşıklığını ya da yabancı kaynaklarda sıkça geçen bir terimin Türkçe karşılığını hızla anımsamak gerekebilir. Bu bölüm, kitabın önceki bölümleri boyunca kurulan teorik çatının derli toplu bir başvuru kılavuzudur.

Burada listelenen formül ve karmaşıklık ifadelerinin dayanakları ilgili bölümlerde ayrıntılı biçimde ele alınmıştır. Bu eklerin amacı ispatları yinelemek değil, temel araçları doğrudan başvurulabilecek biçimde bir arada sunmaktır. Bu eklerde yalnızca kitapta anlatılan kavramlara yer verilmiştir; ileri veri yapıları (Fenwick ağacı, segment tree, sparse table gibi) ve ileri graf algoritmaları (Bellman-Ford, Floyd-Warshall gibi) bu kitabın kapsamı dışındadır.

## 38.1 Formül listesi

Sınav anında bir özdeşliği yanlış anımsamak (örneğin indisi 1 kaydırmak ya da çifte sayma düzeltmesini atlamak) çözümü bütünüyle geçersiz kılabilir. En sık başvurulan temel sonuçlar aşağıda tematik başlıklar altında derlenmiştir.

### 38.1.1 Kombinatorik ve Sayma

- **Permütasyon ve Kombinasyon (Bölüm 4 ve 5):** $n$ elemanlı bir kümeden $r$ elemanın sıralı seçimi $P(n,r)$, sırasız seçimi $\binom{n}{r}$ ile gösterilir:

$$P(n,r) = \frac{n!}{(n-r)!}, \qquad \binom{n}{r} = \frac{n!}{r!(n-r)!} = \frac{P(n,r)}{r!}$$

  Sınır değerler: $\binom{n}{0} = \binom{n}{n} = 1$ ve $r < 0$ ya da $r > n$ için $\binom{n}{r} = 0$.

- **Pascal Özdeşliği (Bölüm 5 ve 6):** Belirli bir elemanın seçilen grupta

bulunup bulunmadığına göre iki duruma ayırarak sayma:

$$\binom{n}{r} = \binom{n-1}{r-1} + \binom{n-1}{r}$$

- **Hokey Sopası Özdeşliği (Bölüm 6):** Pascal üçgeninde bir kenardan başlayıp çapraz inen terimlerin toplamı:

$$\sum_{i=r}^{n} \binom{i}{r} = \binom{n+1}{r+1}$$

- **Vandermonde Özdeşliği (Bölüm 6 ve 10):** Büyüklükleri $m$ ve $n$ olan iki ayrık kümeden toplam $r$ nesne seçme:

$$\sum_{k=0}^{r} \binom{m}{k}\binom{n}{r-k} = \binom{m+n}{r}$$

- **Binom Teoremi ve Katsayı Toplamları (Bölüm 6):**

$$(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^{n-k}y^k$$

$x = 1, y = 1$ koyulduğunda toplam alt küme sayısı:

$$\sum_{k=0}^{n} \binom{n}{k} = 2^n$$

$x = 1, y = -1$ koyulduğunda tek ve çift büyüklükteki alt kümelerin eşitliği:

$$\sum_{k=0}^{n} (-1)^k \binom{n}{k} = 0 \implies \sum_{k \text{ çift}} \binom{n}{k} = \sum_{k \text{ tek}} \binom{n}{k} = 2^{n-1}$$

- **Tekrarlı Kombinasyon / Ayraç Yöntemi (Bölüm 5):** $n$ özdeş nesnenin $k$ farklı kutuya dağıtılması:
  - Boş kutu kalabilirse (negatif olmayan tam sayı çözümleri, $x_i \geq 0$): $\binom{n+k-1}{k-1}$
  - Her kutuda en az bir nesne olacaksa ($x_i \geq 1$, $n \geq k$): $\binom{n-1}{k-1}$

- **İçerme-Dışarma İlkesi (Bölüm 8):** Sonlu $A_1, A_2, \ldots, A_n$ kümeleri için:

$$\left|\bigcup_{i=1}^{n} A_i\right| = \sum_{i=1}^{n} |A_i| - \sum_{1 \leq i < j \leq n} |A_i \cap A_j| + \sum_{1 \leq i < j < k \leq n} |A_i \cap A_j \cap A_k| - \cdots + (-1)^{n-1} |A_1 \cap \cdots \cap A_n|$$

- **Güvercin Yuvası İlkesi (Bölüm 7):** $n$ nesne $k$ yuvaya dağıtıldığında, en az bir yuvada en az $\lceil n/k \rceil$ nesne; en az bir yuvada ise en fazla $\lfloor n/k \rfloor$ nesne bulunur.

- **Katalan Sayıları (Bölüm 9):** Dengeli parantez dizilimleri, $n+1$ yapraklı ikili ağaçlar ve köşegeni aşmayan kafes yolları:

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1}, \qquad C_0 = 1,\ C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i}$$

  İlk terimler: $1, 1, 2, 5, 14, 42, 132, 429, 1430, \ldots$

- **Düzensizlik Sayısı / Deranjman (Bölüm 8):** Hiçbir elemanın kendi başlangıç konumunda yer almadığı permütasyonlar:

$$D_n = n! \sum_{k=0}^{n} \frac{(-1)^k}{k!} = (n-1)(D_{n-1} + D_{n-2}) = nD_{n-1} + (-1)^n$$

  $D_0 = 1, D_1 = 0, D_2 = 1, D_3 = 2, D_4 = 9, D_5 = 44, D_6 = 265, \ldots$

### 38.1.2 Sayılar Teorisi

- **Bölme Algoritması ve EBOB-EKOK (Bölüm 11 ve 13):** Her $a, b \in \mathbb{Z}^+$ için $a = bq + r$ $(0 \le r < b)$ eşitliğini sağlayan $q, r$ tam sayıları tektir.

$$\gcd(a,b) \cdot \operatorname{lcm}(a,b) = a \cdot b$$

- **Aritmetiğin Temel Teoremi ve Bölen Fonksiyonları (Bölüm 12):** $n > 1$ sayısının asal çarpan ayrışımı $n = p_1^{a_1} p_2^{a_2} \cdots p_k^{a_k}$ olsun:
  - Pozitif bölen sayısı: $d(n) = \prod_{i=1}^{k} (a_i + 1)$
  - Pozitif bölenler toplamı: $\sigma(n) = \prod_{i=1}^{k} \frac{p_i^{a_i+1} - 1}{p_i - 1}$

- **Fermat'nın Küçük Teoremi (Bölüm 14'te ispatlanır; Bölüm 11, 34 ve 37'de kullanılır):** $p$ asal ve $\gcd(a,p) = 1$ ise:

$$a^{p-1} \equiv 1 \pmod{p} \implies a^p \equiv a \pmod{p}$$

  Modüler ters hesabı: $p$ asal ise $a^{-1} \equiv a^{p-2} \pmod{p}$.

- **Legendre Formülü (Bölüm 12):** $n!$ çarpımındaki $p$ asalının en yüksek kuvveti $E_p(n!)$:

$$E_p(n!) = \sum_{k=1}^{\infty} \left\lfloor \frac{n}{p^k} \right\rfloor = \left\lfloor \frac{n}{p} \right\rfloor + \left\lfloor \frac{n}{p^2} \right\rfloor + \left\lfloor \frac{n}{p^3} \right\rfloor + \ldots$$

### 38.1.3 Cebirsel ve Aritmetik Toplamlar

- **Aritmetik ve Geometrik Dizi Toplamları:**

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}, \qquad \sum_{i=0}^{n} r^i = \frac{r^{n+1}-1}{r-1} \quad (r \neq 1)$$

- **Kuvvet Toplamları:**

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}, \qquad \sum_{i=1}^{n} i^3 = \left(\frac{n(n+1)}{2}\right)^2 = \left(\sum_{i=1}^{n} i\right)^2$$

### 38.1.4 Logaritma

- **Temel Kural: Çarpma → Toplama (Bölüm 16):** $a > 0$, $a \neq 1$ ve $x, y > 0$ için:

$$\log_a(xy) = \log_a x + \log_a y, \qquad \log_a \frac{x}{y} = \log_a x - \log_a y, \qquad \log_a x^k = k \log_a x$$

  Tanımdan doğrudan çıkan özel değerler: $\log_a 1 = 0$, $\log_a a = 1$ ve $a^{\log_a x} = x$.

- **Taban Değiştirme (Bölüm 16):**

$$\log_a x = \frac{\log_b x}{\log_b a}, \qquad \log_a b = \frac{1}{\log_b a}$$

  Bu eşitlikteki $\frac{1}{\log_b a}$ çarpanı $n$'den bağımsız bir sabittir; bu yüzden büyük $O$ gösteriminde $O(\log n)$ yazarken taban belirtilmez.

### 38.1.5 Graf Teorisi

- **El Sıkışma Lemması / Çifte Sayım (Bölüm 10 ve 28):** Köşe kümesi $V$, kenar kümesi $E$ olan her yönsüz grafta:

$$\sum_{v \in V} \deg(v) = 2|E|$$

  *Doğrudan sonuç:* Tek dereceli köşelerin sayısı daima çifttir.

- **Ağaçların Karakterizasyonu (Bölüm 28):** $|V| = n$ köşeli bir $G$ grafı için aşağıdaki özellikler denktir:

  1. $G$ bir ağaçtır (bağlantılı ve döngüsüzdür).
  2. $G$ bağlantılıdır ve $|E| = n - 1$.
  3. $G$ döngüsüzdür ve $|E| = n - 1$.

4. Herhangi iki köşe arasında tam olarak bir basit yol vardır.

- **Düzlemsel Graflar için Euler Formülü:** Bağlantılı bir düzlemsel çizimde köşe sayısı $V$, kenar sayısı $E$, yüz sayısı $F$ ise:

$$V - E + F = 2$$

Grafın $k$ bağlantılı bileşeni varsa: $V - E + F = k + 1$.

## 38.2 Terimler sözlüğü

Yarışma programlama kaynakları hem Türkçe hem de İngilizce terminolojiyi bir arada kullanır. TÜBİTAK sınav sorularında yerleşik Türkçe terimler tercih edilirken, çevrim içi platformlarda İngilizce terimler öne çıkar. Aşağıdaki sözlük, kitap boyunca kullanılan kavramların iki dildeki karşılıklarını derler.

| **Türkçe Terim** | **İngilizce Karşılığı** |
|---|---|
| alt taban | floor ($\lfloor x \rfloor$) |
| amortize edilmiş analiz | amortized analysis |
| ayraç yöntemi (çubuk-yıldız) | stars and bars |
| ayrık kümeler (Kruskal bağlamında, Bölüm 28) | disjoint set union (DSU) |
| bağlantılı bileşen (Bölüm 28) | connected component |
| basit yol | simple path |
| binom katsayısı | binomial coefficient |
| bölünebilme | divisibility |
| çap | diameter |
| çiftleme (eşleme) | matching |
| değişmez | invariant |
| derece | degree |
| derinlik öncelikli arama | depth-first search (DFS) |
| dinamik programlama | dynamic programming (DP) |
| düzensiz dizilim (Bölüm 8) | derangement |
| döngü (çevrim) | cycle |
| en büyük ortak bölen (EBOB) | greatest common divisor (GCD) |
| en küçük ortak kat (EKOK) | least common multiple (LCM) |
| en kısa yol | shortest path |
| genişlik öncelikli arama | breadth-first search (BFS) |
| graf | graph |
| güvercin yuvası ilkesi | pigeonhole principle |
| hızlı üs alma | binary exponentiation / modular exponentiation |
| hokey sopası özdeşliği | hockey-stick identity |
| topolojik sıralama (Bölüm 28) | topological sort |
| en kısa yol (Bölüm 28) | shortest path |
| iki yoldan sayma | double counting |
| ikili arama | binary search |
| binary heap (Bölüm 28'de Prim/Dijkstra bağlamında) | binary heap |
| içerme-dışarma ilkesi (Bölüm 8) | inclusion-exclusion principle |
| kalan sınıfı (modüler kongruans) | residue class (congruence) |
| logaritma (Bölüm 16) | logarithm |
| MST (Bölüm 27-28) | minimum spanning tree (MST) |
| karşıt durum / uç değer ilkesi (Bölüm 19) | extremal principle |
| prefix sums (Bölüm 22) | prefix sums |
| modüler ters eleman | modular inverse |
| öklid algoritması (Bölüm 13) | Euclidean algorithm |
| rekürsiyon (Bölüm 2 ve 26) | recursion |
| paylaştırılmış karmaşıklık | amortized complexity |
| rekürans bağıntısı | recurrence relation |
| sayaçla sıralama | counting sort |
| seyrek tablo | sparse table |
| taban aritmetiği | base representation |
| teklik-çiftlik (parite) | parity |
| tersine çalışma | working backwards |
| üst tavan | ceiling ($\lceil x \rceil$) |
| yarı-değişmez | monovariant |
| yönlendirilmiş döngüsüz graf | directed acyclic graph (DAG) |

| İngilizce Terim | Kitapta Kullanımı |
|---|---|
| Adjacency list / matrix (Bölüm 28) | adjacency list / matrix |
| BFS (Bölüm 28) | BFS (genişlik öncelikli arama) |
| Bitmask (Bölüm 15 ve 32) | bitmask |
| Bitwise (Bölüm 32) | bitwise |
| DFS (Bölüm 28) | DFS (derinlik öncelikli arama) |
| Greedy (Bölüm 27-28) | greedy |
| Logarithm (Bölüm 16) | logaritma |
| Memoization (Bölüm 26) | memoization |
| Modular arithmetic (Bölüm 14) | modüler aritmetik |
| Prefix sums (Bölüm 22) | prefix sums |
| Sieve of Eratosthenes (Bölüm 12) | Eratosthenes kalburu |
| Two pointers (Bölüm 22) | two pointers |

Bu tabloda yalnızca kitapta geçen İngilizce terimlere yer verilmiştir. Kitapta anlatılmayan yapılar (binary search tree, bipartite graph, LCA, monoton kuyruk, segment tree, bit manipulation, brute-force, divide and conquer, combinatorial proof, Euler totient, in-degree/out-degree, time/space complexity gibi ayrı başlıklar) bu tabloya alınmamıştır.