

Bölüm 5: Dizgi Algoritmaları (2)

Algoritmalar

Sözlüksel sıralama

Basamak sıralaması

Düzenli ifadeler

Sonlu durum makineleri

Levenshtein uzaklığı

En uzun ortak alt dizi

Sonek dizisi

Burrows–Wheeler dönüşümü

GİRİŞ

Öğrenme Çıktıları

Sıralama

Dizgileri sözlüksel sırayla karşılaştırmak; LSD ve MSD basamak sıralamasını adım adım uygulamak ve $O(W \cdot (N + R))$ maliyetini açıklamak.

Ayrıştırma

Düzenli ifadeler yazıp okumak; bir sonlu durum makinesini (DFA) tasarlamak, çizmek ve bir girdi üzerinde çalıştırmak.

Benzerlik

Levenshtein uzaklığını ve en uzun ortak alt diziye (LCS) dinamik programlama tablosuyla hesaplamak ve çözümü tablodan geri okumak.

Dönüşüm

Sonek dizisini kurmak ve aramada kullanmak; Burrows–Wheeler dönüşümünü hesaplamak ve tersine çevirmek.

Dizgi Algoritmaları: Dört Problem Ailesi

Bu sunum, dizgiler (string) üzerinde çalışan dört algoritma ailesini ele alır:

Sıralama (String Sorting)

Sözlüksel sıralama (lexicographic order): dizgiler karakter kodları karşılaştırılarak sözlükteki gibi dizilir. **Basamak sıralaması**: karşılaştırma yapmadan, her karakteri bir basamak gibi kullanarak gruplar.

Ayrıştırma (Parsing)

Düzenli ifadeler (regular expressions): bir arama örüntüsünü tanımlayan dizgi. **Sonlu durum makineleri** (FSM): girdiyi karakter karakter okuyup durum değiştiren otomatlar.

Benzerlik (Similarity)

Levenshtein uzaklığı: bir dizgiyi diğerine dönüştüren en az tek karakterlik düzenleme sayısı. **LCS**: görelî sırası korunan, ardışık olması gerekmeyen en uzun ortak alt dizi.

Dönüşüm (Transformation)

Sonek dizisi (suffix array): tüm soneklerin sıralı başlangıç indisleri. **Burrows–Wheeler dönüşümü** (BWT): döndürmelerin sıralanmasıyla elde edilen, tersine çevrilebilir dönüşüm; bzip2'nin ön adımı.

Sözlüksel Sıralama (Lexicographic Order)

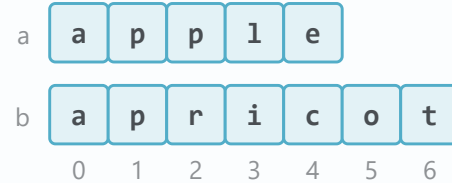
- **Sözlüksel (alfabetik) sıralama** (lexicographic order), öğeleri karakterlerin alfabadeki ya da kod tablosundaki **konumlarına** göre dizer.
- Karşılaştırma **ilk karakterlerle** başlar; eşitse bir **sonraki** karakterlere bakılır.
- İlk **farklı** karakter sonucu belirler: "apple" < "banana", çünkü 'a' < 'b'.
- Bir dizgi tükenirse **kısa olan (önek) önce** gelir: "car" < "cart".

Karşılaştırma	Karar veren
apple · banana	a < b (1. karakter)
apple · apricot	p < r (3. karakter)
car · cart	"car" tükendi
Zebra · apple	'Z' (90) < 'a' (97)

Dikkat: Bilgisayar "alfabe" olarak karakter kodlarını (ASCII / Unicode) kullanır: tüm büyük harfler küçük harflerden önce gelir; ç, ğ, ı, ö, ş, ü ise z'den sonra gelir. Türkçe sözlük sırası için `java.text.Collator` kullanılır.

Örnek: "apple" ve "apricot" Karşılaştırması

```
► int karsilastir(String a, String b) {  
  2  int n = Math.min(a.length(), b.length());  
  3  for (int k = 0; k < n; k++) {  
  4      char x = a.charAt(k), y = b.charAt(k);  
  5      if (x != y)  
  6          return x - y;    // ilk farklı karakter  
  7  }  
  8  return a.length() - b.length();  
  9 }
```



karsilastir("apple", "apricot") çağrıldı. Sonuç negatifse a önce, pozitifse b önce gelir (String.compareTo da böyle çalışır).

Adım 1 / 12

DEĞİŞKENLER

a = "apple"

b = "apricot"

n = -

k = -

x = -

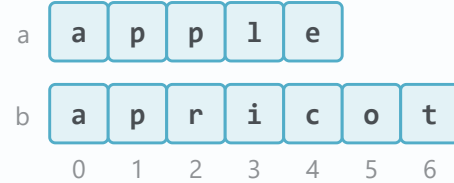
y = -

ÇIKTI



Örnek: "apple" ve "apricot" Karşılaştırması

```
1 int karsilastir(String a, String b) {  
2     int n = Math.min(a.length(), b.length());  
3     for (int k = 0; k < n; k++) {  
4         char x = a.charAt(k), y = b.charAt(k);  
5         if (x != y)  
6             return x - y;    // ilk farklı karakter  
7     }  
8     return a.length() - b.length();  
9 }
```



Kısa olanın uzunluğu kadar karakter karşılaştırılabilir: $n = \min(5, 7) = 5$.

Adım 2 / 12

DEĞİŞKENLER

a = "apple"

b = "apricot"

n = 5

k = -

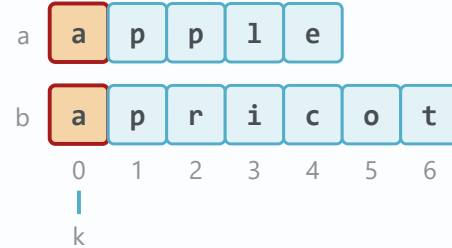
x = -

y = -

ÇIKTI

Örnek: "apple" ve "apricot" Karşılaştırması

```
1 int karsilastir(String a, String b) {  
2   int n = Math.min(a.length(), b.length());  
3   for (int k = 0; k < n; k++) {  
4     char x = a.charAt(k), y = b.charAt(k);  
5     if (x != y)  
6       return x - y;    // ilk farklı karakter  
7   }  
8   return a.length() - b.length();  
9 }
```



k = 0 < n: döngü sürer; 1. karakterlere bakılacak.

Adım 3 / 12

DEĞİŞKENLER

a = "apple"

b = "apricot"

n = 5

k = 0

x = -

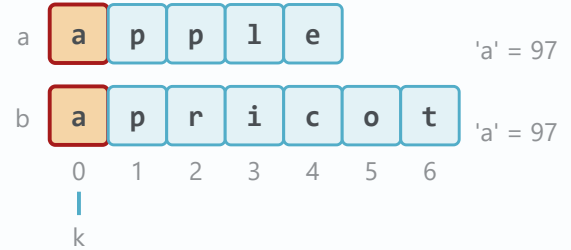
y = -

ÇIKTI



Örnek: "apple" ve "apricot" Karşılaştırması

```
1 int karsilastir(String a, String b) {  
2     int n = Math.min(a.length(), b.length());  
3     for (int k = 0; k < n; k++) {  
4         ▶ char x = a.charAt(k), y = b.charAt(k);  
5         if (x != y)  
6             return x - y;    // ilk farklı karakter  
7     }  
8     return a.length() - b.length();  
9 }
```



x = 'a' (97), y = 'a' (97) okundu.

Adım 4 / 12

DEĞİŞKENLER

a = "apple"

b = "apricot"

n = 5

k = 0

x = 'a'

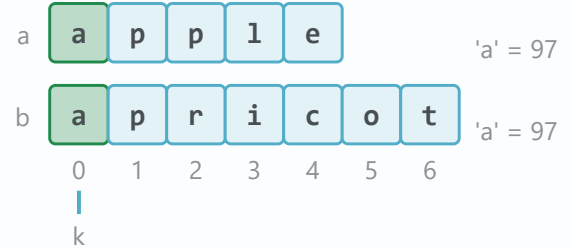
y = 'a'

ÇIKTI



Örnek: "apple" ve "apricot" Karşılaştırması

```
1 int karsilastir(String a, String b) {  
2     int n = Math.min(a.length(), b.length());  
3     for (int k = 0; k < n; k++) {  
4         char x = a.charAt(k), y = b.charAt(k);  
5         if (x != y)  
6             return x - y;    // ilk farklı karakter  
7     }  
8     return a.length() - b.length();  
9 }
```



'a' == 'a': karakterler eşit, karar verilemez;
sonraki karaktere geçilir.

Adım 5 / 12

DEĞİŞKENLER

a = "apple"

b = "apricot"

n = 5

k = 0

x = 'a'

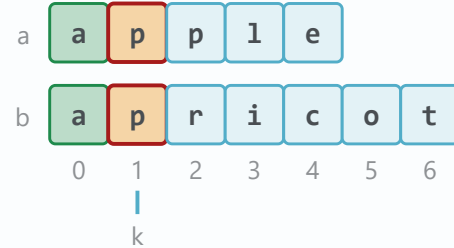
y = 'a'

ÇIKTI



Örnek: "apple" ve "apricot" Karşılaştırması

```
1 int karsilastir(String a, String b) {  
2   int n = Math.min(a.length(), b.length());  
3   for (int k = 0; k < n; k++) {  
4     char x = a.charAt(k), y = b.charAt(k);  
5     if (x != y)  
6       return x - y;    // ilk farklı karakter  
7   }  
8   return a.length() - b.length();  
9 }
```



k = 1 < n: döngü sürer; 2. karakterlere bakılacak.

Adım 6 / 12

DEĞİŞKENLER

a = "apple"

b = "apricot"

n = 5

k = 1

x = 'a'

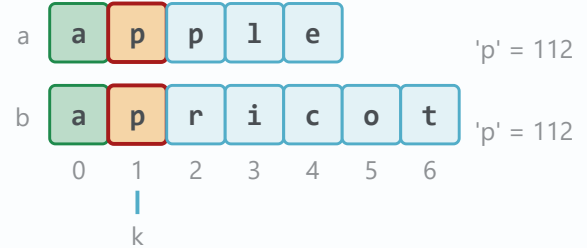
y = 'a'

ÇIKTI



Örnek: "apple" ve "apricot" Karşılaştırması

```
1 int karsilastir(String a, String b) {  
2     int n = Math.min(a.length(), b.length());  
3     for (int k = 0; k < n; k++) {  
4         ▶ char x = a.charAt(k), y = b.charAt(k);  
5         if (x != y)  
6             return x - y;    // ilk farklı karakter  
7     }  
8     return a.length() - b.length();  
9 }
```



x = 'p' (112), y = 'p' (112) okundu.

Adım 7 / 12

DEĞİŞKENLER

a = "apple"

b = "apricot"

n = 5

k = 1

x = 'p'

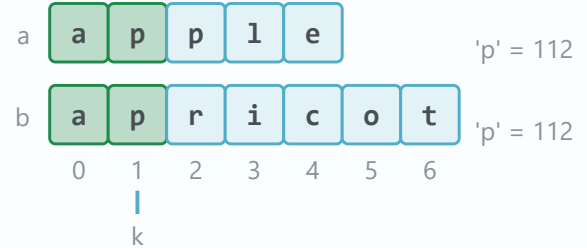
y = 'p'

ÇIKTI



Örnek: "apple" ve "apricot" Karşılaştırması

```
1 int karsilastir(String a, String b) {  
2     int n = Math.min(a.length(), b.length());  
3     for (int k = 0; k < n; k++) {  
4         char x = a.charAt(k), y = b.charAt(k);  
5         if (x != y)  
6             return x - y;    // ilk farklı karakter  
7     }  
8     return a.length() - b.length();  
9 }
```



'p' == 'p': karakterler eşit, karar verilemez;
sonraki karaktere geçilir.

Adım 8 / 12

DEĞİŞKENLER

a = "apple"

b = "apricot"

n = 5

k = 1

x = 'p'

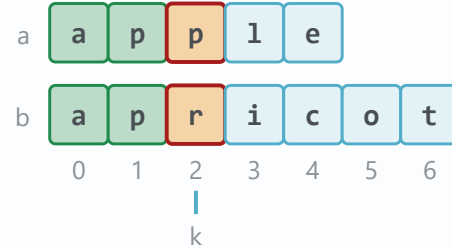
y = 'p'

ÇIKTI



Örnek: "apple" ve "apricot" Karşılaştırması

```
1 int karsilastir(String a, String b) {  
2   int n = Math.min(a.length(), b.length());  
3   for (int k = 0; k < n; k++) {  
4     char x = a.charAt(k), y = b.charAt(k);  
5     if (x != y)  
6       return x - y;    // ilk farklı karakter  
7   }  
8   return a.length() - b.length();  
9 }
```



k = 2 < n: döngü sürer; 3. karakterlere bakılacak.

Adım 9 / 12

DEĞİŞKENLER

a = "apple"

b = "apricot"

n = 5

k = 2

x = 'p'

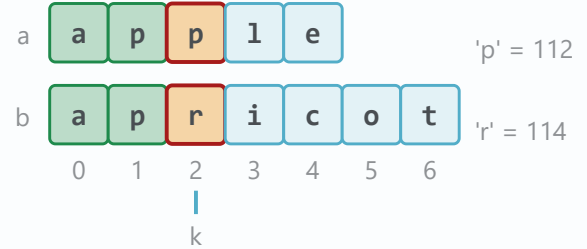
y = 'p'

ÇIKTI



Örnek: "apple" ve "apricot" Karşılaştırması

```
1 int karsilastir(String a, String b) {  
2   int n = Math.min(a.length(), b.length());  
3   for (int k = 0; k < n; k++) {  
4     ▶ char x = a.charAt(k), y = b.charAt(k);  
5     if (x != y)  
6       return x - y;    // ilk farklı karakter  
7   }  
8   return a.length() - b.length();  
9 }
```



x = 'p' (112), y = 'r' (114) okundu.

Adım 10 / 12

DEĞİŞKENLER

a = "apple"

b = "apricot"

n = 5

k = 2

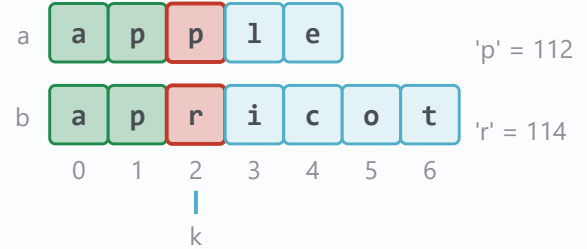
x = 'p'

y = 'r'

ÇIKTI

Örnek: "apple" ve "apricot" Karşılaştırması

```
1 int karsilastir(String a, String b) {  
2   int n = Math.min(a.length(), b.length());  
3   for (int k = 0; k < n; k++) {  
4     char x = a.charAt(k), y = b.charAt(k);  
5     if (x != y)  
6       return x - y; // ilk farklı karakter  
7   }  
8   return a.length() - b.length();  
9 }
```



'p' ≠ 'r': ilk farklı karakter bulundu (3. karakter).

Adım 11 / 12

DEĞİŞKENLER

a = "apple"

b = "apricot"

n = 5

k = 2

x = 'p'

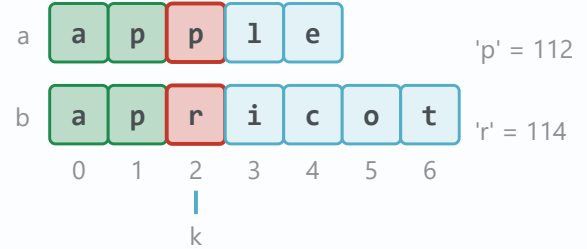
y = 'r'

ÇIKTI



Örnek: "apple" ve "apricot" Karşılaştırması

```
1 int karsilastir(String a, String b) {  
2   int n = Math.min(a.length(), b.length());  
3   for (int k = 0; k < n; k++) {  
4     char x = a.charAt(k), y = b.charAt(k);  
5     if (x != y)  
6       return x - y; // ilk farklı karakter  
7   }  
8   return a.length() - b.length();  
9 }
```



return $112 - 114 = -2 < 0 \rightarrow$ **apple, apricot'tan önce gelir.** Kalan karakterlere hiç bakılmaz.

Adım 12 / 12

DEĞİŞKENLER

a = "apple"

b = "apricot"

n = 5

k = 2

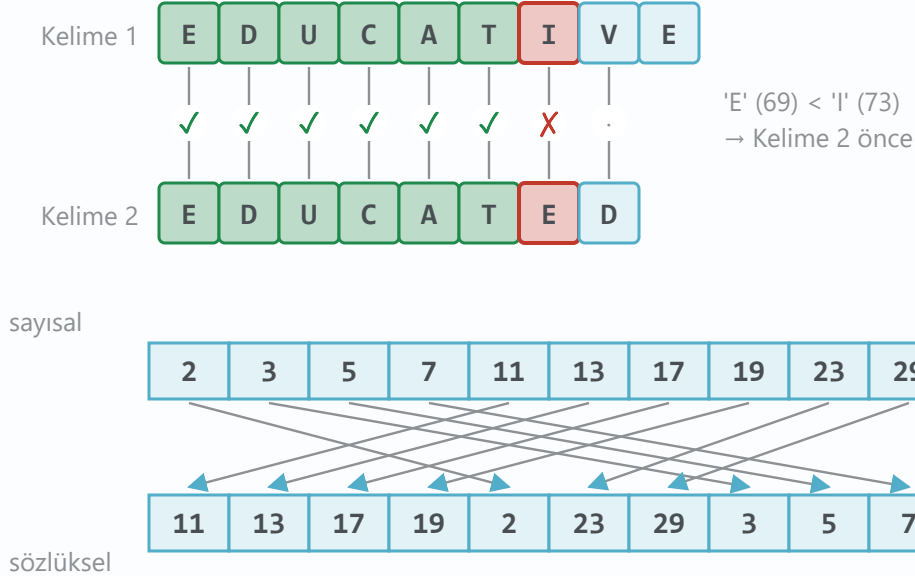
x = 'p'

y = 'r'

ÇIKTI

karsilastir("apple", "apricot") → -2

Sözlüksel Sıra: Kelimeler ve Sayılar



- Sayılar **dizgi olarak** sıralanırsa soldan sağa karakter karşılaştırılır: "11" < "2", çünkü '1' < '2'. Benzer şekilde 123, 45, 6, 789 sözlüksel olarak bu sıradadır.
- Sayısal sıra isteniyorsa sayılar tam sayıya çevrilmeli ya da soldan sıfırla aynı uzunluğa getirilmelidir ("02" < "11").

DİZGİ SIRALAMA

Karşılaştırmalı Dizgi Sıralama

```
void sozlukselSiralama(String[] a) {  
    int n = a.length;  
    for (int i = 0; i < n - 1; i++)  
        for (int j = i + 1; j < n; j++)  
            if (a[i].compareTo(a[j]) > 0) {  
                String gecici = a[i];  
                a[i] = a[j];  
                a[j] = gecici;  
            }  
}
```

- Her konuma, kalanların sözlükçe en küçüğü takaslarla getirilir.
- `compareTo` en fazla W karakter okur (W : en uzun dizgi).
- Zaman: $O(N^2)$ karşılaştırma $\times O(W) = \mathbf{O(N^2 \cdot W)}$.
- Birleştirme sıralaması ile $O(N \log N)$ karşılaştırma $\rightarrow \mathbf{O(W \cdot N \log N)}$.

Girdi	Çıktı
eabcdf, abcde, bcdea, cdeab, deabcb	abcde, bcdea, cdeab, deabcb, eabcdf
banana, orange, grape, apple, kiwi	apple, banana, grape, kiwi, orange

Karşılaştırmalı sıralamalar en az $\Omega(N \log N)$ karşılaştırma yapar. Basamak sıralaması karşılaştırma yapmadığı için bu alt sınıra tabi değildir.

Basamak Sıralaması (Radix Sort)

- **Basamak sıralaması** (radix sort), **karşılaştırmalı olmayan** bir sıralama algoritmasıdır.
- Anahtarlar basamaklara ayrılır: tam sayıda **rakamlar**, dizgide **karakterler**.
- Her geçişte, aynı konumda aynı değeri taşıyan öğeler **aynı kovaya** (bucket) konur.
- Her basamak için ayrı bir geçiş yapılır; geçişin kendisi **kararlı** (stable) bir saymalı sıralamadır (counting sort).

LSD (Least Significant Digit)

En az anlamlı basamaktan (en sağdaki) en anlamlıya doğru. En uzun dizginin uzunluğu W kadar geçiş; kısa dizgiler boş karakterle tamamlanır.

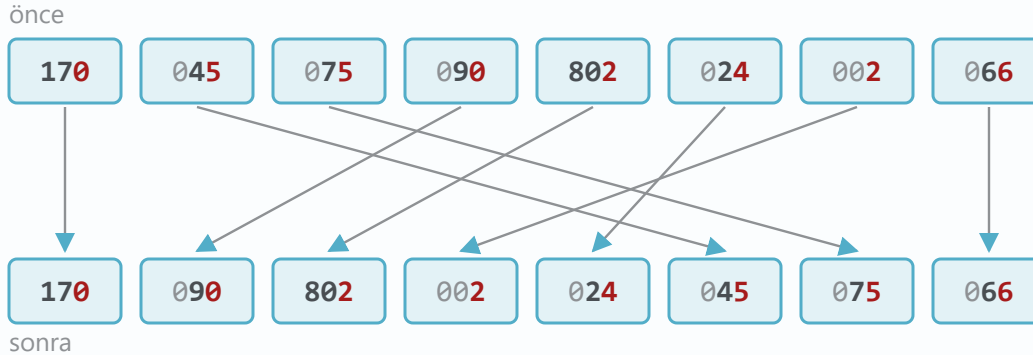
MSD (Most Significant Digit)

En soldaki karakterden başlar, her kovayı ayrı ayrı özyinelemeli sıralar. Değişken uzunluklu dizgilere uygundur.

Neden sağdan? Son geçiş en anlamlı basamağa göre yapılır; kararlılık sayesinde bu basamakta eşit olan öğeler önceki geçişlerin kurduğu (daha az anlamlı basamaklara göre) sırayı korur.

LSD Basamak Sıralaması: Sayısal Örnek

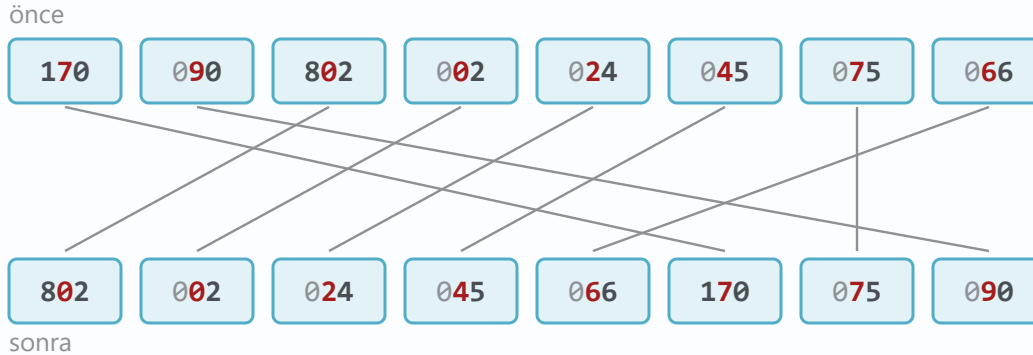
Veri kümesi: [170, 45, 75, 90, 802, 24, 2, 66]. Kısa sayılar soldan 0 ile tamamlanmış gibi düşünülür (soluk rakamlar).



1. geçiş (birler basamağı): anahtarlar (0, 5, 5, 0, 2, 4, 2, 6) → [170, 90, 802, 2, 24, 45, 75, 66]. Eşit anahtarlar (170, 90) giriş sırasını korur.

LSD Basamak Sıralaması: Sayısal Örnek

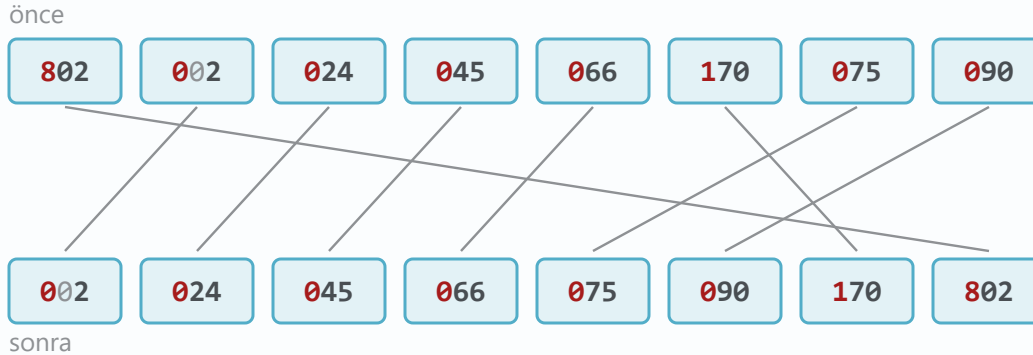
Veri kümesi: [170, 45, 75, 90, 802, 24, 2, 66]. Kısa sayılar soldan 0 ile tamamlanmış gibi düşünülür (soluk rakamlar).



2. geçiş (onlar basamağı): anahtarlar (7, 9, 0, 0, 2, 4, 7, 6) → [802, 2, 24, 45, 66, 170, 75, 90].

LSD Basamak Sıralaması: Sayısal Örnek

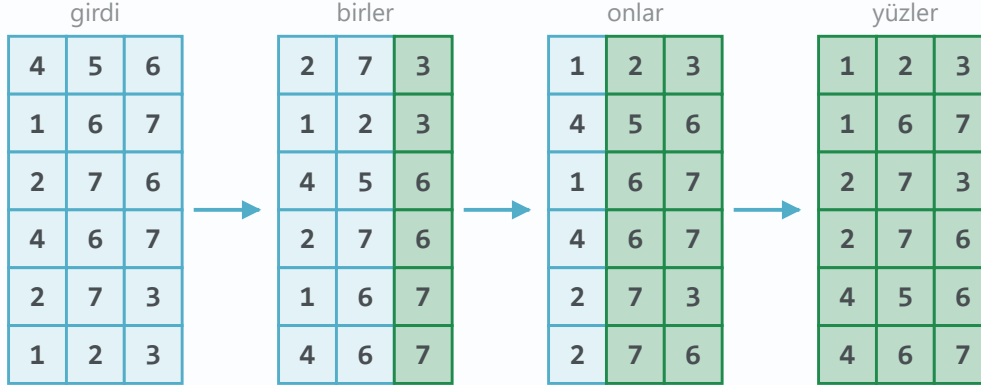
Veri kümesi: **[170, 45, 75, 90, 802, 24, 2, 66]**. Kısa sayılar soldan 0 ile tamamlanmış gibi düşünülür (soluk rakamlar).



3. geçiş (yüzler basamağı): anahtarlar (8, 0, 0, 0, 0, 1, 0, 0) → [2, 24, 45, 66, 75, 90, 170, 802]. Dizi sıralandı.

DİZGİ SIRALAMA

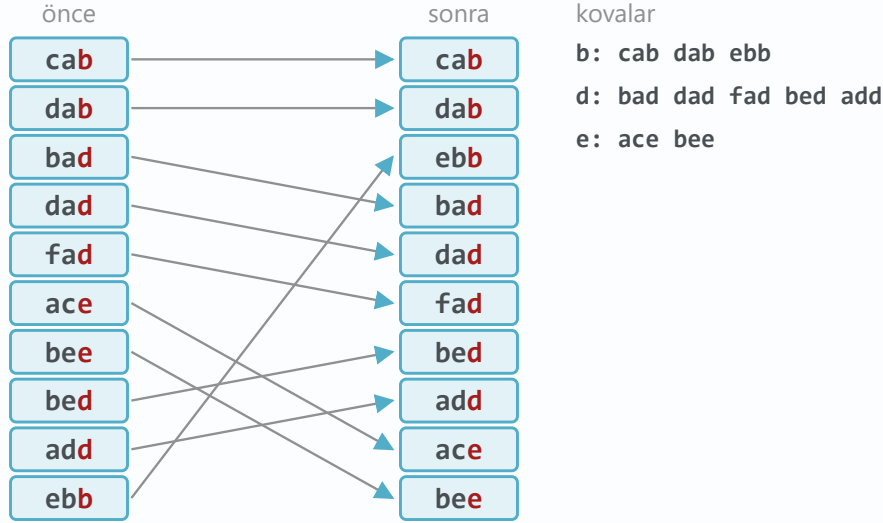
LSD Basamak Sıralaması: Kararlılık Neden Gerekli?



- Yeşil sütunlar, o ana kadar **sıralı hâle gelen son ekler**dir: k geçişten sonra dizi, son k basamağa göre sıralıdır.
- 2. geçişte 167 ile 467'nin onlar basamağı eşittir (6); 1. geçişten gelen sıra (7 = 7, giriş sırası) korunur.
- Alt sıralama **kararsız** olsaydı, örneğin 273 ile 276 son geçişte yer değiştirebilir ve sonuç bozulurdu.

LSD Basamak Sıralaması: Dizgiler

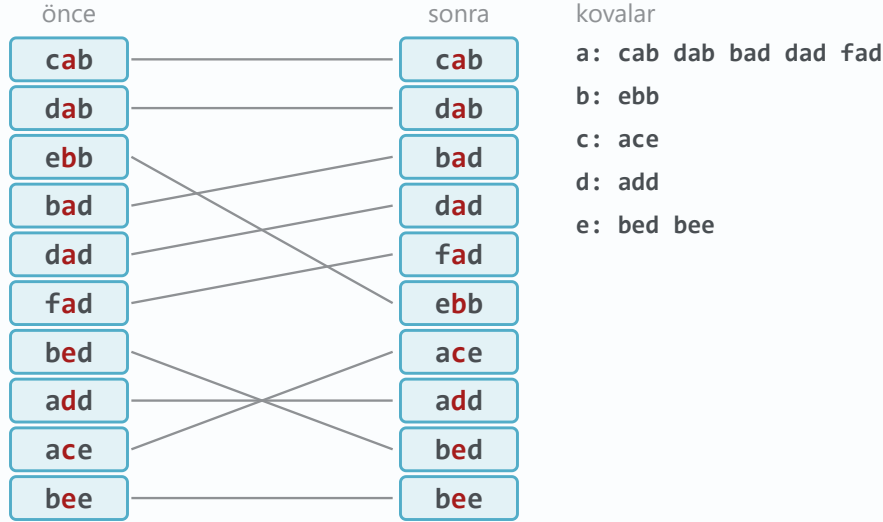
Simülatördeki örnek: **cab, dab, bad, dad, fad, ace, bee, bed, add, ebb** (W = 3, sağdan sola 3 geçiş).



1. geçiş (3. karakter): son karaktere göre kovalara dağıtılır; aynı kovadakiler (cab, dab) giriş sırasını korur.

LSD Basamak Sıralaması: Dizgiler

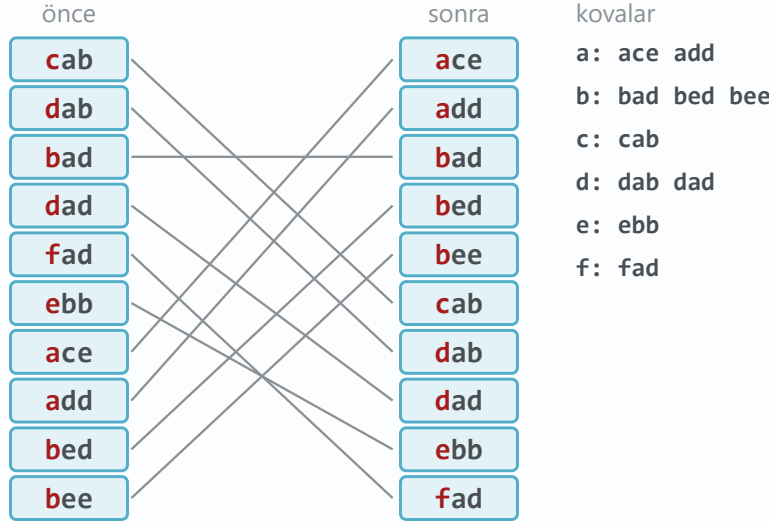
Simülatördeki örnek: **cab, dab, bad, dad, fad, ace, bee, bed, add, ebb** (W = 3, sağdan sola 3 geçiş).



2. geçiş (2. karakter): ortanca karaktere göre; 'a' kovalarında cab, dab, bad, dad, fad önceki sırayla (b < d) kalır.

LSD Basamak Sıralaması: Dizgiler

Simülatördeki örnek: **cab, dab, bad, dad, fad, ace, bee, bed, add, ebb** (W = 3, sağdan sola 3 geçiş).



3. geçiş (1. karakter): ilk karaktere göre; artık dizi sözlüksel olarak sıralıdır: ace, add, bad, bed, bee, cab, dab, dad, ebb, fad.

LSD Basamak Sıralaması: Java Kodu

```
void radixSiralama(String[] a) {
    int W = enUzun(a);           // en uzun dizgi
    for (int d = W - 1; d >= 0; d--)
        sayarakSiralama(a, d);    // sağdan sola
}

void sayarakSiralama(String[] a, int d) {
    int[] say = new int[256];     // ASCII
    for (String s : a) say[kar(s, d)]++;
    for (int r = 1; r < 256; r++) say[r] += say[r - 1];
    String[] gec = new String[a.length];
    for (int i = a.length - 1; i >= 0; i--) // kararlı
        gec[--say[kar(a[i], d)]] = a[i];
    System.arraycopy(gec, 0, a, 0, a.length);
}

int kar(String s, int d) {        // kısa dizgi → 0
    return d < s.length() ? s.charAt(d) : 0;
}
```

- **Sayma:** d. karakteri c olan dizgi sayısı `say[c]`.
- **Önek toplamı:** `say[c]` artık c kovanının **bittiği** konumu gösterir.
- **Yerleştirme** sondan başa yapılır; böylece eşit anahtarlar aynı sırada kalır (kararlılık).
- Uzunluğu yetmeyen dizgi için `kar` 0 döndürür: "car", "cart"tan önce gelir.

Bir geçiş $O(N + R)$, $R = 256$ alfabe boyutu; W geçiş $\rightarrow O(W \cdot (N + R))$.

DİZGİ SIRALAMA

MSD Basamak Sıralaması: Soldan Sağa

B	A	D	G	E	\0					
B	A	N	N	E	R	\0				
C	O	F	F	E	E	\0				
C	O	M	P	A	R	I	S	O	N	\0
C	O	M	P	U	T	E	R	\0		
M	I	D	N	I	G	H	T	\0		
W	A	N	D	E	R	\0				
W	A	R	D	R	O	B	E	\0		
W	O	R	K	E	R	\0				

1. Tüm dizgiler **ilk karaktere** göre kovalara ayrılır (B, C, M, W).
2. Her kova, **bir sonraki karaktere** göre ayrı ayrı, özyinelemeli sıralanır.
3. Tek öğeli kovada ya da dizgi sonunda (\0) durulur.

Yeşil hücreler, kelimenin yerini belirlemek için **okunması gereken** karakterlerdir (ayırt edici örnek). MSD yalnızca bunları inceler: MIDNIGHT için tek karakter yeter, COMPARISON / COMPUTER için beş.

Uzun ortak önekli dizgilerde MSD de tüm karakterleri okur. Küçük kovalarda R = 256 boyutlu sayma dizisi pahalı olduğundan pratikte küçük alt dizilere eklemeli sıralama uygulanır.

Zaman Karmaşıklığı ve Karşılaştırma

- LSD: W geçiş (W : en uzun dizginin uzunluğu), her geçiş N dizgi + R kova $\rightarrow O(W \cdot (N + R))$.
- Alfabe sabitse ($R = 256$) maliyet $O(W \cdot N)$ olur: en uzun dizginin uzunluğu $\times$ öge sayısı.

Yöntem	Zaman	Ek bellek	Kararlı
Yer değiştirmeli sıralama + compareTo	$O(N^2 \cdot W)$	$O(1)$	hayır
Birleştirme sıralaması + compareTo	$O(W \cdot N \log N)$	$O(N)$	evet
LSD basamak sıralaması	$O(W \cdot (N + R))$	$O(N + R)$	evet
MSD basamak sıralaması	$O(N \cdot W)$ en kötü	$O(N + W \cdot R)$	evet

Ne zaman basamak sıralaması? Anahtarlar kısa ve sabit uzunlukluysa (plaka, posta kodu, IP adresi, 3 harfli kodlar) LSD idealdir. Uzunluklar çok farklıysa MSD ya da karşılaştırmalı sıralama tercih edilir.

Düzenli İfadeler (Regular Expressions)

- **Düzenli ifade** (regular expression, regex): bir **arama örüntüsünü** tanımlayan dizgidir.
- Metinde örüntüye uyan **tüm parçaları** bulmak, doğrulamak, ayıklamak ya da değiştirmek için kullanılır.
- Metin işleme işlerini **otomatikleştirir**: günlük (log) analizi, form doğrulama, veri temizleme, derleyicilerde sözcük çözümleme (lexer).
- Kuramsal temeli: her düzenli ifadeye eşdeğer bir **sonlu durum makinesi** vardır (Kleene teoremi).

Örüntü	Metin	Eşleşmeler
<code>colou?r</code>	color colour	color, colour
<code>gr(a e)y</code>	gray grey groy	gray, grey
<code>\d+</code>	20.99 TL, 5 adet	20, 99, 5
<code>\bkod\b</code>	kod kodlama	yalnızca "kod"

Java'da: `java.util.regex (Pattern, Matcher)` ve `String.matches, replaceAll, split`.
Java dizgisinde ters bölü iki kez yazılır: `"\\d+"`.

Temel Operatörler ve Kavramlar

Op.	Anlamı	Örnek	İçinde bulunur	Bulunmaz
.	Herhangi bir tek karakter (yeni satır hariç)	a.c	abc, a7c	ac
*	Önceki öğenin 0 ya da daha fazla tekrarı	ab*c	ac, abbc	adc
+	Önceki öğenin 1 ya da daha fazla tekrarı	ab+c	abc, abbc	ac
?	Önceki öğenin 0 ya da 1 kez geçmesi	ab?c	ac, abc	abbc
[]	Kümedeki karakterlerden biri	[ck]ar	car, kar	bar
^	Dizginin başı	^ab	abc	cab
\$	Dizginin sonu	ab\$	cab	abc
()	Gruplama; alt ifade tanımlar	^(ab)+\$	ab, abab	aba

Sütunlar, dizginin içinde örüntüye uyan bir parça bulunup bulunmadığını gösterir

(`Matcher.find`). `*` , `+` , `?` yalnızca hemen önlerindeki öğeye (karakter, küme ya da grup) uygulanır.

Karmaşık Operatörler ve Özel Karakterler

İfade	Anlamı
<code>a</code>	tek bir 'a' karakteri
<code>abc</code>	tam olarak "abc"
<code>{ }</code>	belirli sayıda tekrar: <code>\d{4}</code>
<code> </code>	seçenekler: <code>kedi köpek</code>
<code>\b</code>	kelime sınırı (başı ya da sonu)
<code>\d</code>	bir rakam: [0-9]
<code>\w</code>	bir kelime karakteri : [a-zA-Z0-9_]
<code>\s</code>	boşluk karakteri: boşluk, tab, yeni satır

- Büyük harfli biçimler **tümleyendir**: `\D` rakam olmayan, `\W` kelime karakteri olmayan, `\S` boşluk olmayan karakter.
- Özel karakterin kendisi için **kaçış** gerekir: `\.` nokta, `\+` artı, `\C` parantez.
- `\w` tek bir karakterle eşleşir; bir kelime için `\w+` yazılır.

Java'da `\w` varsayılan olarak yalnızca ASCII harfleri kapsar: "şeker" içindeki 'ş' eşleşmez. Türkçe için `\p{L}` ya da `Pattern.UNICODE_CHARACTER_CLASS` kullanılır.

Karakter Sınıfları

Karakter sınıfları

[abc]	'a', 'b' ya da 'c'
[a-z]	herhangi bir küçük harf
[A-Z]	herhangi bir büyük harf
[0-9]	herhangi bir rakam
[a-zA-Z_]	harf ya da alt çizgi

Tümleyen karakter sınıfları

[^abc]	'a', 'b', 'c' dışındaki karakter
[^0-9]	rakam olmayan karakter
[^0-9]+	"ab12cd" → "ab", "cd"

- `^` köşeli parantezin **içinde ve başında** "değil" anlamına gelir; dışarıda dizginin başıdır.
- Küme içinde `.`, `*`, `+` özel değildir: `[.+]` nokta ya da artı ile eşleşir. Tire (`-`) aralık bildirir; kendisi için başa ya da sona yazılır.
- `[a-z]` aralığı karakter kodlarına dayanır; ç, ğ, ı, ö, ş, ü bu aralıkta **değildir**.

Yinelenen Karakterler ve Gruplama

Niceleyiciler (quantifiers)

<code>a*</code>	sıfır ya da daha fazla 'a'
<code>a+</code>	bir ya da daha fazla 'a'
<code>a?</code>	sıfır ya da bir 'a'
<code>a{3}</code>	tam olarak üç 'a'
<code>a{2,4}</code>	iki ile dört arası 'a'
<code>a{2,}</code>	en az iki 'a'

Gruplama

<code>(abc)</code>	tam olarak "abc"; grup olarak yakalanır
<code>(a b c)</code>	'a', 'b' ya da 'c'; grup olarak yakalanır
<code>(ab)+</code>	"ab", "abab", ...

Yakalanan grup Java'da `m.group(1)` ile okunur.

Açgözlü ve tembel: `a{2,4}` , "aaaaa" içinde mümkün olan en uzun parçayı (aaaa) alır.

Niceleyiciden sonra `?` yazılırsa en kısa eşleşme seçilir: `"<.+?>"` , `"<b>x</b>"` içinde yalnızca `"<b>"` ile eşleşir.

İleri ve Geri Bakış (Lookahead, Lookbehind)

Örüntü	Adı	Eşleşme koşulu	"ab ac ba"
<code>a(?=b)</code>	ileri bakış (lookahead)	'a', ardından 'b' geliyorsa	<u>a</u> b ac ba
<code>a(?!b)</code>	olumsuz ileri bakış	'a', ardından 'b' gelmiyorsa	ab <u>a</u> c <u>b</u> a
<code>(?<=b)a</code>	geri bakış (lookbehind)	'a', önünde 'b' varsa	ab ac <u>b</u> a
<code>(?<!b)a</code>	olumsuz geri bakış	'a', önünde 'b' yoksa	<u>a</u> b <u>a</u> c ba

- Bakış ifadeleri bir konumu **sınar**, karakter **tüketmez**: `a(?=b)` eşleşmesi yalnızca "a"dır, 'b' eşleşmeye dahil olmaz.
- Örnek: `^(?=.*\d)(?=.*[A-Z]).{8,}$` en az bir rakam ve bir büyük harf içeren, en az 8 karakterlik parolayı doğrular.

DÜZENLİ İFADELER

Örnek: Doğrulama İfadeleri

Amaç	Düzenli ifade
E-posta	<code>^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}\$</code>
Telefon	<code>^(+\d{1,3}[-]?)?\d{10}\$</code>
URL	<code>^https?:\/\/[^\s/\$.?\#]\.[^\s]*\$</code>

E-posta

[...] + kullanıcı adı, @, alan adı, \. gerçek nokta, en az 2 harfli uzanti. ✓ ali.veli@giresun.edu.tr
X ali@tr X a b@x.com

Telefon

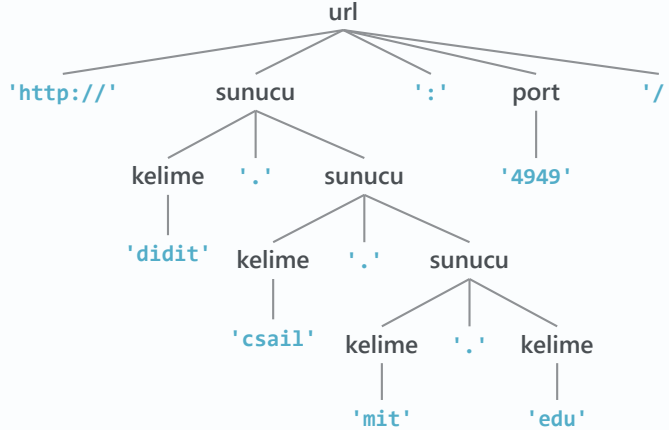
İsteğe bağlı (...) ? ülke kodu: + ve 1–3 rakam, ardından tire ya da boşluk; sonra tam 10 rakam.
✓ +90 5321234567 ✓
5321234567 X 0532 123 45 67

URL

s? http ya da https; ilk karakter boşluk, /, \$, ., ?, # olamaz; sonra boşluksuz karakterler. ✓
https://giresun.edu.tr X
ftp://x.com X "http:// x"

Bu ifadeler pratik yaklaşımlardır; standartların (RFC 5322 e-posta, RFC 3986 URI) tamamını kapsamaz.

Ayrıştırma Ağacı: URL Örneği



`http://didit.csail.mit.edu:4949/`

için dilbilgisi (grammar):

```
url    ::= 'http://' sunucu
        ':' port '/'
sunucu ::= kelime '.' sunucu
        | kelime
kelime ::= [a-z]+
port   ::= [0-9]+
```

Bu dilbilgisi özyinelemeli görünse de kuyruk özyinelemesidir; eşdeğer düzenli ifade:

`http://([a-z]+\.)*[a-z]+:[0-9]+/`. Dengeli parantez gibi **iç içe** yapılar ise düzenli ifadeyle tanımlanamaz; bağlamdan bağımsız dilbilgisi ve ayrıştırıcı (parser) gerekir.

Java'da Düzenli İfadeler: Pattern ve Matcher

```
String girdi = "Ad: Ali Veli, Yaş: 30, E-posta: ali.veli@ornek.com";
Pattern p = Pattern.compile("Yaş: (\\d+),"); // derle
Matcher m = p.matcher(girdi);
if (m.find()) { // ilk eşleşmeyi bul
    int yas = Integer.parseInt(m.group(1)); // 1. grup: "30"
    System.out.println("Yaş: " + yas);
}
```

Örüntü	group(1)	Açıklama
Ad: (.*) ,	Ali Veli	tembel .*? : ilk virgülde durur
Yaş: (\d+) ,	30	bir ya da daha çok rakam
E-posta: (.*)\$	ali.veli@ornek.com	satır sonuna kadar

- Aynı iş düzenli ifade kullanmadan, `indexOf` ve `substring` ile de yapılabilir: başlangıç işareti (ör. "Yaş: ") bulunur, bitiş işaretine (",") kadar kesilir.
- `find()` metnin **bir parçasını**, `matches()` **tamamını** eşleştirmeye çalışır.

Deneyin: Düzenli İfade Sınavı

Örüntü

☐ büyük/küçük harf duyarlı

Metin

\d+

colou?r

e-posta

a(?=b)

gruplar

Fiyat 20.99 TL, adet 5, kod A7

4 eşleşme bulundu.

Tarayıcının JavaScript düzenli ifade motoru kullanılır; söz dizimi Java ile büyük ölçüde aynıdır. Eşleşmeler vurgulanır, gruplar listelenir.

Sonlu Durum Makineleri (Finite State Machines)

- **Sonlu durum makinesi** (finite state machine, FSM): bir dizgiyi karakter karakter işleyen **matematiksel model**.
- Sonlu sayıda durumu vardır; bellek yalnızca **şu anki durumdur**.
- Basit ve anlaşılması kolaydır, yine de karmaşık dizgi analizi problemlerini (sözcük çözümleme, protokol, örüntü arama) ele alabilir.

Biçimsel tanım: $M = (Q, \Sigma, \delta, q_0, F)$

Durumlar Q (states)

Makinenin bulunabileceği farklı durumlar.

Alfabe Σ (alphabet)

Kabul edilen girdi sembollerinin kümesi.

Başlangıç durumu q_0

İşlemenin başladığı durum.

Geçişler δ (transitions)

$\delta(\text{durum}, \text{sembol}) \rightarrow \text{yeni durum}$.

Kabul durumları $F \subseteq Q$

Dizgi bittiğinde makine bunlardan birindeyse dizgi **kabul** edilir.

FSM Türleri: DFA, NFA, ϵ -NFA

DFA

Deterministik sonlu otomat (Deterministic Finite Automaton): her (durum, sembol) çifti için **tam olarak bir** geçiş vardır. Çalıştırması basittir: $O(n)$.

NFA

Deterministik olmayan sonlu otomat: bir (durum, sembol) çifti için **sıfır, bir ya da birden fazla** geçiş tanımlanabilir. Makine aynı anda bir durum kümesinde bulunur.

ϵ -NFA

Sembol okumadan yapılan **boş geçişlere** (epsilon, ϵ) izin veren NFA. Düzenli ifadeden doğrudan kurulur (Thompson yapısı).



- Üçü de **aynı dilleri** (düzenli diller) tanıır: her NFA, alt küme kurulumu (subset construction) ile bir DFA'ya çevrilir; ancak durum sayısı en kötü 2^n 'e çıkabilir.
- grep, lex/flex ve derleyici sözcük çözümleyicileri bu zinciri kullanır.

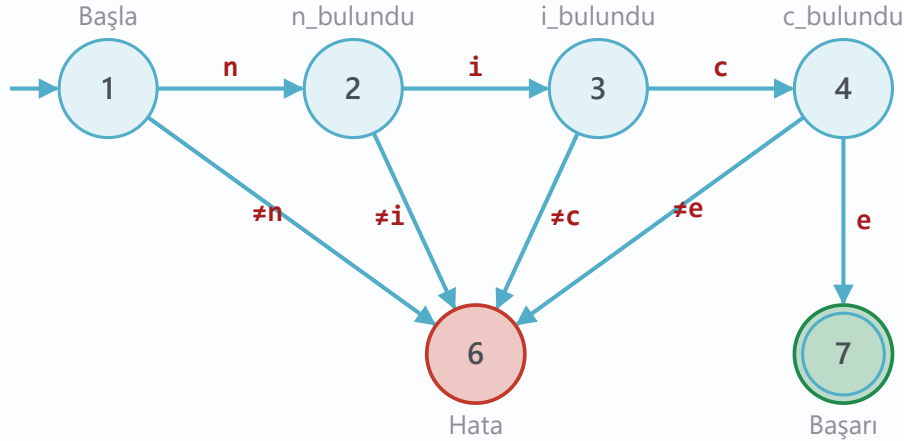
Kullanımı: DFA Simülasyonu

1. **Başlangıç** durumuna geç.
2. Her karakter için durumu **geçiş tablosundan** güncelle.
3. Dizgi bitince son durumun **kabul durumu** olup olmadığını denetle.

```
boolean kabulEder(int[][] gecis, boolean[] kabul,
                  String s) {
    int durum = 0;                // başlangıç durumu
    for (char c : s.toCharArray())
        durum = gecis[durum][c - '0']; // tek geçiş
    return kabul[durum];          // son durum kabul mü?
}
```

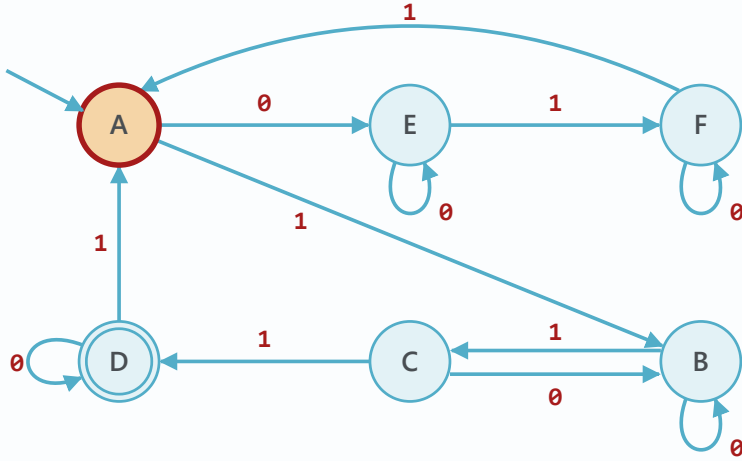
Geçiş tablosu $\text{int} [|Q|] [|\Sigma|]$ boyutundadır. Her karakter için bir tablo okuması yapılır: n uzunluklu dizgi **$O(n)$** zamanda, geri dönmeden (backtracking olmadan) işlenir.

Örnek: "nice" Kelimesini Tanıyan Makine



- "nice" kelimesini tanıyan makine: her durum, **şimdiye kadar okunan öneki** temsil eder (ε, n, ni, nic, nice).
- Beklenen karakter gelmezse (≠n, ≠i, ...) **Hata** durumuna gidilir ve orada kalınır (tuzak durum, trap state).
- "nice" → 1 → 2 → 3 → 4 → 7: kabul. "nine" → 1 → 2 → 3 → 6: ret (üçüncü karakter 'n' ≠ 'c').

Örnek: İkili Alfabe Üzerinde DFA



Başlangıç: durum **A**. Girdi: **10011**.

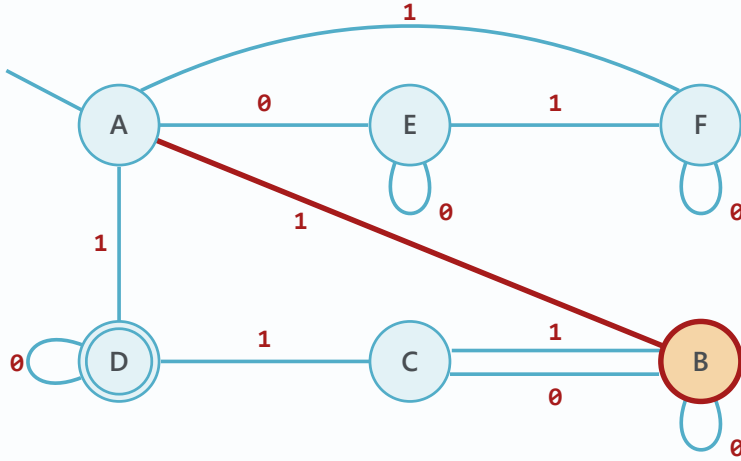
Adım 1 / 6

δ	0	1
$\rightarrow A$	E	B
B	B	C
C	B	D
D^*	D	A
E	E	F
F	F	A

$\rightarrow$ başlangıç, * kabul.

Her hücrede tek durum: **DFA**.

Örnek: İkili Alfabe Üzerinde DFA



1. sembol **1**: $\delta(A, 1) = \mathbf{B}$.

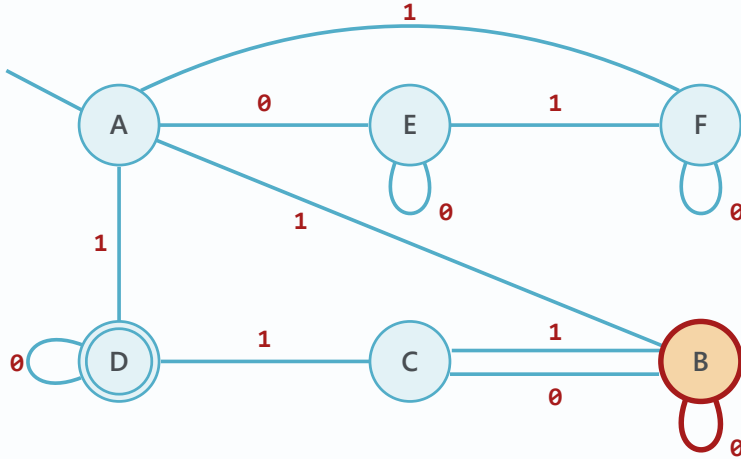
Adım 2 / 6

δ	0	1
$\rightarrow A$	E	B
B	B	C
C	B	D
D^*	D	A
E	E	F
F	F	A

$\rightarrow$ başlangıç, * kabul.

Her hücrede tek durum: **DFA**.

Örnek: İkili Alfabe Üzerinde DFA



2. sembol **0**: $\delta(B, 0) = \mathbf{B}$.

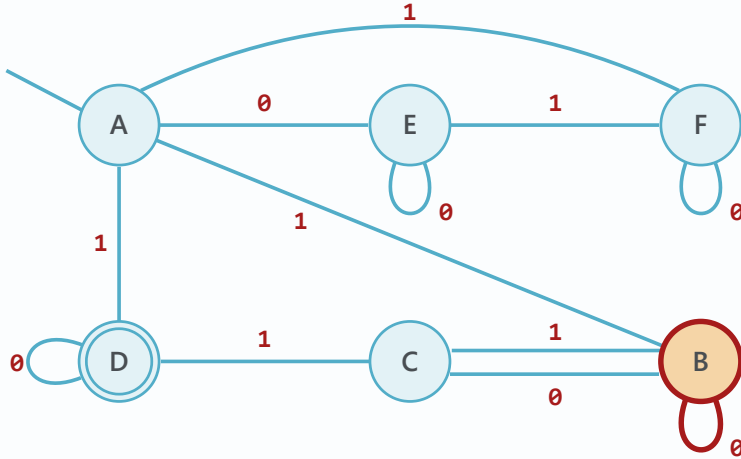
Adım 3 / 6

δ	0	1
$\rightarrow A$	E	B
B	B	C
C	B	D
D^*	D	A
E	E	F
F	F	A

$\rightarrow$ başlangıç, * kabul.

Her hücrede tek durum: **DFA**.

Örnek: İkili Alfabe Üzerinde DFA



3. sembol **0**: $\delta(B, 0) = \mathbf{B}$.

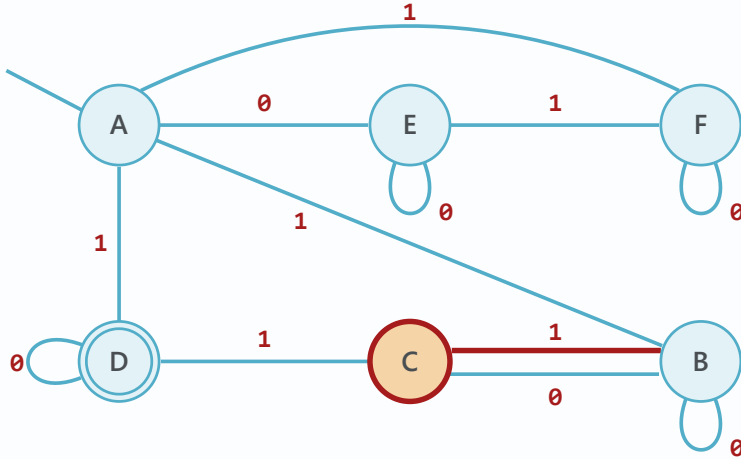
Adım 4 / 6

δ	0	1
$\rightarrow A$	E	B
B	B	C
C	B	D
D^*	D	A
E	E	F
F	F	A

$\rightarrow$ başlangıç, * kabul.

Her hücrede tek durum: **DFA**.

Örnek: İkili Alfabe Üzerinde DFA



4. sembol **1**: $\delta(B, 1) = \mathbf{C}$.

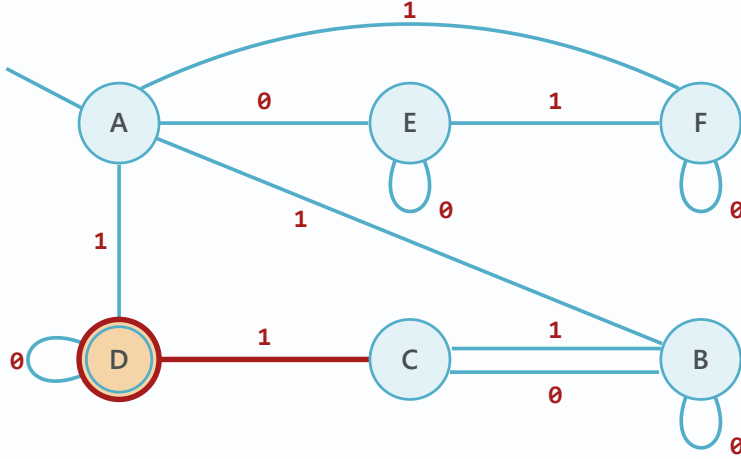
Adım 5 / 6

δ	0	1
$\rightarrow A$	E	B
B	B	C
C	B	D
D^*	D	A
E	E	F
F	F	A

$\rightarrow$ başlangıç, * kabul.

Her hücrede tek durum: **DFA**.

Örnek: İkili Alfabe Üzerinde DFA



δ	0	1
$\rightarrow A$	E	B
B	B	C
C	B	D
D^*	D	A
E	E	F
F	F	A

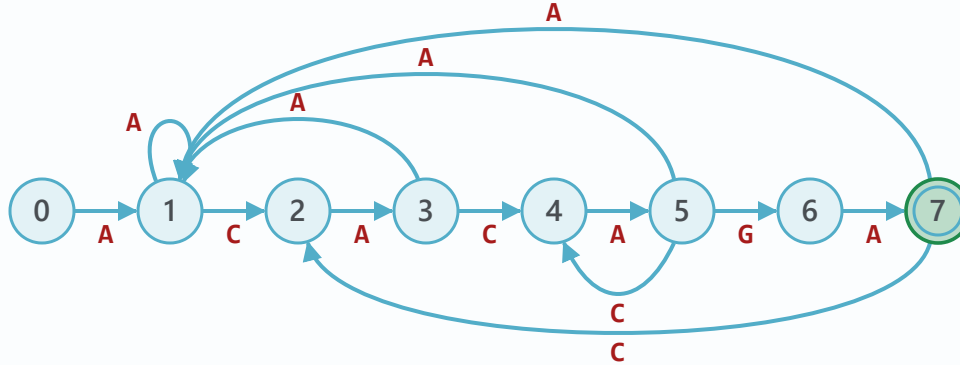
5. sembol **1**: $\delta(C, 1) = \mathbf{D}$. Girdi bitti; D kabul durumu $\rightarrow$ **kabul**.

Adım 6 / 6

$\rightarrow$ başlangıç, * kabul.

Her hücrede tek durum: **DFA**.

Örnek: "ACACAGA" Örüntüsünü Arayan Otomat



metin	G	A	C	A	C	A	C	A	G	A
durum	0	1	2	3	4	5	4	5	6	7

- Durum q = metnin sonunda örüntüyle uyuşan **en uzun önekin** uzunluğu. Çizilmeyen tüm geçişler 0'a gider. Durum 7'ye ulaşıncaya eşleşme bulunur (burada konum 3'te).
- 7. karakter 'C' gelince $5 \rightarrow 4$ dönülür: "ACACAC"nin sonu "ACAC" öneki gibi devam edebilir. Bu, KMP'deki önek fonksiyonunun otomat biçimidir: ön işlem $O(m \cdot |\Sigma|)$, arama **$O(n)$** .

FSM ile Sayı Ayırıştırma: Kodu İzle

```
1 void sayilariBul(String s) {  
2     ▶ Durum d = Durum.BASLA;  
3     ▶ StringBuilder tampon = new StringBuilder();  
4     for (char c : s.toCharArray()) {  
5         boolean rakam = Character.isDigit(c);  
6         if (rakam) {  
7             if (d == Durum.BASLA) d = Durum.RAKAM;  
8             tampon.append(c);  
9         } else if (c == '.' && d != Durum.ONDALIK) {  
10            d = Durum.ONDALIK;  
11            tampon.append(c);  
12        } else if (d != Durum.BASLA) { // sayı bitti  
13            System.out.println(tampon);  
14            tampon.setLength(0);  
15            d = Durum.BASLA;  
16        }  
17    }  
18    if (d != Durum.BASLA)  
19        System.out.println(tampon);  
20 }
```

\$ 2 0 . 9 9 v e 5 .

BASLA

RAKAM

ONDALIK

Girdi "\$20.99 ve 5.". Durum BASLA, tampon boş.

Adım 1 / 14

DEĞİŞKENLER

c = -

rakam = -

d = BASLA

tampon = ""

ÇIKTI

FSM ile Sayı Ayırıştırma: Kodu İzle

```
1 void sayilariBul(String s) {  
2     Durum d = Durum.BASLA;  
3     StringBuilder tampon = new StringBuilder();  
4     for (char c : s.toCharArray()) {  
5         ▶ boolean rakam = Character.isDigit(c);  
6         if (rakam) {  
7             if (d == Durum.BASLA) d = Durum.RAKAM;  
8             tampon.append(c);  
9         } else if (c == '.' && d != Durum.ONDALIK) {  
10            d = Durum.ONDALIK;  
11            tampon.append(c);  
12        } else if (d != Durum.BASLA) { // sayı bitti  
13            System.out.println(tampon);  
14            tampon.setLength(0);  
15            d = Durum.BASLA;  
16        }  
17    }  
18    if (d != Durum.BASLA)  
19        System.out.println(tampon);  
20 }
```

\$ 2 0 . 9 9 v e 5 .

BASLA

RAKAM

ONDALIK

'\$' ne rakam ne nokta; durum BASLA, hiçbir şey yapılmaz.

Adım 2 / 14

DEĞİŞKENLER

c = '\$'

rakam = false

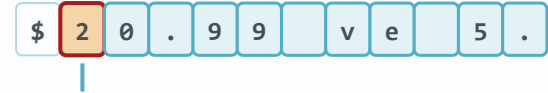
d = BASLA

tampon = ""

ÇIKTI

FSM ile Sayı Ayırıştırma: Kodu İzle

```
1 void sayilariBul(String s) {
2   Durum d = Durum.BASLA;
3   StringBuilder tampon = new StringBuilder();
4   for (char c : s.toCharArray()) {
5     boolean rakam = Character.isDigit(c);
6     if (rakam) {
7       if (d == Durum.BASLA) d = Durum.RAKAM;
8       tampon.append(c);
9     } else if (c == '.' && d != Durum.ONDALIK) {
10      d = Durum.ONDALIK;
11      tampon.append(c);
12    } else if (d != Durum.BASLA) { // sayı bitti
13      System.out.println(tampon);
14      tampon.setLength(0);
15      d = Durum.BASLA;
16    }
17  }
18  if (d != Durum.BASLA)
19    System.out.println(tampon);
20 }
```



BASLA

RAKAM

ONDALIK

'2' rakam ve durum BASLA → **RAKAM**; tampona eklendi.

Adım 3 / 14

DEĞİŞKENLER

c = '2'

rakam = true

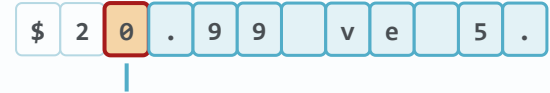
d = RAKAM

tampon = "2"

ÇIKTI

FSM ile Sayı Ayırıştırma: Kodu İzle

```
1 void sayilariBul(String s) {  
2     Durum d = Durum.BASLA;  
3     StringBuilder tampon = new StringBuilder();  
4     for (char c : s.toCharArray()) {  
5         boolean rakam = Character.isDigit(c);  
6         if (rakam) {  
7             if (d == Durum.BASLA) d = Durum.RAKAM;  
8             tampon.append(c);  
9         } else if (c == '.' && d != Durum.ONDALIK) {  
10             d = Durum.ONDALIK;  
11             tampon.append(c);  
12         } else if (d != Durum.BASLA) { // sayı bitti  
13             System.out.println(tampon);  
14             tampon.setLength(0);  
15             d = Durum.BASLA;  
16         }  
17     }  
18     if (d != Durum.BASLA)  
19         System.out.println(tampon);  
20 }
```



BASLA

RAKAM

ONDALIK

'0' rakam: durum RAKAM kalır, tampona eklenir.

Adım 4 / 14

DEĞİŞKENLER

c = '0'

rakam = true

d = RAKAM

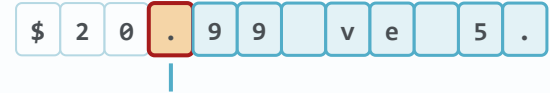
tampon = "20"

ÇIKTI



FSM ile Sayı Ayırıştırma: Kodu İzle

```
1 void sayilariBul(String s) {  
2     Durum d = Durum.BASLA;  
3     StringBuilder tampon = new StringBuilder();  
4     for (char c : s.toCharArray()) {  
5         boolean rakam = Character.isDigit(c);  
6         if (rakam) {  
7             if (d == Durum.BASLA) d = Durum.RAKAM;  
8             tampon.append(c);  
9         } else if (c == '.' && d != Durum.ONDALIK) {  
10            d = Durum.ONDALIK;  
11            tampon.append(c);  
12        } else if (d != Durum.BASLA) { // sayı bitti  
13            System.out.println(tampon);  
14            tampon.setLength(0);  
15            d = Durum.BASLA;  
16        }  
17    }  
18    if (d != Durum.BASLA)  
19        System.out.println(tampon);  
20 }
```



BASLA

RAKAM

ONDALIK

'.' geldi → **ONDALIK** durumuna geçildi, nokta tampona eklendi.

Adım 5 / 14

DEĞİŞKENLER

c = '.'

rakam = false

d = ONDALIK

tampon = "20."

ÇIKTI



FSM ile Sayı Ayırıştırma: Kodu İzle

```
1 void sayilariBul(String s) {  
2     Durum d = Durum.BASLA;  
3     StringBuilder tampon = new StringBuilder();  
4     for (char c : s.toCharArray()) {  
5         boolean rakam = Character.isDigit(c);  
6         ▶ if (rakam) {  
7             if (d == Durum.BASLA) d = Durum.RAKAM;  
8             ▶ tampon.append(c);  
9         } else if (c == '.' && d != Durum.ONDALIK) {  
10             d = Durum.ONDALIK;  
11             tampon.append(c);  
12         } else if (d != Durum.BASLA) { // sayı bitti  
13             System.out.println(tampon);  
14             tampon.setLength(0);  
15             d = Durum.BASLA;  
16         }  
17     }  
18     if (d != Durum.BASLA)  
19         System.out.println(tampon);  
20 }
```



BASLA

RAKAM

ONDALIK

'9' rakam: durum ONDALIK kalır, tampona eklenir.

Adım 6 / 14

DEĞİŞKENLER

c = '9'

rakam = true

d = ONDALIK

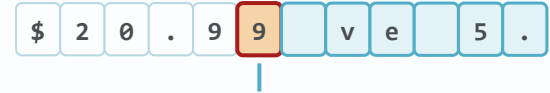
tampon = "20.9"

ÇIKTI



FSM ile Sayı Ayırıştırma: Kodu İzle

```
1 void sayilariBul(String s) {  
2     Durum d = Durum.BASLA;  
3     StringBuilder tampon = new StringBuilder();  
4     for (char c : s.toCharArray()) {  
5         boolean rakam = Character.isDigit(c);  
6         ▶ if (rakam) {  
7             if (d == Durum.BASLA) d = Durum.RAKAM;  
8             ▶ tampon.append(c);  
9         } else if (c == '.' && d != Durum.ONDALIK) {  
10             d = Durum.ONDALIK;  
11             tampon.append(c);  
12         } else if (d != Durum.BASLA) { // sayı bitti  
13             System.out.println(tampon);  
14             tampon.setLength(0);  
15             d = Durum.BASLA;  
16         }  
17     }  
18     if (d != Durum.BASLA)  
19         System.out.println(tampon);  
20 }
```



BASLA

RAKAM

ONDALIK

'9' rakam: durum ONDALIK kalır, tampona eklenir.

Adım 7 / 14

DEĞİŞKENLER

c = '9'

rakam = true

d = ONDALIK

tampon = "20.99"

ÇIKTI

FSM ile Sayı Ayırıştırma: Kodu İzle

```
1 void sayilariBul(String s) {  
2     Durum d = Durum.BASLA;  
3     StringBuilder tampon = new StringBuilder();  
4     for (char c : s.toCharArray()) {  
5         boolean rakam = Character.isDigit(c);  
6         if (rakam) {  
7             if (d == Durum.BASLA) d = Durum.RAKAM;  
8             tampon.append(c);  
9         } else if (c == '.' && d != Durum.ONDALIK) {  
10            d = Durum.ONDALIK;  
11            tampon.append(c);  
12        } else if (d != Durum.BASLA) { // sayı bitti  
13            System.out.println(tampon);  
14            tampon.setLength(0);  
15            d = Durum.BASLA;  
16        }  
17    }  
18    if (d != Durum.BASLA)  
19        System.out.println(tampon);  
20 }
```

\$ 2 0 . 9 9 v e 5 .

BASLA

RAKAM

ONDALIK

' ' sayı karakteri değil: sayı bitti, **20.99** yazdırıldı;
BASLA'ya dönülür.

Adım 8 / 14

DEĞİŞKENLER

c = ' '

rakam = false

d = BASLA

tampon = ""

ÇIKTI

20.99

FSM ile Sayı Ayırıştırma: Kodu İzle

```
1 void sayilariBul(String s) {  
2     Durum d = Durum.BASLA;  
3     StringBuilder tampon = new StringBuilder();  
4     for (char c : s.toCharArray()) {  
5         ▶ boolean rakam = Character.isDigit(c);  
6         if (rakam) {  
7             if (d == Durum.BASLA) d = Durum.RAKAM;  
8             tampon.append(c);  
9         } else if (c == '.' && d != Durum.ONDALIK) {  
10            d = Durum.ONDALIK;  
11            tampon.append(c);  
12        } else if (d != Durum.BASLA) { // sayı bitti  
13            System.out.println(tampon);  
14            tampon.setLength(0);  
15            d = Durum.BASLA;  
16        }  
17    }  
18    if (d != Durum.BASLA)  
19        System.out.println(tampon);  
20 }
```



BASLA

RAKAM

ONDALIK

'v' ne rakam ne nokta; durum BASLA, hiçbir şey yapılmaz.

Adım 9 / 14

DEĞİŞKENLER

c = 'v'

rakam = false

d = BASLA

tampon = ""

ÇIKTI

20.99

FSM ile Sayı Ayırıştırma: Kodu İzle

```
1 void sayilariBul(String s) {  
2     Durum d = Durum.BASLA;  
3     StringBuilder tampon = new StringBuilder();  
4     for (char c : s.toCharArray()) {  
5         ▶ boolean rakam = Character.isDigit(c);  
6         if (rakam) {  
7             if (d == Durum.BASLA) d = Durum.RAKAM;  
8             tampon.append(c);  
9         } else if (c == '.' && d != Durum.ONDALIK) {  
10            d = Durum.ONDALIK;  
11            tampon.append(c);  
12        } else if (d != Durum.BASLA) { // sayı bitti  
13            System.out.println(tampon);  
14            tampon.setLength(0);  
15            d = Durum.BASLA;  
16        }  
17    }  
18    if (d != Durum.BASLA)  
19        System.out.println(tampon);  
20 }
```



BASLA

RAKAM

ONDALIK

'e' ne rakam ne nokta; durum BASLA, hiçbir şey yapılmaz.

Adım 10 / 14

DEĞİŞKENLER

c = 'e'

rakam = false

d = BASLA

tampon = ""

ÇIKTI

20.99

FSM ile Sayı Ayırıştırma: Kodu İzle

```
1 void sayilariBul(String s) {  
2     Durum d = Durum.BASLA;  
3     StringBuilder tampon = new StringBuilder();  
4     for (char c : s.toCharArray()) {  
5         ▶ boolean rakam = Character.isDigit(c);  
6         if (rakam) {  
7             if (d == Durum.BASLA) d = Durum.RAKAM;  
8             tampon.append(c);  
9         } else if (c == '.' && d != Durum.ONDALIK) {  
10            d = Durum.ONDALIK;  
11            tampon.append(c);  
12        } else if (d != Durum.BASLA) { // sayı bitti  
13            System.out.println(tampon);  
14            tampon.setLength(0);  
15            d = Durum.BASLA;  
16        }  
17    }  
18    if (d != Durum.BASLA)  
19        System.out.println(tampon);  
20 }
```



BASLA

RAKAM

ONDALIK

' ' ne rakam ne nokta; durum BASLA, hiçbir şey yapılmaz.

Adım 11 / 14

DEĞİŞKENLER

c = '5'

rakam = false

d = BASLA

tampon = ""

ÇIKTI

20.99

FSM ile Sayı Ayırıştırma: Kodu İzle

```
1 void sayilariBul(String s) {  
2     Durum d = Durum.BASLA;  
3     StringBuilder tampon = new StringBuilder();  
4     for (char c : s.toCharArray()) {  
5         boolean rakam = Character.isDigit(c);  
6         if (rakam) {  
7             if (d == Durum.BASLA) d = Durum.RAKAM;  
8             tampon.append(c);  
9         } else if (c == '.' && d != Durum.ONDALIK) {  
10             d = Durum.ONDALIK;  
11             tampon.append(c);  
12         } else if (d != Durum.BASLA) { // sayı bitti  
13             System.out.println(tampon);  
14             tampon.setLength(0);  
15             d = Durum.BASLA;  
16         }  
17     }  
18     if (d != Durum.BASLA)  
19         System.out.println(tampon);  
20 }
```



BASLA

RAKAM

ONDALIK

'5' rakam ve durum BASLA → **RAKAM**; tampona eklendi.

Adım 12 / 14

DEĞİŞKENLER

c = '5'

rakam = true

d = RAKAM

tampon = "5"

ÇIKTI

20.99

FSM ile Sayı Ayırıştırma: Kodu İzle

```
1 void sayilariBul(String s) {  
2     Durum d = Durum.BASLA;  
3     StringBuilder tampon = new StringBuilder();  
4     for (char c : s.toCharArray()) {  
5         boolean rakam = Character.isDigit(c);  
6         if (rakam) {  
7             if (d == Durum.BASLA) d = Durum.RAKAM;  
8             tampon.append(c);  
9             } else if (c == '.' && d != Durum.ONDALIK) {  
10                d = Durum.ONDALIK;  
11                tampon.append(c);  
12            } else if (d != Durum.BASLA) { // sayı bitti  
13                System.out.println(tampon);  
14                tampon.setLength(0);  
15                d = Durum.BASLA;  
16            }  
17        }  
18        if (d != Durum.BASLA)  
19            System.out.println(tampon);  
20    }
```

\$ 2 0 . 9 9 v e 5 .

BASLA

RAKAM

ONDALIK

'.' geldi → **ONDALIK** durumuna geçildi, nokta tampona eklendi.

Adım 13 / 14

DEĞİŞKENLER

c = '.'

rakam = false

d = ONDALIK

tampon = "5."

ÇIKTI

20.99

FSM ile Sayı Ayırıştırma: Kodu İzle

```
1 void sayilariBul(String s) {  
2     Durum d = Durum.BASLA;  
3     StringBuilder tampon = new StringBuilder();  
4     for (char c : s.toCharArray()) {  
5         boolean rakam = Character.isDigit(c);  
6         if (rakam) {  
7             if (d == Durum.BASLA) d = Durum.RAKAM;  
8             tampon.append(c);  
9         } else if (c == '.' && d != Durum.ONDALIK) {  
10            d = Durum.ONDALIK;  
11            tampon.append(c);  
12        } else if (d != Durum.BASLA) { // sayı bitti  
13            System.out.println(tampon);  
14            tampon.setLength(0);  
15            d = Durum.BASLA;  
16        }  
17    }  
18    if (d != Durum.BASLA)  
19        System.out.println(tampon);  
20 }
```

\$ 2 0 . 9 9 v e 5 .

BASLA

RAKAM

ONDALIK

Girdi bitti; durum ONDALIK olduğundan tampon yazdırılır: **5..** Sondaki nokta da sayıya katıldı (kusur; bkz. Alistirmalar).

Adım 14 / 14

DEĞİŞKENLER

c = -

rakam = -

d = ONDALIK

tampon = "5."

ÇIKTI

20.99

5.

Levenshtein Uzaklığı (Edit Distance)

- **Levenshtein uzaklığı** (V. Levenshtein, 1965): iki dizgi arasındaki farkı **nicel** olarak ölçen bir metriktir.
- Bir dizgiyi diğerine dönüştürmek için gereken **en az tek karakterlik düzenleme** sayısıdır.
- Otomatik düzeltme ve tahmin sistemlerinde, yazım denetiminde, DNA dizilerini karşılaştırmada, bulanık (fuzzy) aramada kullanılır.
- Metrik özellikleri: $d(a, a) = 0$, $d(a, b) = d(b, a)$, üçgen eşitsizliği.

Ekleme (insertion)

Bir karakter eklemek: "sittin" → "sitting"

Silme (deletion)

Bir karakteri çıkarmak: "shine" → "shin"

Değiştirme (substitution)

Bir karakteri başkasıyla değiştirmek: "**k**itten" → "sitten"

kitten → sitten (k→s) → sittin (e→i) → sitting (+g): **d = 3**. Daha kısa bir yol yoktur; bunu dinamik programlama kanıtlar.

Düzenleme Örnekleri

HONDA	H	O		N	D	A	
HYUNDAI	H	Y	U	N	D	A	I

HONDA → HYUNDAI: O→Y değiştir, U ekle, I ekle: d = 3

rain

sainshinshinee

(a) 3

shine

rhineraineraine

(b) 3

shine

tshinetrhinetrainetraine

(c) 4

değiştirme

ekleme

silme

- (a) rain → **s**ain → **sh**in → shine · (b) shine → **r**hine → **raine** → rain (e silinir) · (c) shine → **t**shine → **tr**hine → **traine** → train (e silinir). Üstü çizili turuncu harf silinir.
- Uzaklık **simetrik**: (a) ile (b) birbirinin tersidir; ekleme ↔ silme yer değiştirir.

Levenshtein Uzaklığını Hesaplama

$D[i][j]$ = a'nın ilk i karakterini b'nin ilk j karakterine dönüştürmenin en az maliyeti.

$D[i][0] = i$ (i silme) · $D[0][j] = j$ (j ekleme)

$a_i = b_j$ ise $D[i][j] = D[i-1][j-1]$

aksi hâlde $D[i][j] = 1 + \min(D[i-1][j] \text{ silme, } D[i][j-1] \text{ ekleme, } D[i-1][j-1] \text{ değiştirme })$

- **Dinamik programlama:** her alt problem bir kez çözülür, $(m+1) \times (n+1)$ tabloda saklanır; cevap sağ alt köşededir.
- Bir hücre yalnızca **üst**, **sol** ve **sol üst** komşulara bağlıdır → tablo satır satır doldurulur.
- Zaman **$O(m \cdot n)$** ; bellek $O(m \cdot n)$, yalnızca uzaklık gerekiyorsa iki satırla **$O(\min(m, n))$** .

DP Tablosu: CARS → PAIRS

a = **CARS** (satırlar), b = **PAIRS** (sütunlar)

	ε	P	A	I	R	S
ε	0	1	2	3	4	5
C	1					
A	2					
R	3					
S	4					

Başlangıç: ilk satır 0..5 ($\epsilon \rightarrow P, PA, \dots$: ekleme), ilk sütun 0..4 (C, CA, $\dots \rightarrow \epsilon$: silme).

DP Tablosu: CARS → PAIRS

a = **CARS** (satırlar), b = **PAIRS** (sütunlar)

	ε	P	A	I	R	S
ε	0	1	2	3	4	5
C	1	1	2	3	4	5
A	2					
R	3					
S	4					

1. satır (C) dolduruldu. $C \neq P$: $D[1][1] = 1 + \min(0, 1, 1) = 1$ (değiştirme).

DP Tablosu: CARS → PAIRS

a = **CARS** (satırlar), b = **PAIRS** (sütunlar)

	ε	P	A	I	R	S
ε	0	1	2	3	4	5
C	1	1	2	3	4	5
A	2	2	1	2	3	4
R	3					
S	4					

2. satır (A) dolduruldu. A = A: $D[2][2] = D[1][1] = 1$, maliyet eklenmez.

DP Tablosu: CARS → PAIRS

a = **CARS** (satırlar), b = **PAIRS** (sütunlar)

	ε	P	A	I	R	S
ε	0	1	2	3	4	5
C	1	1	2	3	4	5
A	2	2	1	2	3	4
R	3	3	2	2	2	3
S	4					

3. satır (R) dolduruldu. R = R: $D[3][4] = D[2][3] = 2$. A ≠ I: $D[2][3] = 1 + \min(1, 2, 3) = 2$.

DP Tablosu: CARS → PAIRS

a = **CARS** (satırlar), b = **PAIRS** (sütunlar)

	ε	P	A	I	R	S
ε	0	1	2	3	4	5
C	1	1	2	3	4	5
A	2	2	1	2	3	4
R	3	3	2	2	2	3
S	4	4	3	3	3	2

4. satır (S) dolduruldu. S = S: $D[4][5] = D[3][4] = 2$.

DP Tablosu: CARS → PAIRS

a = **CARS** (satırlar), b = **PAIRS** (sütunlar)

	ε	P	A	I	R	S
ε	0	1	2	3	4	5
C	1	1	2	3	4	5
A	2	2	1	2	3	4
R	3	3	2	2	2	3
S	4	4	3	3	3	2

Sonuç **D[4][5] = 2**. Geri okuma: C→P değiştir, A eşit, **I ekle**, R ve S eşit: CARS → PARS → PAIRS.

Levenshtein: Java Kodu

```
int levenshtein(String a, String b) {  
    int m = a.length(), n = b.length();  
    int[][] D = new int[m + 1][n + 1];  
    for (int i = 0; i <= m; i++) D[i][0] = i; // silme  
    for (int j = 0; j <= n; j++) D[0][j] = j; // ekleme  
    for (int i = 1; i <= m; i++)  
        for (int j = 1; j <= n; j++)  
            if (a.charAt(i - 1) == b.charAt(j - 1))  
                D[i][j] = D[i - 1][j - 1]; // eşit  
            else  
                D[i][j] = 1 + Math.min(D[i - 1][j - 1], // değiştir  
                                       Math.min(D[i][j - 1], // ekle  
                                                  D[i - 1][j])); // sil  
    return D[m][n];  
}
```

- Tablo indisi i , dizgi indisi $i - 1$ 'dir: D'nin 0. satırı **boş örnek** içindir.
- Örnekler: kitten → sitting = 3, ercan → sercan = 1, apple → banana = 5.

En Uzun Ortak Alt Dizi (LCS)

- **En uzun ortak alt dizi** (longest common subsequence, LCS): iki dizginin her ikisinde de **sıralı** olarak bulunan en uzun dizi.
- Karakterlerin **ardışık olması gerekmez**, ancak göreceli sıraları korunmalıdır.
- İki dizgi arasındaki benzerlik seviyesini ölçer: `diff`, sürüm kontrolü, DNA hizalama, intihal tespiti.

Dizi 1: A G G T A B
Dizi 2: G X T X A Y B
LCS = GTAB, uzunluk 4

dizi 1: a c b a e d
dizi 2: a b c a d f
LCS = acad, uzunluk 4 (abad da olur)

Alt dizi ≠ alt dizgi: "GTAB", GXTXAYB'nin alt dizisidir ama alt dizgisi (substring) değildir. LCS tek olmak zorunda değildir; **uzunluğu** tektir.

Dinamik Programlama Yaklaşımı

```
String lcs(String a, String b) {  
    int m = a.length(), n = b.length();  
    int[][] L = new int[m + 1][n + 1];  
    for (int i = 1; i <= m; i++)  
        for (int j = 1; j <= n; j++)  
            if (a.charAt(i - 1) == b.charAt(j - 1))  
                L[i][j] = L[i - 1][j - 1] + 1;  
            else  
                L[i][j] = Math.max(L[i - 1][j], L[i][j - 1]);  
    StringBuilder s = new StringBuilder(); // geri oku  
    for (int i = m, j = n; i > 0 && j > 0; )  
        if (a.charAt(i - 1) == b.charAt(j - 1)) {  
            s.insert(0, a.charAt(i - 1)); i--; j--;  
        } else if (L[i - 1][j] > L[i][j - 1]) i--;  
        else j--;  
    return s.toString();  
}
```

Zaman ve bellek $O(m \cdot n)$; geri okuma $O(m + n)$.

1. Uzunluklar m ve n ; $(m+1) \times (n+1)$ tablo.
2. İlk satır ve sütun **0**.
3. $A[i] = B[j]$ ise hücre = **sol üst + 1**.
4. $A[i] \neq B[j]$ ise hücre = **max(üst, sol)**.
5. Son hücre LCS **uzunluğudur**; dizinin kendisi geri okumayla bulunur.

LCS Tablosu: AGGTAB · GXTXAYB

A = **AGGTAB** (satırlar), B = **GXTXAYB** (sütunlar)

	ε	G	X	T	X	A	Y	B
ε	0	0	0	0	0	0	0	0
A	0							
G	0							
G	0							
T	0							
A	0							
B	0							

İlk satır ve sütun 0: boş dizginin herhangi bir dizgiyle LCS'si boştur.

LCS Tablosu: AGGTAB · GXTXAYB

A = **AGGTAB** (satırlar), B = **GXTXAYB** (sütunlar)

	ε	G	X	T	X	A	Y	B
ε	0	0	0	0	0	0	0	0
A	0	0	0	0	0	1	1	1
G	0							
G	0							
T	0							
A	0							
B	0							

1. satır (A). A yalnızca B[5] = A ile eşleşir: $L[1][5] = 0 + 1 = 1$; sağa doğru max ile 1 taşınır.

LCS Tablosu: AGGTAB · GXTXAYB

A = **AGGTAB** (satırlar), B = **GXTXAYB** (sütunlar)

	ε	G	X	T	X	A	Y	B
ε	0	0	0	0	0	0	0	0
A	0	0	0	0	0	1	1	1
G	0	1	1	1	1	1	1	1
G	0							
T	0							
A	0							
B	0							

2. satır (G). G = G (j = 1): $L[2][1] = 1$. Satırın geri kalanı $\max(\text{üst}, \text{sol}) = 1$.

LCS Tablosu: AGGTAB · GXTXAYB

A = **AGGTAB** (satırlar), B = **GXTXAYB** (sütunlar)

	ε	G	X	T	X	A	Y	B
ε	0	0	0	0	0	0	0	0
A	0	0	0	0	0	1	1	1
G	0	1	1	1	1	1	1	1
G	0	1	1	1	1	1	1	1
T	0							
A	0							
B	0							

3. satır (G). İkinci G de yalnızca 1 verir: aynı G iki kez kullanılamaz (sol üst = 0).

LCS Tablosu: AGGTAB · GXTXAYB

A = **AGGTAB** (satırlar), B = **GXTXAYB** (sütunlar)

	ε	G	X	T	X	A	Y	B
ε	0	0	0	0	0	0	0	0
A	0	0	0	0	0	1	1	1
G	0	1	1	1	1	1	1	1
G	0	1	1	1	1	1	1	1
T	0	1	1	2	2	2	2	2
A	0							
B	0							

4. satır (T). T = T (j = 3): $L[4][3] = L[3][2] + 1 = 2 \rightarrow \text{"GT"}$.

LCS Tablosu: AGGTAB · GXTXAYB

A = **AGGTAB** (satırlar), B = **GXTXAYB** (sütunlar)

	ε	G	X	T	X	A	Y	B
ε	0	0	0	0	0	0	0	0
A	0	0	0	0	0	1	1	1
G	0	1	1	1	1	1	1	1
G	0	1	1	1	1	1	1	1
T	0	1	1	2	2	2	2	2
A	0	1	1	2	2	3	3	3
B	0							

5. satır (A). A = A (j = 5): $L[5][5] = L[4][4] + 1 = 3 \rightarrow \text{"GTA"}$.

LCS Tablosu: AGGTAB · GXTXAYB

A = **AGGTAB** (satırlar), B = **GXTXAYB** (sütunlar)

	ε	G	X	T	X	A	Y	B
ε	0	0	0	0	0	0	0	0
A	0	0	0	0	0	1	1	1
G	0	1	1	1	1	1	1	1
G	0	1	1	1	1	1	1	1
T	0	1	1	2	2	2	2	2
A	0	1	1	2	2	3	3	3
B	0	1	1	2	2	3	3	4

6. satır (B). B = B (j = 7): $L[6][7] = L[5][6] + 1 = 4 \rightarrow \text{"GTAB"}$.

LCS Tablosu: AGGTAB · GXTXAYB

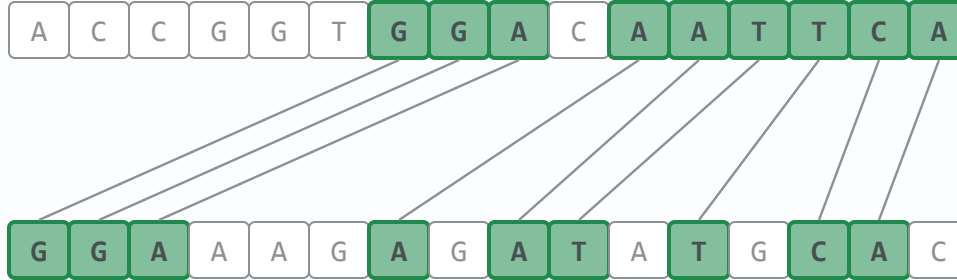
A = **AGGTAB** (satırlar), B = **GXTXAYB** (sütunlar)

	ε	G	X	T	X	A	Y	B
ε	0	0	0	0	0	0	0	0
A	0	0	0	0	0	1	1	1
G	0	1	1	1	1	1	1	1
G	0	1	1	1	1	1	1	1
T	0	1	1	2	2	2	2	2
A	0	1	1	2	2	3	3	3
B	0	1	1	2	2	3	3	4

Geri okuma sağ alttan: eşleşmede çapraz (turuncu, karakter LCS'ye eklenir), değilse büyük olan komşuya. Sonuç **GTAB**, uzunluk 4.

EN UZUN ORTAK ALT DİZİ

Örnek: DNA Dizilerinde LCS



- Eşleşen karakterleri birleştiren çizgiler **kesişmez**: görelî sıra korunur. Bu kural sağlanan her eşleştirme bir ortak alt dizidir.
- DP sonucu: LCS = **GGAAATTCA**, uzunluk **9**.

Dikkat: Göz kararı eşleştirme yanıltıcıdır. Örneğin AAAATTCA (8 karakter) da geçerli bir ortak alt dizidir, ama **en uzun** değildir; doğru cevabı DP tablosu verir.

Benzerlik Ölçülerinin Karşılaştırılması

Ölçü	Ne ölçer?	Zaman	AGGTAB · GXTXAYB
Hamming	eşit uzunlukta, farklı konum sayısı	$O(n)$	tanımsız ($6 \neq 7$)
Levenshtein	ekle / sil / değiştir sayısı	$O(m \cdot n)$	4
LCS	en uzun ortak alt dizi	$O(m \cdot n)$	4 (GTAB)
En uzun ortak alt dizgi	ardışık ortak parça	$O(m \cdot n)$	1 (ör. "G")

- Yalnızca ekleme ve silmeye izin verilirse uzaklık = $m + n - 2 \cdot \text{LCS}$: $6 + 7 - 8 = 5$; değiştirmeye de izin veren Levenshtein bunu 4'e indirir.
- En uzun ortak alt dizgi: eşleşmede $L[i][j] = L[i-1][j-1] + 1$, eşleşmezse **0**; cevap tablonun en büyük değeridir (ercan · sercan → "ercan").
- Benzerlik oranı örneği: $1 - d / \max(m, n)$. kitten · sitting: $1 - 3/7 \approx 0,57$.

DİZGİ DÖNÜŞÜMLERİ

Sonek Dizisi (Suffix Array)

- **Sonek dizisi** (suffix array): bir dizginin tüm soneklerinin **alfabetik sıraya** dizilmiş hâlinin başlangıç indisleri.
- Her sonek, dizginin belirli bir konumundan başlayıp sonuna kadar giden alt dizgidir. Sonek dizisi tüm alt dizgilerin **sıkı bir temsilidir**: her alt dizgi bir sonekin önekidir.

i	sonek
0	banana
1	anana
2	nana
3	ana
4	na
5	a

sıra	SA	sıralı sonek
0	5	a
1	3	ana
2	1	anana
3	0	banana
4	4	na
5	2	nana

"banana" → SA = [5, 3, 1, 0, 4, 2]. Kullanım: dizgi arama, sıralama, genetik dizilim analizi, BWT ve veri sıkıştırma.

Sonek Dizisi Oluşturma Yöntemleri

Yöntem	Fikir	Zaman
Kaba kuvvet	Tüm sonekleri oluştur, <code>Arrays.sort</code> ile sırala; her karşılaştırma $O(n)$	$O(n^2 \log n)$
Manber–Myers (1990)	Önek ikiye katlama: k karakterlik sıralardan $2k$ karakterlik sırayı harf karşılaştırmadan , sıra çiftleriyle basamak sıralamasıyla sıralayarak bulur	$O(n \log n)$
Larsson–Sadakane (1999)	Aynı ikiye katlama; yalnızca henüz ayrışmamış gruplar, art arda soneklerin sıraları karşılaştırılarak yeniden sıralanır	$O(n \log n)$
DC3 (2003), SA-IS (2009)	Soneklerin bir kısmını özyinelemeli sıralayıp kalanı birleştirir	$O(n)$

Önek ikiye katlama yöntemleri **$O(n \log n)$** zamanlıdır, doğrusal değildir. Doğrusal zamanlı kurulum DC3 ve SA-IS gibi sonraki algoritmalarla mümkündür.

Önek İkiye Katlama (Prefix Doubling)

Manber–Myers fikri, "banana" üzerinde: k karakterlik sıralardan $2k$ karakterlik sıra elde edilir. $\log_2 n$ tur, her tur basamak sıralamasıyla $O(n)$.

i	ilk 1 karakter	anahtar	yeni sıra
1	a	'a'	0
3	a	'a'	0
5	a	'a'	0
0	b	'b'	1
2	n	'n'	2
4	n	'n'	2

$k = 1$: sonekler ilk karakterlerine göre sıralanır: $a \rightarrow 0$, $b \rightarrow 1$, $n \rightarrow 2$. Eşit sıralar henüz ayrıışmamıştır.

Önek İkiye Katlama (Prefix Doubling)

Manber–Myers fikri, "banana" üzerinde: k karakterlik sıralardan $2k$ karakterlik sıra elde edilir. $\log_2 n$ tur, her tur basamak sıralamasıyla $O(n)$.

i	ilk 2 karakter	anahtar	yeni sıra
5	a	(0, -1)	0
1	an	(0, 2)	1
3	an	(0, 2)	1
0	ba	(1, 0)	2
2	na	(2, 0)	3
4	na	(2, 0)	3

$k = 2$: anahtar = (sıra[i], sıra[i + 1]); $i + 1 \geq n$ ise -1. Harf karşılaştırılmaz, yalnızca sayı çiftleri sıralanır.

Önek İkiye Katlama (Prefix Doubling)

Manber–Myers fikri, "banana" üzerinde: k karakterlik sıralardan $2k$ karakterlik sıra elde edilir. $\log_2 n$ tur, her tur basamak sıralamasıyla $O(n)$.

i	ilk 4 karakter	anahtar	yeni sıra
5	a	(0, -1)	0
3	ana	(1, 0)	1
1	anan	(1, 1)	2
0	bana	(2, 3)	3
4	na	(3, -1)	4
2	nana	(3, 3)	5

$k = 4$: anahtar = (sıra[i], sıra[i + 2]); $i + 2 \geq n$ ise -1 . Harf karşılaştırılmaz, yalnızca sayı çiftleri sıralanır. Tüm sıralar farklı $\rightarrow$ SA = **[5, 3, 1, 0, 4, 2]**.

Sonek Dizisiyle Arama

- Sonekler sıralı olduğundan, P örüntüsüyle **başlayan** sonekler SA'da **bitişik bir aralık** oluşturur.
- Aralığın iki ucu **ikili arama** ile bulunur: $O(\log n)$ adım $\times O(m)$ karşılaştırma = **$O(m \log n)$** .
- Sonek trie'sine göre çok az bellek: yalnızca n tam sayı.
- LCP dizisi (komşu soneklerin en uzun ortak öneki) eklenirse en uzun tekrar eden alt dizgi $O(n)$ 'de bulunur.

sıra	SA	sonek
0	5	a
1	3	ana
2	1	anana
3	0	banana
4	4	na
5	2	nana

Örnek: "banana" içinde $P = "an"$. Orta = sıra 2 ("anana") "an" ile başlar; sol ve sağ sınır aramaları aralığı $[1, 2]$ bulur $\rightarrow$ "an", **3 ve 1** konumlarında geçer.

Burrows–Wheeler Dönüşümü (BWT)

- **Burrows–Wheeler dönüşümü** (BWT, 1994): girdiyi **yeniden düzenler**; karakterleri değiştirmez, yalnızca yerlerini değiştirir.
- **Aynı karakterlerin bir araya** toplanmasını sağlar: benzer bağlamdan önce gelen karakterler yan yana düşer.
- Sıkıştırma algoritmalarının performansını artırır.
- **Tersine çevrilebilir**: orijinal veri dönüşümden geri elde edilir.

Uygulama adımları

1. Girdi dizgisinin **tüm döndürülmüş hâllerini** (cyclic rotations) oluştur.
2. Döndürmeleri **alfabetik** olarak sırala.
3. Sıralı döndürmelerin **son karakterlerinden** yeni bir dizgi oluştur (L sütunu).
4. Orijinal dizginin sıralı listedeki **satır numarasını** sakla (ya da sona tekil bir \$ ekle).

bzip2 zinciri: BWT → Move-to-Front (tekrarlar küçük sayılara dönüşür) → Run-Length Encoding → Huffman kodlama.

BWT Örneği: BANANA

Girdi: **BANANA**

döndürmeler

0	B	A	N	A	N	A
1	A	N	A	N	A	B
2	N	A	N	A	B	A
3	A	N	A	B	A	N
4	N	A	B	A	N	A
5	A	B	A	N	A	N

1. Tüm döndürmeler: BANANA, ANANAB, NANABA, ANABAN, NABANA, ABANAN (ilk karakter sona taşınarak).

BWT Örneği: BANANA

Girdi: **BANANA**

döndürmeler							sıralı					
B	A	N	A	N	A	0	A	B	A	N	A	N
A	N	A	N	A	B	1	A	N	A	B	A	N
N	A	N	A	B	A	2	A	N	A	N	A	B
A	N	A	B	A	N	3	B	A	N	A	N	A
N	A	B	A	N	A	4	N	A	B	A	N	A
A	B	A	N	A	N	5	N	A	N	A	B	A

2. Alfabetik sıralama: ABANAN, ANABAN, ANANAB, BANANA, NABANA, NANABA. Orijinal dizgi **3. satırda**.

BWT Örneği: BANANA

Girdi: **BANANA**

	sıralı					
0	A	B	A	N	A	N
1	A	N	A	B	A	N
2	A	N	A	N	A	B
3	B	A	N	A	N	A
4	N	A	B	A	N	A
5	N	A	N	A	B	A

—

N	N	B	A	A	A
---	---	---	---	---	---

BWT = NNBAAA, satır 3

3. Son karakterler (L sütunu): **NNBAAA**. Çıktı: (NNBAAA, 3). Aynı harfler gruplandı: NN, AAA.

DİZGİ DÖNÜŞÜMLERİ

BWT Örneği: abraca

S = abraca (başlangıç)

a	b	r	a	c	a
b	r	a	c	a	a
r	a	c	a	a	b
a	c	a	a	b	r
c	a	a	b	r	a
a	a	b	r	a	c

sıralı matris

a	a	b	r	a	c
a	b	r	a	c	a
a	c	a	a	b	r
b	r	a	c	a	a
c	a	a	b	r	a
r	a	c	a	a	b

← L

- Sıralı matrisin son sütunu: **L = caraab**.
- Orijinal "abraca" sıralı matriste **1. satırdadır** (0'dan sayarak); ters dönüşüm için bu indis saklanır.
- İlk sütun (F = aaabcr), L'nin harflerinin sıralanmış hâlidir; L'yi bilmek F'yi de verir.

Doğrudan yöntem n döndürmeyi `substring` ile üretip `Arrays.sort` ile sıralar → $O(n^2 \log n)$ zaman, $O(n^2)$ bellek. Pratikte son ek dizisinden **$L[i] = T[SA[i] - 1]$** ile $O(n)$ 'de elde edilir (T sonuna \$ eklenmiş metin).

DİZGİ DÖNÜŞÜMLERİ

BWT'yi Tersine Çevirme (LF Eşlemesi)

i	F	...	L
0	\$	banan	a
1	a	\$bana	n
2	a	na\$ba	n
3	a	nana\$	b
4	b	anana	\$
5	n	a\$ban	a
6	n	ana\$b	a

- $T = \text{banana\$}$ (\$ en küçük, tekil) $\rightarrow \text{BWT} = \text{annb\$aa}$.
- $F = \text{sırala}(L) = \$\text{aaabnn}$.
- **LF kuralı:** L'deki k. 'c', F'deki k. 'c' ile aynı karakterdir. i satırındaki L[i], metinde F[i]'den hemen önce gelir.
- \$ ile başlayan 0. satırdan başla, L[i]'yi oku, LF(i) satırına git:

satırlar $0 \rightarrow 1 \rightarrow 5 \rightarrow 2 \rightarrow 6 \rightarrow 3 \rightarrow 4$

okunan L: a n a n a b $\rightarrow$ ters çevir $\rightarrow$ **banana**

Sıra sayıları (her karakterin L'de kaçınıcı olduğu) önceden hesaplanırsa ters dönüşüm $O(n)$ zaman alır. \$ yoksa, BANANA örneğindeki gibi orijinal satır indisi (3) saklanır.

ÖZET

Özet

Konu	Hatırlanacaklar
Sözlüksel sıra	ilk farklı karakter karar verir; önek önce gelir; kod sırası $\neq$ Türkçe alfabe
Basamak sıralaması	LSD: sağdan sola kararlı saymalı sıralama, $O(W \cdot (N + R))$; MSD: soldan, kovalarda özyineleme
Düzenli ifade	. * + ? [] ^ \$ () { } \d \w \s; bakış ifadeleri karakter tüketmez
FSM	$(Q, \Sigma, \delta, q_0, F)$; DFA'da tek geçiş, $O(n)$ tarama; $NFA \equiv DFA \equiv$ düzenli ifade
Levenshtein	$1 + \min(\text{sil, ekle, değiştir})$, eşitse sol üst; $O(m \cdot n)$
LCS	eşitse sol üst + 1, değilse $\max(\text{üst, sol})$; geri okumayla dizi
Sonek dizisi	sıralı sonek indisleri; kaba kuvvet $O(n^2 \log n)$, ikiye katlama $O(n \log n)$; arama $O(m \log n)$
BWT	sıralı döndürmelerin son sütunu + satır indisi; LF eşlemesiyle geri dönüş; bzip2

Etkileşimli Simülatörler

Dizgi Sıralama: LSD Basamak Sıralaması

Virgülle ayrılmış dizgileri sağdan sola, karakter karakter kovalara dağıtarak sıralayın; her geçişte kovaları ve diziyi izleyin.

Dizgi Ayırıştırma: FSM

Bir tanımlayıcıyı (harf ya da alt çizgiyle başlayan) sonlu durum makinesiyle karakter karakter doğrulayın; geçiş tablosundaki satır vurgulanır.

Dizgi Benzerliği: Levenshtein

İki dizgi girin (ör. kitten · sitting); DP tablosunu hücre hücre doldurun ve ekleme, silme, değiştirme maliyetlerini görün.

Dizgi Dönüşümü: BWT

Tüm döndürmeleri üretin, sıralayın ve son sütunu okuyarak Burrows–Wheeler dönüşümünü adım adım elde edin.

ÖZET

Sık Yapılan Hatalar

Kod sırasını alfabe sanmak

"Zebra" < "apple" ve "şeker" > "zeytin" çıkar.
Büyük/küçük harf ve Türkçe için `Collator` ya da normalleştirme gerekir.

Sayıları dizgi olarak sıralamak

"10" < "9" olur. Sayısal sıra için tam sayıya çevirin ya da sıfırla aynı uzunluğa getirin.

LSD'de kararsız alt sıralama

Her geçiş kararlı olmalıdır; aksi hâlde önceki geçişlerin sırası bozulur. Saymalı sıralamada yerleştirme sondan başa yapılır.

Kaçışsız nokta, açgözlü .*

`a.b` "a+b" ile de eşleşir; `<.*>` tüm satırı yutar.
`\.` ve tembel `.*?` kullanın. `(a+)+$` gibi ifadeler üstel sayıda geri dönüşe yol açar.

DP tablosunda indis kayması

Tablo $(m+1) \times (n+1)$ 'dir; $D[i][j]$ için `charAt(i - 1)` karşılaştırılır. İlk satır/sütunu başlatmamak sonucu bozar.

Alt dizi ile alt dizgiyi karıştırmak

LCS ardışıklık istemez; en uzun ortak alt dizgi ister.
BWT'de de satır indisini ya da `$` 'ı saklamayı unutmak dönüşümü geri alınamaz kılar.

ÖZET

Örnek Problemler

1. "10", "9", "100", "2" dizgilerini sözlüksel sıraya koyun.
2. [329, 457, 657, 839, 436, 720, 355] dizisine LSD basamak sıralaması uygulayın; her geçişten sonraki diziye yazın.
3. $\text{^}\backslash\text{d}\{3\}\text{-}\backslash\text{d}\{4\}\text{\$}$ ifadesi "555-1234", "5551234", "55-12345" dizgilerinden hangileriyle eşleşir?
4. Levenshtein("sunday", "saturday") kaçtır?
5. LCS("ABCBADAB", "BDCABA") uzunluğu kaçtır? Bir LCS yazın.
6. "kayak" dizgisinin BWT'sini ve orijinal satır indisini bulun.

► Cevapları göster

ÖZET

Alıştırmalar

1. MSD basamak sıralamasını özyinelemeli olarak yazın.
2. Türkiye cep telefonu biçimini (0532 123 45 67) doğrulayan bir düzenli ifade yazın.
3. İkili dizgilerde 1 sayısı çift olanları kabul eden bir DFA çizin ve geçiş tablosunu yazın.
4. Levenshtein'ı yalnızca iki satırlık bellekle, $O(\min(m, n))$ ek alanla hesaplayın.
5. Sayı ayrıştıran FSM "... ve 5." girdisinde "5." yazdırır. Makineyi düzeltin.
6. "mississippi" metninin sonek dizisini bulun.

► Cevaplar / ipuçları

Kaynaklar

- Sedgewick ve Wayne, *Algorithms*, 4. baskı, 2011 (5.1 String Sorts, 5.4 Regular Expressions).
- Cormen, Leiserson, Rivest ve Stein, *Introduction to Algorithms*, 4. baskı, 2022 (LCS, dizgi eşleştirme otomatları).
- Hopcroft, Motwani ve Ullman, *Introduction to Automata Theory, Languages, and Computation*, 3. baskı, 2006.
- V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals", *Soviet Physics Doklady* 10(8), 1966.
- R. A. Wagner ve M. J. Fischer, "The String-to-String Correction Problem", *Journal of the ACM* 21(1), 1974.
- U. Manber ve G. Myers, "Suffix Arrays: A New Method for On-Line String Searches", *SIAM Journal on Computing* 22(5), 1993.
- M. Burrows ve D. J. Wheeler, "A Block-sorting Lossless Data Compression Algorithm", DEC SRC Research Report 124, 1994.
- J. E. F. Friedl, *Mastering Regular Expressions*, 3. baskı, O'Reilly, 2006; Java `java.util.regex.Pattern` belgeleri.

Son

Bölüm 5: Dizgi Algoritmaları (2)

Sonraki bölüm: Bölüm 5: Trie Veri Yapısı · [Terim sözlüğü](#)