Bölüm 5.2: İleri Dizgi Algoritmaları

Algoritmalar

Sözlüksel sıralama | Basamak sıralaması | Düzenli ifadeler | Sonlu durum makineleri

Levenshtein uzaklığı | Benzerlik ölçüleri | Sonek dizisi | Burrows–Wheeler dönüşümü

Sercan KÜLCÜ · Giresun Üniversitesi · Bilgisayar Mühendisliği

GİRİŞ

Öğrenme Çıktıları

Sıralama

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

Ayrıştırma

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

Benzerlik

Levenshtein uzaklığını dinamik programlama tablosuyla hesaplamak, düzenleme dizisini tablodan geri okumak; Hamming, LCS ve ortak alt dizgi ölçüleriyle karşılaştırmak.

Dönüşüm

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

GİRİŞ

Dizgi Algoritmaları: Dört Problem Ailesi

Dizgiler (string) üzerinde çalışan dört algoritma ailesi:

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**: göreli 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)

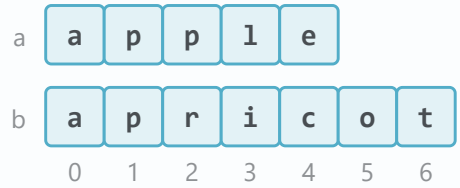
- **Sözlüksel (alfabetik) sıralama** (lexicographic order), öğeleri karakterlerin alfabedeki ya da kod tablosundaki **konumlarına** göre dizer.
- Karşılaştırma **ilk karakterlerle** başlar; eşitse bir **sonraki** karakterlere bakılır.
- İlk **farklı** karakter sonucu belirler: "apple" < "banana", çünkü '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)

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

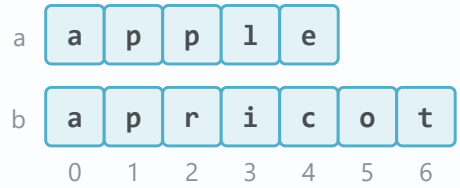
DEĞİŞKENLER

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

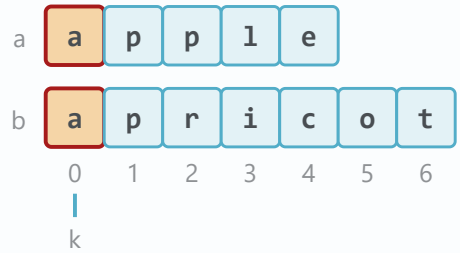
DEĞİŞKENLER

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());
▶   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

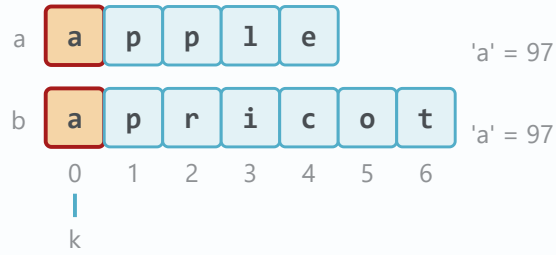
DEĞİŞKENLER

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++) {
▶     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

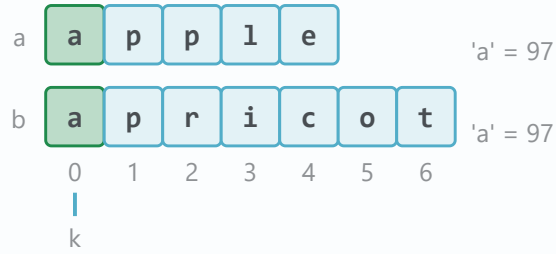
DEĞİŞKENLER

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);
▶     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

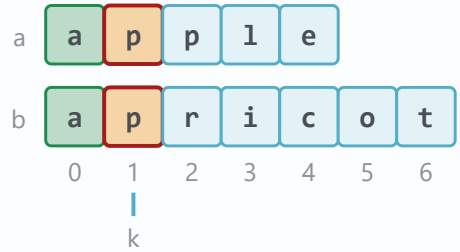
DEĞİŞKENLER

`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());
▶   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

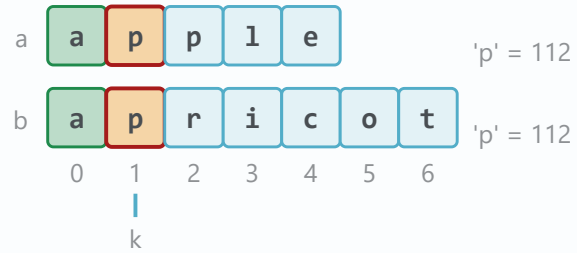
DEĞİŞKENLER

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++) {
▶     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

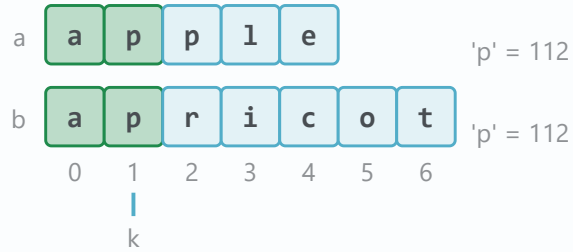
DEĞİŞKENLER

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);
▶     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

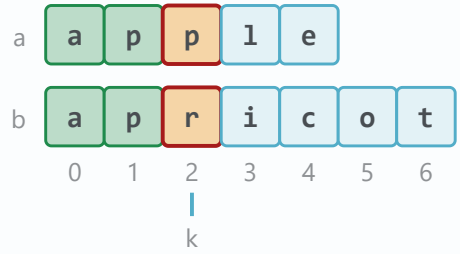
DEĞİŞKENLER

`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());
▶   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

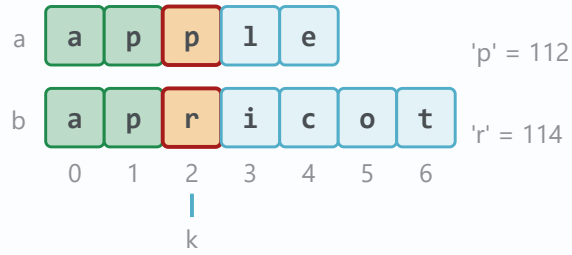
DEĞİŞKENLER

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++) {
▶     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

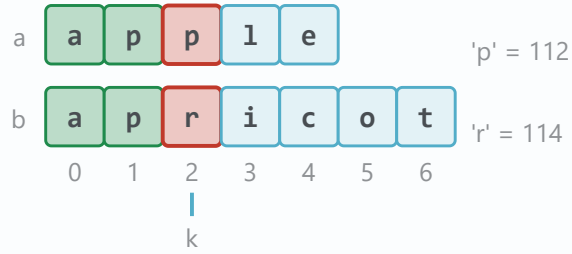
DEĞİŞKENLER

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);
▶     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).

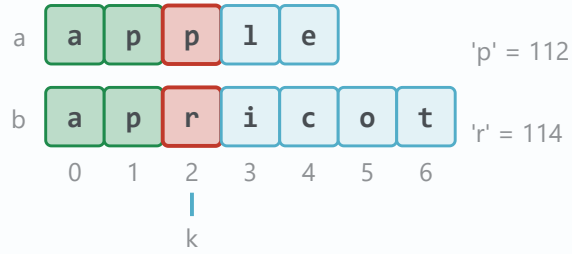
DEĞİŞKENLER

`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)
▶       return x - y;   // ilk farklı karakter
7   }
8   return a.length() - b.length();
9 }
```



return 112 − 114 = **−2** < 0 → **apple, apricot'tan önce** gelir. Kalan karakterlere hiç bakılmaz.

Adım 12 / 12

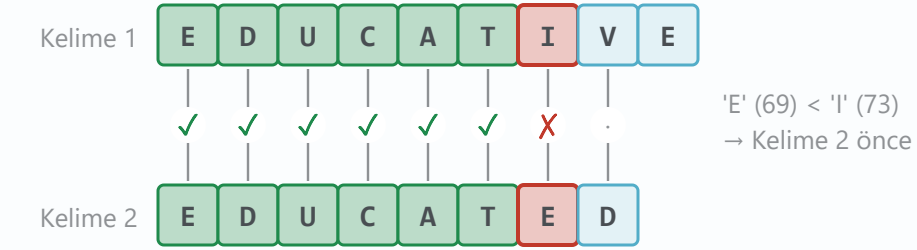
DEĞİŞKENLER

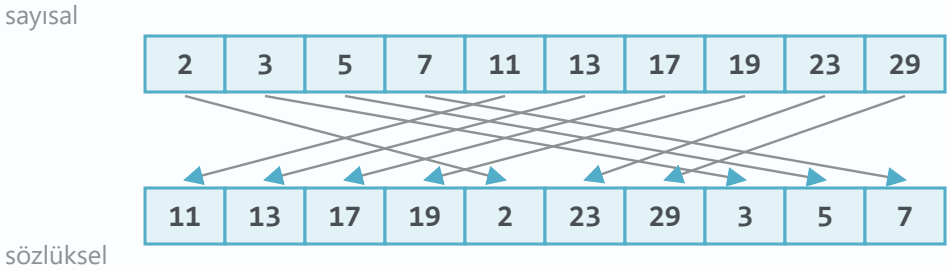
`a = "apple"` `b = "apricot"` `n = 5` `k = 2` `x = 'p'` `y = 'r'`

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").

Karşılaştırmalı Dizgi Sıralama

```
void sozlukselSirala(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;
      }
}
```

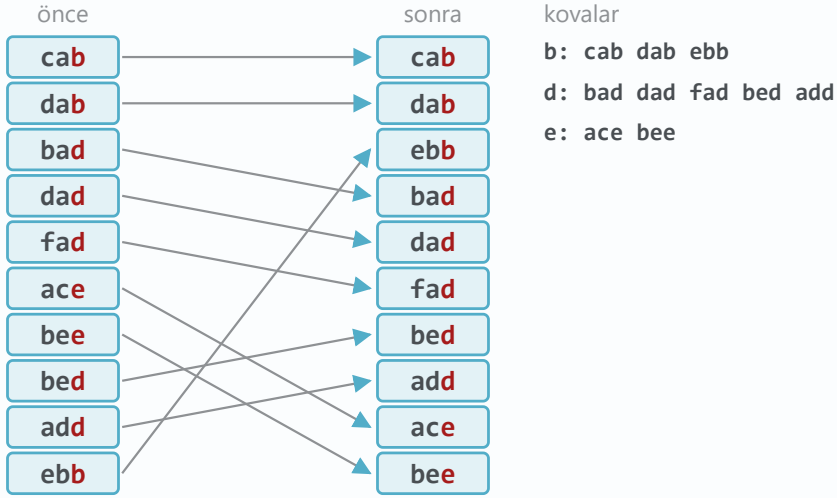
- Her konuma, kalanların sözlükçe en küçüğü takaslarla getirilir.
- `compareTo` en fazla W karakter okur (W: en uzun dizgi).
- Zaman: $O(N^2)$ karşılaştırma × $O(W)$ = **$O(N^2 \cdot W)$**.
- Birleştirme sıralaması ile O(N log N) karşılaştırma → **$O(W \cdot N \log N)$**.

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

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

LSD Basamak Sıralaması: Dizgiler

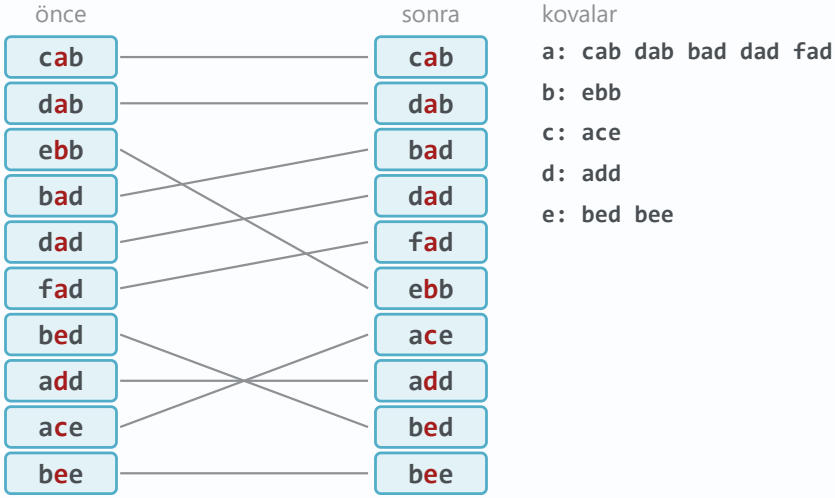
LSD: sağdan sola, her geçişte o karaktere göre **kararlı** saymalı sıralama (kısa dizgiye boş karakter); eşitler önceki geçişin sırasını korur. Örnek: **cab, dab, bad, dad, fad, ace, bee, bed, add, ebb** (W = 3).



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

LSD Basamak Sıralaması: Dizgiler

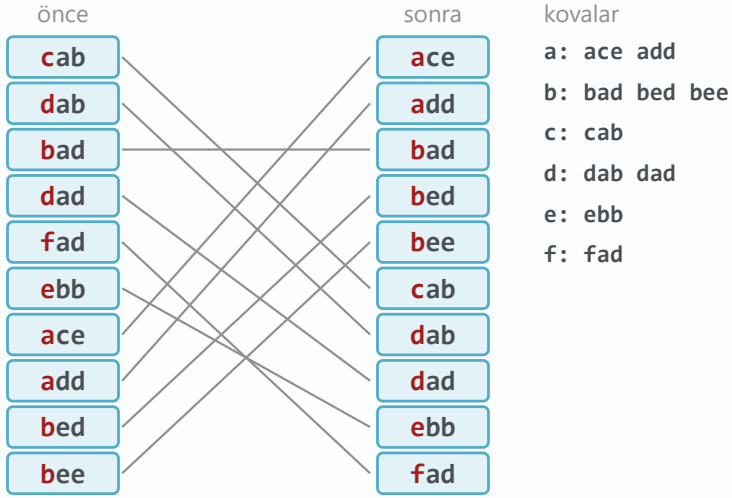
LSD: sağdan sola, her geçişte o karaktere göre **kararlı** saymalı sıralama (kısa dizgiye boş karakter); eşitler önceki geçişin sırasını korur. Örnek: **cab, dab, bad, dad, fad, ace, bee, bed, add, ebb** (W = 3).



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

LSD Basamak Sıralaması: Dizgiler

LSD: sağdan sola, her geçişte o karaktere göre **kararlı** saymalı sıralama (kısa dizgiye boş karakter); eşitler önceki geçişin sırasını korur. Örnek: **cab, dab, bad, dad, fad, ace, bee, bed, add, ebb** (W = 3).



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

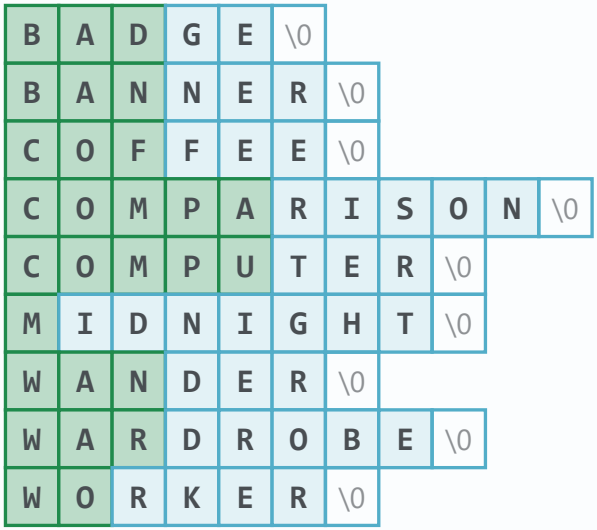
LSD Basamak Sıralaması: Java Kodu

```
void radixSirala(String[] a) {
  int W = enUzun(a);                // en uzun dizgi
  for (int d = W - 1; d >= 0; d--)
    sayarakSirala(a, d);            // sağdan sola
}
void sayarakSirala(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 kovasının **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 → O(W · (N + R)).

MSD Basamak Sıralaması: Soldan Sağa



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 önek). 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 → **O(W · (N + R))**.
- Alfabe sabitse (R = 256) maliyet **O(W · N)** olur: en uzun dizginin uzunluğu × öğe sayısı.

Yöntem	Zaman	Ek bellek	Kararlı
Yer değiştirmeli sıralama + compareTo	$O(N^2 \cdot W)$	O(1)	hayır
Birleştirme sıralaması + compareTo	O(W · N log N)	O(N)	evet
LSD basamak sıralaması	O(W · (N + R))	O(N + R)	evet
MSD basamak sıralaması	O(N · W) en kötü	O(N + W · 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
`colou?r`	color colour	color, colour
`gr(a|e)y`	gray grey groy	gray, grey
`\d+`	20.99 TL, 5 adet	20, 99, 5
`\bkod\b`	kod kodlama	yalnızca "kod"

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

DÜZENLİ İFADELER

Temel İşleçler 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 İşleçler ve Özel Karakterler

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

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

Gruplama

`(abc)`	tam olarak "abc"; grup olarak yakalanır
`(a|b|c)`	'a', 'b' ya da 'c'; grup olarak yakalanır
`(ab)+`	"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: `"<.+?>"` , "x" içinde bütün metni değil, önce yalnızca "" parçasını alır.

İleri ve Geri Bakış (Lookahead, Lookbehind)

Örüntü	Adı	Eşleşme koşulu	"ab ac ba"
`a(?=b)`	ileri bakış (lookahead)	'a', ardından 'b' **geliyorsa**	ab ac ba
`a(?!b)`	olumsuz ileri bakış	'a', ardından 'b' **gelmiyorsa**	ab ac ba
`(?<=b)a`	geri bakış (lookbehind)	'a', önünde 'b' **varsa**	ab ac ba
`(?<!b)a`	olumsuz geri bakış	'a', önünde 'b' **yoksa**	ab ac 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.

Örnek: Doğrulama İfadeleri

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

E-posta

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

Telefon

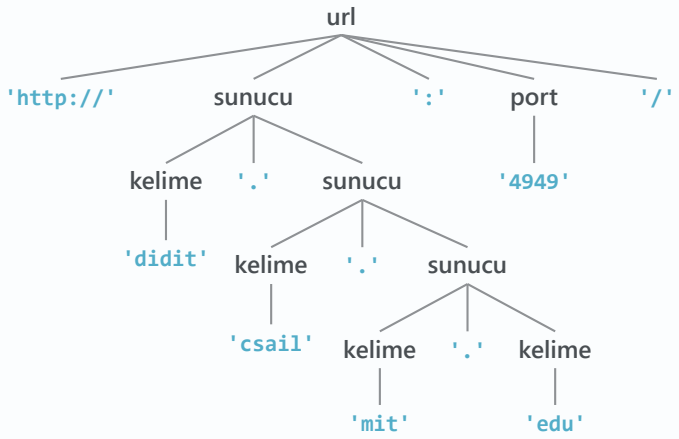
İsteğe bağlı `(…)?` ülke kodu: + ve 1–3 rakam, ardından tire ya da boşluk; sonra tam 10 rakam. ✓ +90 5321234567 ✓ 5321234567 ✗ 0532 123 45 67

URL

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

DÜZENLİ İFADELER

Deneyin: Düzenli İfade Sınayıcı

Örüntü `\d+` büyük/küçük harf duyarsız

Metin `Fiyat 20.99 TL, adet 5, kod A7`

\d+ | colou?r | e-posta | a(?=b) | gruplar

Fiyat 20.99 TL, adet 5, kod A7

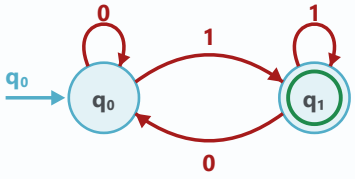
4 eşleşme bulundu.

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, Σ, δ, q_0, F)**



Q = {q_0, q_1}, Σ = {0, 1}
F = {q_1} (çift halka)
"1 ile biten" dizgiler

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)
δ(durum, sembol) → yeni durum.

Kabul durumları F ⊆ 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) tanır: 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.

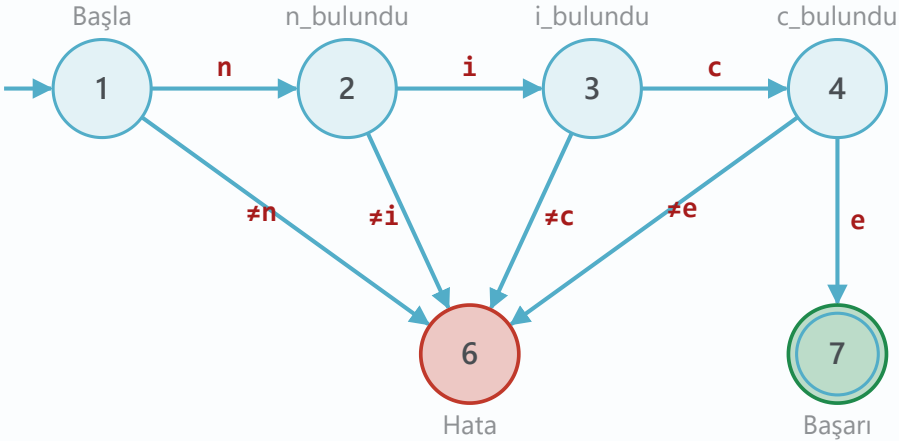
Kullanımı: DFA Simülasyonu

1. **Başlangıç** durumuna geçilir.
2. Her karakter için durumu **geçiş tablosundan** güncellenir.
3. Dizgi bitince son durumun **kabul durumu** olup olmadığı denetlenir.

```
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?
}
```

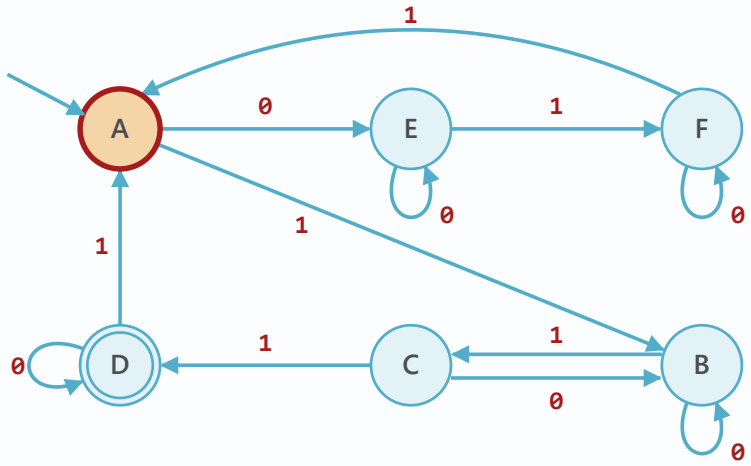
Geçiş tablosu `int[|Q|][|Σ|]` boyutundadır. Her karakter için bir tablo okuması yapılır: n uzunluklu dizgi **O(n)** zamanda, geri dönmeden (backtracking olmadan) işlenir.

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



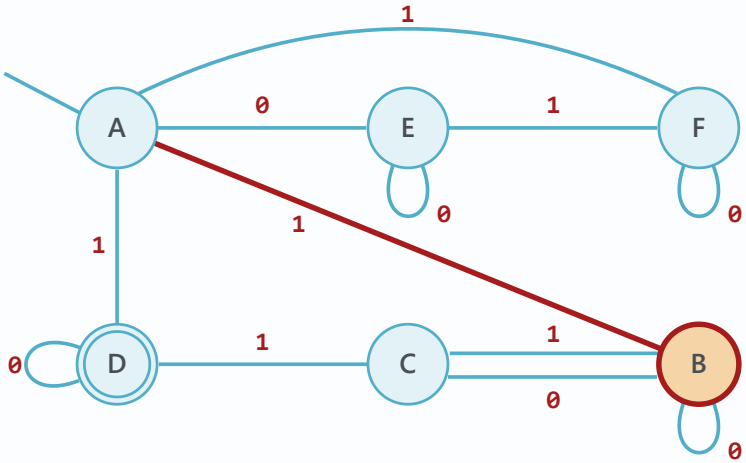
Başlangıç: durum **A**. Girdi: **10011**.

Adım 1 / 6

δ	0	1
→ A	E	B
B	B	C
C	B	D
D *	D	A
E	E	F
F	F	A

→ başlangıç, * kabul. Her hücrede tek durum: **DFA**.

Örnek: İkili Alfabe Üzerinde DFA

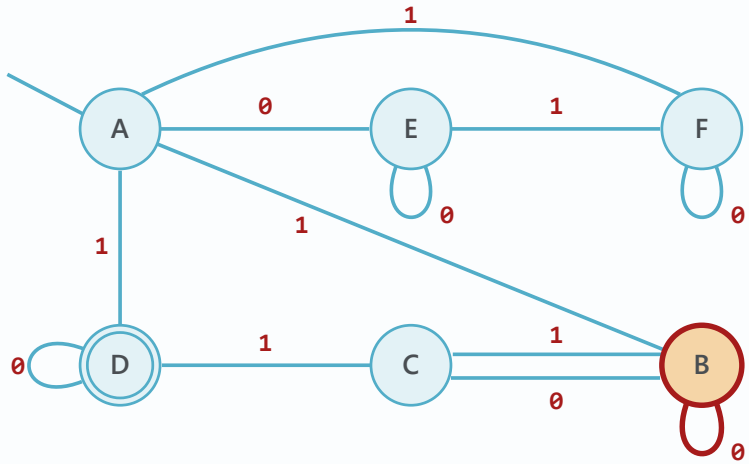


1. sembol **1**: δ(A, 1) = **B**.

δ	0	1
→ A	E	B
B	B	C
C	B	D
D *	D	A
E	E	F
F	F	A

→ başlangıç, * kabul. Her hücrede tek durum: **DFA**.

Örnek: İkili Alfabe Üzerinde DFA

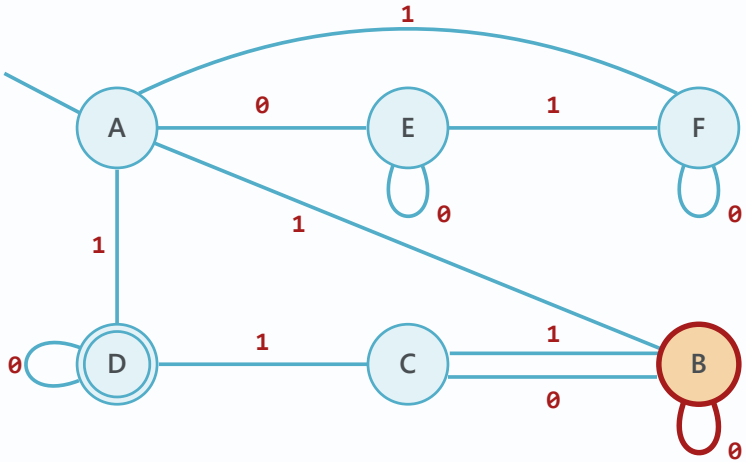


2. sembol **0**: δ(B, 0) = **B**.

δ	0	1
→ A	E	B
B	B	C
C	B	D
D *	D	A
E	E	F
F	F	A

→ başlangıç, * kabul. Her hücrede tek durum: **DFA**.

Örnek: İkili Alfabe Üzerinde DFA

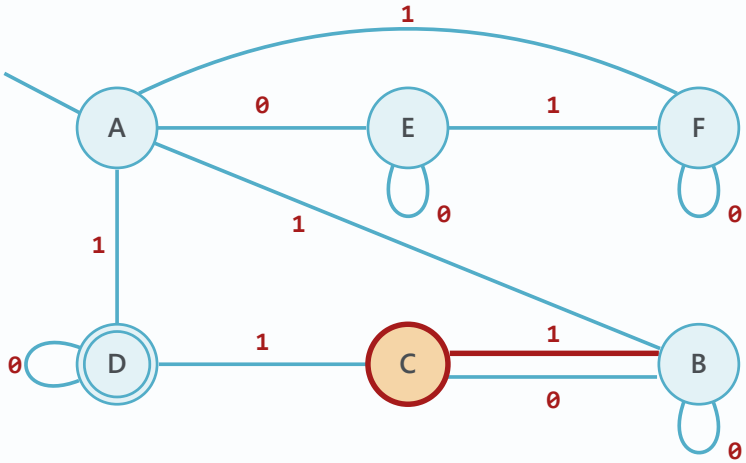


3. sembol **0**: δ(B, 0) = **B**.

δ	0	1
→ A	E	B
B	B	C
C	B	D
D *	D	A
E	E	F
F	F	A

→ başlangıç, * kabul. Her hücrede tek durum: **DFA**.

Örnek: İkili Alfabe Üzerinde DFA

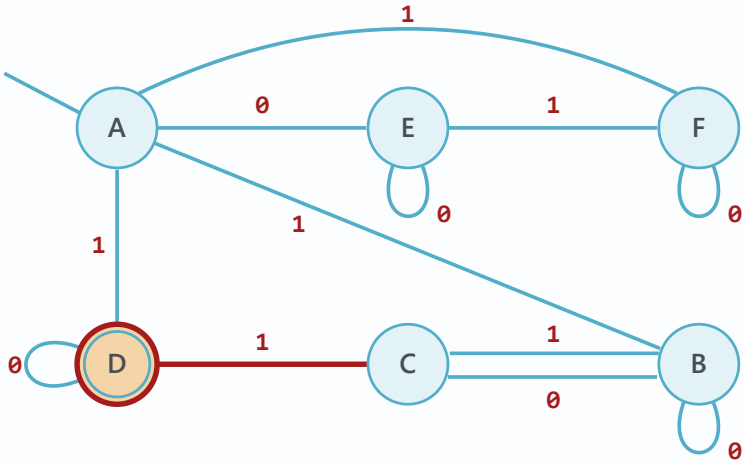


4. sembol **1**: δ(B, 1) = **C**.

δ	0	1
→ A	E	B
B	B	C
C	B	D
D *	D	A
E	E	F
F	F	A

→ başlangıç, * kabul. Her hücrede tek durum: **DFA**.

Örnek: İkili Alfabe Üzerinde DFA

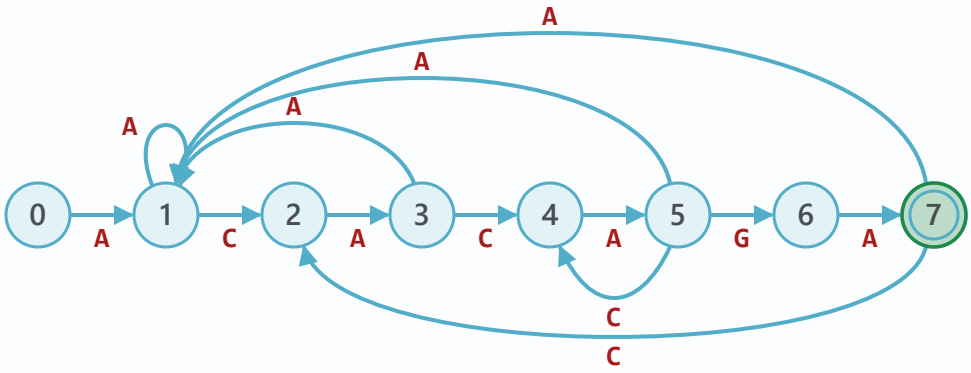


δ	0	1
→ A	E	B
B	B	C
C	B	D
D *	D	A
E	E	F
F	F	A

5. sembol **1**: δ(C, 1) = **D**. Girdi bitti; D kabul durumu → **kabul**.

→ başlangıç, * kabul. Her hücrede tek durum: **DFA**.

Örnek: "ACACAGA" Örüntüsünü Arayan Otomat

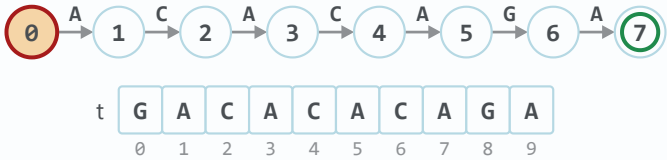


metin	G	A	C	A	C	A	C	A	G	A
durum	0	1	2	3	4	5	4	5	6	7

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

dfaAra(dfa, "GACACACAGA", 7)

```
1 int dfaAra(int[][] dfa, String t, int m) {
▶   int q = 0;                          // durum
3   for (int i = 0; i < t.length(); i++) {
4     q = dfa[q][t.charAt(i)];
5     if (q == m)
6       return i - m + 1;               // eşleşme
7   }
8   return -1;
9 }
```



Otomat ACACAGA için önceden kuruldu. Durum q = metnin sonunda örüntüyle uyuşan en uzun önekin uzunluğu; çizilmeyen geçişler 0'a gider. Başlangıçta q = 0.

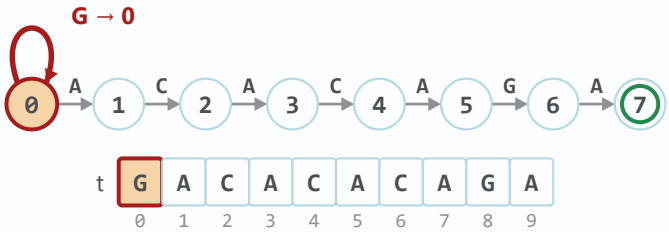
Adım 1 / 11

DEĞİŞKENLER

`m = 7` `i = –` `q = 0`

dfaAra(dfa, "GACACACAGA", 7)

```
1 int dfaAra(int[][] dfa, String t, int m) {
2   int q = 0;                          // durum
▶   for (int i = 0; i < t.length(); i++) {
▶     q = dfa[q][t.charAt(i)];
▶     if (q == m)
6       return i - m + 1;               // eşleşme
7   }
8   return -1;
9 }
```



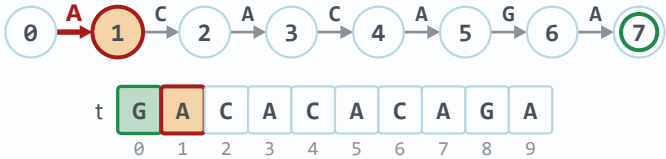
t[0] = 'G' örüntünün hiçbir önekini sürdürmez: q = 0'da kalır.

DEĞİŞKENLER

`m = 7` `i = 0` `q = 0`

dfaAra(dfa, "GACACACAGA", 7)

```
1 int dfaAra(int[][] dfa, String t, int m) {
2   int q = 0;                          // durum
▶   for (int i = 0; i < t.length(); i++) {
▶     q = dfa[q][t.charAt(i)];
▶     if (q == m)
6       return i - m + 1;               // eşleşme
7   }
8   return -1;
9 }
```



t[1] = 'A' örüntünün sıradaki karakteri: 0 → 1.

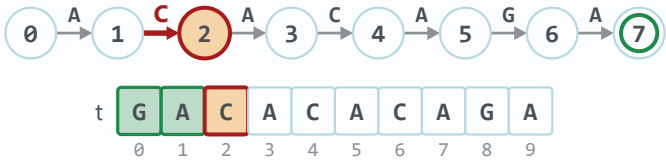
Adım 3 / 11

DEĞİŞKENLER

`m = 7` `i = 1` `q = 1`

dfaAra(dfa, "GACACACAGA", 7)

```
1 int dfaAra(int[][] dfa, String t, int m) {
2   int q = 0;                          // durum
▶   for (int i = 0; i < t.length(); i++) {
▶     q = dfa[q][t.charAt(i)];
▶     if (q == m)
6       return i - m + 1;               // eşleşme
7   }
8   return -1;
9 }
```



t[2] = 'C' örüntünün sıradaki karakteri: 1 → 2.

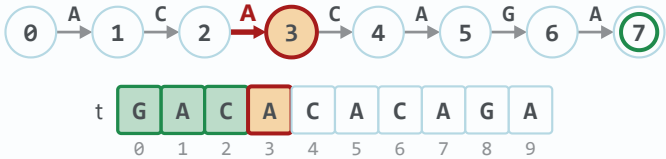
Adım 4 / 11

DEĞİŞKENLER

`m = 7` `i = 2` `q = 2`

dfaAra(dfa, "GACACACAGA", 7)

```
1 int dfaAra(int[][] dfa, String t, int m) {
2   int q = 0;                          // durum
▶   for (int i = 0; i < t.length(); i++) {
▶     q = dfa[q][t.charAt(i)];
▶     if (q == m)
6       return i - m + 1;               // eşleşme
7   }
8   return -1;
9 }
```



t[3] = 'A' örüntünün sıradaki karakteri: 2 → 3.

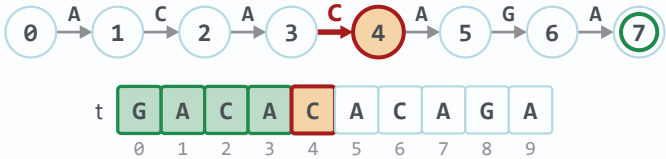
Adım 5 / 11

DEĞİŞKENLER

`m = 7` `i = 3` `q = 3`

dfaAra(dfa, "GACACACAGA", 7)

```
1 int dfaAra(int[][] dfa, String t, int m) {
2   int q = 0;                          // durum
▶   for (int i = 0; i < t.length(); i++) {
▶     q = dfa[q][t.charAt(i)];
▶     if (q == m)
6       return i - m + 1;               // eşleşme
7   }
8   return -1;
9 }
```



t[4] = 'C' örüntünün sıradaki karakteri: 3 → 4.

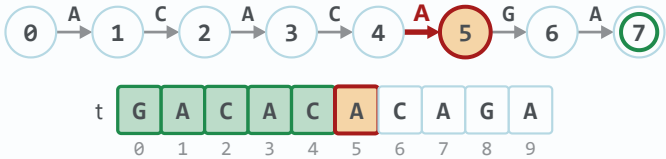
Adım 6 / 11

DEĞİŞKENLER

`m = 7` `i = 4` `q = 4`

dfaAra(dfa, "GACACACAGA", 7)

```
1 int dfaAra(int[][] dfa, String t, int m) {
2   int q = 0;                          // durum
▶   for (int i = 0; i < t.length(); i++) {
▶     q = dfa[q][t.charAt(i)];
▶     if (q == m)
6       return i - m + 1;               // eşleşme
7   }
8   return -1;
9 }
```



t[5] = 'A' örüntünün sıradaki karakteri: 4 → 5.

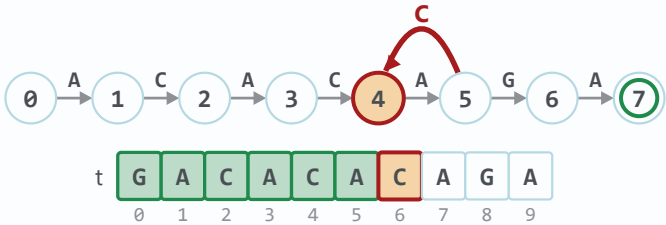
Adım 7 / 11

DEĞİŞKENLER

`m = 7` `i = 5` `q = 5`

dfaAra(dfa, "GACACACAGA", 7)

```
1 int dfaAra(int[][] dfa, String t, int m) {
2   int q = 0;                          // durum
▶   for (int i = 0; i < t.length(); i++) {
▶     q = dfa[q][t.charAt(i)];
▶     if (q == m)
6       return i - m + 1;               // eşleşme
7   }
8   return -1;
9 }
```



t[6] = 'C' beklenen 'G' değil, ama q sıfırlanmaz: "ACACAC"ın sonu "ACAC" önekidir → 5 → **4**. Metinde geri dönülmez.

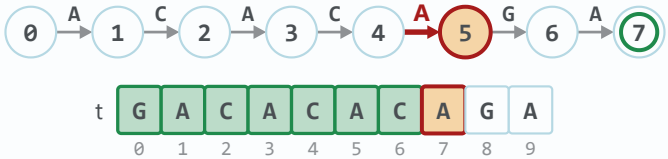
Adım 8 / 11

DEĞİŞKENLER

`m = 7` `i = 6` `q = 4`

dfaAra(dfa, "GACACACAGA", 7)

```
1 int dfaAra(int[][] dfa, String t, int m) {
2   int q = 0;                          // durum
▶   for (int i = 0; i < t.length(); i++) {
▶     q = dfa[q][t.charAt(i)];
▶     if (q == m)
6       return i - m + 1;               // eşleşme
7   }
8   return -1;
9 }
```



t[7] = 'A' örüntünün sıradaki karakteri: 4 → 5.

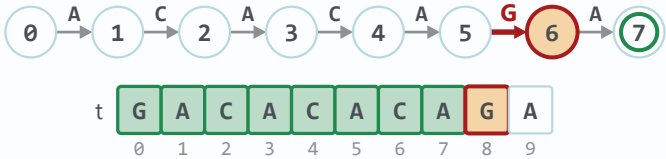
Adım 9 / 11

DEĞİŞKENLER

m = 7 i = 7 q = 5

dfaAra(dfa, "GACACACAGA", 7)

```
1 int dfaAra(int[][] dfa, String t, int m) {
2   int q = 0;                          // durum
▶   for (int i = 0; i < t.length(); i++) {
▶     q = dfa[q][t.charAt(i)];
▶     if (q == m)
6       return i - m + 1;               // eşleşme
7   }
8   return -1;
9 }
```



t[8] = 'G' örüntünün sıradaki karakteri: 5 → 6.

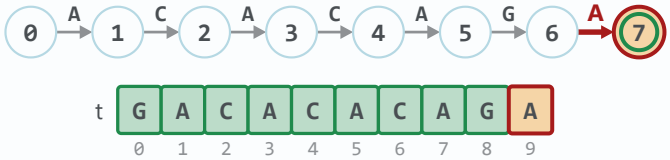
Adım 10 / 11

DEĞİŞKENLER

`m = 7` `i = 8` `q = 6`

dfaAra(dfa, "GACACACAGA", 7)

```
1 int dfaAra(int[][] dfa, String t, int m) {
2   int q = 0;                          // durum
3   for (int i = 0; i < t.length(); i++) {
      q = dfa[q][t.charAt(i)];
      if (q == m)
        return i - m + 1;               // eşleşme
7   }
8   return -1;
9 }
```



'A' ile 6 → **7** = m: örüntü bulundu, i − m + 1 = 9 − 7 + 1 = **3** döndürülür.

Adım 11 / 11

DEĞİŞKENLER

`m = 7` `i = 9` `q = 7`

FSM ile Sayı Ayrıştırma: Kodu İzle

```
1  void sayilariBul(String s) {
▶    Durum d = Durum.BASLA;
▶    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)
   System.out.println(tampon);
19 }
```



BASLA | RAKAM | ONDALIK

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

Adım 1 / 14

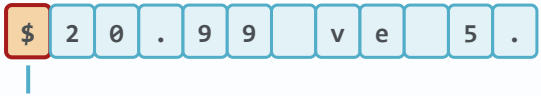
DEĞİŞKENLER

`c = –` `rakam = –` `d = BASLA` `tampon = ""`

ÇIKTI

FSM ile Sayı Ayrıştırma: Kodu İzle

```
1  void sayilariBul(String s) {
2    Durum d = Durum.BASLA;
3    StringBuilder tampon = new StringBuilder();
4    for (char c : s.toCharArray()) {
▶      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)
   System.out.println(tampon);
19 }
```



BASLA | RAKAM | ONDALIK

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

Adım 2 / 14

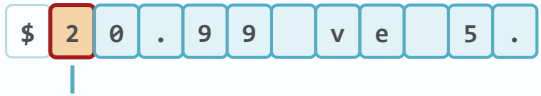
DEĞİŞKENLER

`c = '$'` `rakam = false` `d = BASLA` `tampon = ""`

ÇIKTI

FSM ile Sayı Ayrıştırma: 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);
▶      if (rakam) {
▶        if (d == Durum.BASLA) d = Durum.RAKAM;
▶        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)
   System.out.println(tampon);
19 }
```



BASLA | **RAKAM** | ONDALIK

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

Adım 3 / 14

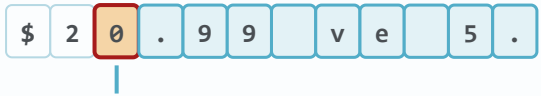
DEĞİŞKENLER

`c = '2'` `rakam = true` `d = RAKAM` `tampon = "2"`

ÇIKTI

FSM ile Sayı Ayrıştırma: 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);
▶      if (rakam) {
7        if (d == Durum.BASLA) d = Durum.RAKAM;
▶        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)
   System.out.println(tampon);
19 }
```



BASLA | **RAKAM** | ONDALIK

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

Adım 4 / 14

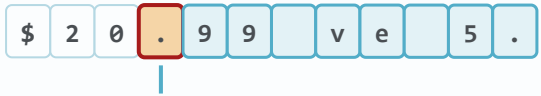
DEĞİŞKENLER

`c = '0'` `rakam = true` `d = RAKAM` `tampon = "20"`

ÇIKTI

FSM ile Sayı Ayrıştırma: 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);
▶      } else if (c == '.' && d != Durum.ONDALIK) {
▶        d = Durum.ONDALIK;
▶        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)
   System.out.println(tampon);
19 }
```



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 Ayrıştırma: 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);
▶      if (rakam) {
7        if (d == Durum.BASLA) d = Durum.RAKAM;
▶        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)
   System.out.println(tampon);
19 }
```



BASLA | RAKAM | ONDALIK

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

Adım 6 / 14

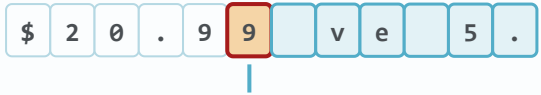
DEĞİŞKENLER

`c = '9'` `rakam = true` `d = ONDALIK` `tampon = "20.9"`

ÇIKTI

FSM ile Sayı Ayrıştırma: 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);
▶      if (rakam) {
7        if (d == Durum.BASLA) d = Durum.RAKAM;
▶        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)
   System.out.println(tampon);
19 }
```



BASLA | RAKAM | ONDALIK

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

Adım 7 / 14

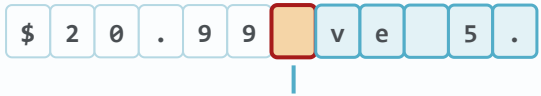
DEĞİŞKENLER

`c = '9'` `rakam = true` `d = ONDALIK` `tampon = "20.99"`

ÇIKTI

FSM ile Sayı Ayrıştırma: 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);
▶      } else if (d != Durum.BASLA) {  // sayı bitti
▶        System.out.println(tampon);
▶        tampon.setLength(0);
▶        d = Durum.BASLA;
16     }
17   }
18   if (d != Durum.BASLA)
   System.out.println(tampon);
19 }
```



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 Ayrıştırma: Kodu İzle

```
1  void sayilariBul(String s) {
2    Durum d = Durum.BASLA;
3    StringBuilder tampon = new StringBuilder();
4    for (char c : s.toCharArray()) {
▶      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)
   System.out.println(tampon);
19 }
```



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 Ayrıştırma: Kodu İzle

```
1  void sayilariBul(String s) {
2    Durum d = Durum.BASLA;
3    StringBuilder tampon = new StringBuilder();
4    for (char c : s.toCharArray()) {
▶      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)
   System.out.println(tampon);
19 }
```



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 Ayrıştırma: Kodu İzle

```
1  void sayilariBul(String s) {
2    Durum d = Durum.BASLA;
3    StringBuilder tampon = new StringBuilder();
4    for (char c : s.toCharArray()) {
▶      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)
   System.out.println(tampon);
19 }
```



BASLA | RAKAM | ONDALIK

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

Adım 11 / 14

DEĞİŞKENLER

`c = ' '` `rakam = false` `d = BASLA` `tampon = ""`

ÇIKTI

```
20.99
```

FSM ile Sayı Ayrıştırma: 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);
▶      if (rakam) {
▶        if (d == Durum.BASLA) d = Durum.RAKAM;
▶        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)
   System.out.println(tampon);
19 }
```



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 Ayrıştırma: 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);
▶      } else if (c == '.' && d != Durum.ONDALIK) {
▶        d = Durum.ONDALIK;
▶        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)
   System.out.println(tampon);
19 }
```



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 Ayrıştırma: 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   }
▶    if (d != Durum.BASLA)
   System.out.println(tampon);
19 }
```



BASLA | RAKAM | ONDALIK

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

Adım 14 / 14

DEĞİŞKENLER

`c = –` `rakam = –` `d = ONDALIK` `tampon = "5."`

ÇIKTI

```
20.99
5.
```

DÜZENLEME UZAKLIĞI

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"

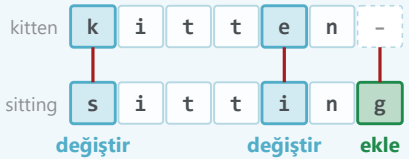
Silme (deletion)

Bir karakteri çıkarmak: "shine" → "shin"

Değiştirme (substitution)

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

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 değiştirme, U ekleme, I ekleme: d = 3

(a) 3	(b) 3	(c) 4
rain	shine	shine
sain	rhine	tshine
shin	raine	trhine
shine	raine	traine
		traine

değiştirme ekleme silme

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

DÜZENLEME UZAKLIĞI

Levenshtein Uzaklığını Hesaplama

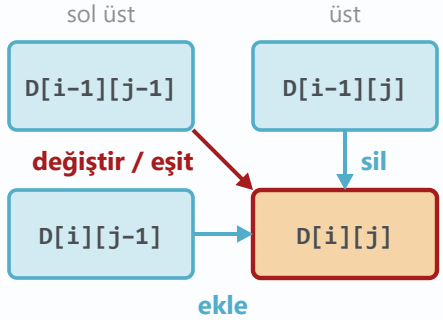
D[i][j] = a'nın ilk **i** karakterini b'nin ilk **j** karakterine dönüştürmenin en az maliyeti.

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] silme, D[i][j−1] ekleme, D[i−1][j−1] değiştirme)

- **Dinamik programlama**: her alt problem bir kez çözülür, (m+1) × (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 · n)**; bellek O(m · 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 (ε → P, PA, … : ekleme), ilk sütun 0..4 (C, CA, … → ε : 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 ≠ 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 değiştirme, A eşit, **I ekleme**, R ve S eşit: CARS → PARS → PAIRS.

Levenshtein: Java Kodu

```java
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 önek** içindir.
- Örnekler: kitten → sitting = 3, ercan → sercan = 1, apple → banana = 5.

levenshtein("kar", "ark")

```
1  int levenshtein(String a, String b) {
2    int m = a.length(), n = b.length();
3    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
8    for (int i = 1; i <= m; i++)
9      for (int j = 1; j <= n; j++)
10       if (a.charAt(i-1) == b.charAt(j-1))
11         D[i][j] = D[i-1][j-1];  // eşit
12       else
13         D[i][j] = 1 + Math.min(D[i-1][j-1],
14             Math.min(D[i][j-1], D[i-1][j]));
15   return D[m][n];
16 }
```

	ε	a	r	k
ε	0	1	2	3
k	1			
a	2			
r	3			

a = "kar" (satırlar), b = "ark" (sütunlar). 0. sütun i silmeyi, 0. satır j eklemeyi gösterir: boş önekten/öneke dönüşüm maliyeti.

Adım 1 / 11

DEĞİŞKENLER

`m = 3` `n = 3` `i = –` `j = –`

levenshtein("kar", "ark")

```
1  int levenshtein(String a, String b) {
2    int m = a.length(), n = b.length();
3    int[][] D = new int[m + 1][n + 1];
4    for (int i = 0; i <= m; i++)
5      D[i][0] = i;              // silme
6    for (int j = 0; j <= n; j++)
7      D[0][j] = j;              // ekleme
8    for (int i = 1; i <= m; i++)
9      for (int j = 1; j <= n; j++)
▶      if (a.charAt(i-1) == b.charAt(j-1))
11       D[i][j] = D[i-1][j-1];  // eşit
12     else
▶        D[i][j] = 1 + Math.min(D[i-1][j-1],
▶            Math.min(D[i][j-1], D[i-1][j]));
15   return D[m][n];
16 }
```

	ε	a	r	k
ε	0	1	2	3
k	1	1		
a	2			
r	3			

'k' ≠ 'a' → 1 + min(değiştir 0, ekle 1, sil 1) = **1**.

Adım 2 / 11

DEĞİŞKENLER

m = 3 | n = 3 | i = 1 | j = 1

levenshtein("kar", "ark")

```
1  int levenshtein(String a, String b) {
2    int m = a.length(), n = b.length();
3    int[][] D = new int[m + 1][n + 1];
4    for (int i = 0; i <= m; i++)
5      D[i][0] = i;            // silme
6    for (int j = 0; j <= n; j++)
7      D[0][j] = j;            // ekleme
8    for (int i = 1; i <= m; i++)
9      for (int j = 1; j <= n; j++)
▶      if (a.charAt(i-1) == b.charAt(j-1))
11       D[i][j] = D[i-1][j-1];  // eşit
12     else
▶        D[i][j] = 1 + Math.min(D[i-1][j-1],
▶            Math.min(D[i][j-1], D[i-1][j]));
15   return D[m][n];
16 }
```

	ε	a	r	k
ε	0	1	2	3
k	1	1	2	
a	2			
r	3			

'k' ≠ 'r' → 1 + min(değiştir 1, ekle 1, sil 2) = **2**.

Adım 3 / 11

DEĞİŞKENLER

m = 3 | n = 3 | i = 1 | j = 2

levenshtein("kar", "ark")

```
1  int levenshtein(String a, String b) {
2    int m = a.length(), n = b.length();
3    int[][] D = new int[m + 1][n + 1];
4    for (int i = 0; i <= m; i++)
5      D[i][0] = i;              // silme
6    for (int j = 0; j <= n; j++)
7      D[0][j] = j;              // ekleme
8    for (int i = 1; i <= m; i++)
9      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
12     else
13       D[i][j] = 1 + Math.min(D[i-1][j-1],
14           Math.min(D[i][j-1], D[i-1][j]));
15   return D[m][n];
16 }
```

	ε	a	r	k
ε	0	1	2	3
k	1	1	2	2
a	2			
r	3			

a[0] = b[2] = 'k' → maliyetsiz: D[1][3] = sol üst = **2**.

Adım 4 / 11

DEĞİŞKENLER

`m = 3` `n = 3` `i = 1` `j = 3`

levenshtein("kar", "ark")

```
1  int levenshtein(String a, String b) {
2    int m = a.length(), n = b.length();
3    int[][] D = new int[m + 1][n + 1];
4    for (int i = 0; i <= m; i++)
5      D[i][0] = i;            // silme
6    for (int j = 0; j <= n; j++)
7      D[0][j] = j;            // ekleme
8    for (int i = 1; i <= m; i++)
9      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
12       else
13         D[i][j] = 1 + Math.min(D[i-1][j-1],
14             Math.min(D[i][j-1], D[i-1][j]));
15   return D[m][n];
16 }
```

	ε	a	r	k
ε	0	1	2	3
k	1	1	2	2
a	2	1		
r	3			

a[1] = b[0] = 'a' → maliyetsiz: D[2][1] = sol üst = **1**.

Adım 5 / 11

DEĞİŞKENLER

m = 3 | n = 3 | i = 2 | j = 1

levenshtein("kar", "ark")

```
1  int levenshtein(String a, String b) {
2    int m = a.length(), n = b.length();
3    int[][] D = new int[m + 1][n + 1];
4    for (int i = 0; i <= m; i++)
5      D[i][0] = i;            // silme
6    for (int j = 0; j <= n; j++)
7      D[0][j] = j;            // ekleme
8    for (int i = 1; i <= m; i++)
9      for (int j = 1; j <= n; j++)
▶      if (a.charAt(i-1) == b.charAt(j-1))
11       D[i][j] = D[i-1][j-1];  // eşit
12     else
▶        D[i][j] = 1 + Math.min(D[i-1][j-1],
▶            Math.min(D[i][j-1], D[i-1][j]));
15   return D[m][n];
16 }
```

	ε	a	r	k
ε	0	1	2	3
k	1	1	2	2
a	2	1	2	
r	3			

'a' ≠ 'r' → 1 + min(değiştir 1, ekle 1, sil 2) = **2**.

Adım 6 / 11

DEĞİŞKENLER

m = 3 | n = 3 | i = 2 | j = 2

levenshtein("kar", "ark")

```
1  int levenshtein(String a, String b) {
2    int m = a.length(), n = b.length();
3    int[][] D = new int[m + 1][n + 1];
4    for (int i = 0; i <= m; i++)
5      D[i][0] = i;              // silme
6    for (int j = 0; j <= n; j++)
7      D[0][j] = j;              // ekleme
8    for (int i = 1; i <= m; i++)
9      for (int j = 1; j <= n; j++)
▶      if (a.charAt(i-1) == b.charAt(j-1))
11       D[i][j] = D[i-1][j-1];  // eşit
12     else
▶        D[i][j] = 1 + Math.min(D[i-1][j-1],
▶            Math.min(D[i][j-1], D[i-1][j]));
15   return D[m][n];
16 }
```

	ε	a	r	k
ε	0	1	2	3
k	1	1	2	2
a	2	1	2	3
r	3			

'a' ≠ 'k' → 1 + min(değiştir 2, ekle 2, sil 2) = **3**.

Adım 7 / 11

DEĞİŞKENLER

`m = 3` `n = 3` `i = 2` `j = 3`

levenshtein("kar", "ark")

```
1  int levenshtein(String a, String b) {
2    int m = a.length(), n = b.length();
3    int[][] D = new int[m + 1][n + 1];
4    for (int i = 0; i <= m; i++)
5      D[i][0] = i;            // silme
6    for (int j = 0; j <= n; j++)
7      D[0][j] = j;            // ekleme
8    for (int i = 1; i <= m; i++)
9      for (int j = 1; j <= n; j++)
▶      if (a.charAt(i-1) == b.charAt(j-1))
11       D[i][j] = D[i-1][j-1];  // eşit
12     else
▶        D[i][j] = 1 + Math.min(D[i-1][j-1],
▶            Math.min(D[i][j-1], D[i-1][j]));
15   return D[m][n];
16 }
```

	ε	a	r	k
ε	0	1	2	3
k	1	1	2	2
a	2	1	2	3
r	3	2		

'r' ≠ 'a' → 1 + min(değiştir 2, ekle 3, sil 1) = **2**.

Adım 8 / 11

DEĞİŞKENLER

m = 3 | n = 3 | i = 3 | j = 1

levenshtein("kar", "ark")

```
1  int levenshtein(String a, String b) {
2    int m = a.length(), n = b.length();
3    int[][] D = new int[m + 1][n + 1];
4    for (int i = 0; i <= m; i++)
5      D[i][0] = i;              // silme
6    for (int j = 0; j <= n; j++)
7      D[0][j] = j;              // ekleme
8    for (int i = 1; i <= m; i++)
9      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
12     else
13       D[i][j] = 1 + Math.min(D[i-1][j-1],
14           Math.min(D[i][j-1], D[i-1][j]));
15   return D[m][n];
16 }
```

	ε	a	r	k
ε	0	1	2	3
k	1	1	2	2
a	2	1	2	3
r	3	2	1	

a[2] = b[1] = 'r' → maliyetsiz: D[3][2] = sol üst = **1**.

DEĞİŞKENLER

`m = 3` `n = 3` `i = 3` `j = 2`

levenshtein("kar", "ark")

```
1  int levenshtein(String a, String b) {
2    int m = a.length(), n = b.length();
3    int[][] D = new int[m + 1][n + 1];
4    for (int i = 0; i <= m; i++)
5      D[i][0] = i;              // silme
6    for (int j = 0; j <= n; j++)
7      D[0][j] = j;              // ekleme
8    for (int i = 1; i <= m; i++)
9      for (int j = 1; j <= n; j++)
▶        if (a.charAt(i-1) == b.charAt(j-1))
11         D[i][j] = D[i-1][j-1];  // eşit
12       else
▶          D[i][j] = 1 + Math.min(D[i-1][j-1],
▶              Math.min(D[i][j-1], D[i-1][j]));
15   return D[m][n];
16 }
```

	ε	a	r	k
ε	0	1	2	3
k	1	1	2	2
a	2	1	2	3
r	3	2	1	2

'r' ≠ 'k' → 1 + min(değiştir 2, ekle 1, sil 3) = **2**.

Adım 10 / 11

DEĞİŞKENLER

m = 3 | n = 3 | i = 3 | j = 3

levenshtein("kar", "ark")

```
1  int levenshtein(String a, String b) {
2    int m = a.length(), n = b.length();
3    int[][] D = new int[m + 1][n + 1];
4    for (int i = 0; i <= m; i++)
5      D[i][0] = i;            // silme
6    for (int j = 0; j <= n; j++)
7      D[0][j] = j;            // ekleme
8    for (int i = 1; i <= m; i++)
9      for (int j = 1; j <= n; j++)
10       if (a.charAt(i-1) == b.charAt(j-1))
11         D[i][j] = D[i-1][j-1];  // eşit
12       else
13         D[i][j] = 1 + Math.min(D[i-1][j-1],
14             Math.min(D[i][j-1], D[i-1][j]));
▶    return D[m][n];
16 }
```

	ε	a	r	k
ε	0	1	2	3
k	1	1	2	2
a	2	1	2	3
r	3	2	1	2

D[3][3] = **2**: "kar"ın başındaki k silinir (ar), sona k eklenir (ark). Yeşil yol bu iki düzenlemeyi gösterir.

Adım 11 / 11

DEĞİŞKENLER

`m = 3` `n = 3` `i = 4` `j = 4`

DÜZENLEME UZAKLIĞI

Benzerlik Ölçülerinin Karşılaştırılması

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

- LCS, göreli sırası korunan ama ardışık olması gerekmeyen en uzun ortak alt dizidir; Levenshtein'a benzer bir DP tablosuyla (eşleşmede çapraz + 1, değilse max(üst, sol)) bulunur.
- Yalnızca ekleme ve silmeye izin verilirse uzaklık = **m + n − 2 · 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 ≈ 0,57.

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]. Yalnız n tam sayı tutar: sonek trie'sinin $O(n^2)$ düğümüne karşı O(n) bellek. Kullanım: dizgi arama, sıralama, genetik dizilim analizi, BWT ve veri sıkıştırma.

Sonek Dizisi Oluşturma Yöntemleri

Yöntem	Fikir	Zaman
Kaba kuvvet	Tüm sonekler oluşturulur, `Arrays.sort` ile sıralanır; 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 → 0, b → 1, n → 2. Eşit sıralar henüz ayrışmamıştır.

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 ≥ 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 ≥ n ise −1. Harf karşılaştırılmaz, yalnızca sayı çiftleri sıralanır. Tüm sıralar farklı → 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 × 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 → "an", **3 ve 1** konumlarında geçer.

ara("banana", sa, "an")

```
1  int[] ara(String t, int[] sa, String p) {
▶    int lo = 0, hi = sa.length;
3    while (lo < hi) {   // ilk >= p
4      int mid = (lo + hi) / 2;
5      if (t.substring(sa[mid])
6            .compareTo(p) < 0) lo = mid + 1;
7      else hi = mid;
8    }
9    int bas = lo;
10   hi = sa.length;
11   while (lo < hi) {   // p ile başlamayan
12     int mid = (lo + hi) / 2;
13     if (t.startsWith(p, sa[mid]))
14       lo = mid + 1;
15     else hi = mid;
16   }
17   return new int[] { bas, lo };
18 }
```



t = "banana", SA = [5, 3, 1, 0, 4, 2], p = "an". "an" ile başlayan sonekler sıralı dizide bitişik bir aralıktır; iki ikili arama bu aralığın uçlarını bulur.

Adım 1 / 8

DEĞİŞKENLER

`lo = 0` `hi = 6` `mid = –` `bas = –`

ara("banana", sa, "an")

```
1  int[] ara(String t, int[] sa, String p) {
2    int lo = 0, hi = sa.length;
3    while (lo < hi) {   // ilk >= p
▶      int mid = (lo + hi) / 2;
▶      if (t.substring(sa[mid])
6            .compareTo(p) < 0) lo = mid + 1;
▶      else hi = mid;
8    }
9    int bas = lo;
10   hi = sa.length;
11   while (lo < hi) {   // p ile başlamayan
12     int mid = (lo + hi) / 2;
13     if (t.startsWith(p, sa[mid]))
14       lo = mid + 1;
15     else hi = mid;
16   }
17   return new int[] { bas, lo };
18 }
```

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

mid = 3: "banana" ≥ "an" → hi = mid. Aralık [0, 3).

Adım 2 / 8

DEĞİŞKENLER

`lo = 0` `hi = 3` `mid = 3` `bas = –`

ara("banana", sa, "an")

```
1  int[] ara(String t, int[] sa, String p) {
2    int lo = 0, hi = sa.length;
3    while (lo < hi) {   // ilk >= p
▶      int mid = (lo + hi) / 2;
▶      if (t.substring(sa[mid])
6            .compareTo(p) < 0) lo = mid + 1;
▶      else hi = mid;
8    }
9    int bas = lo;
10   hi = sa.length;
11   while (lo < hi) {   // p ile başlamayan
12     int mid = (lo + hi) / 2;
13     if (t.startsWith(p, sa[mid]))
14       lo = mid + 1;
15     else hi = mid;
16   }
17   return new int[] { bas, lo };
18 }
```

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

mid = 1: "ana" ≥ "an" → hi = mid. Aralık [0, 1).

Adım 3 / 8

DEĞİŞKENLER

`lo = 0` `hi = 1` `mid = 1` `bas = –`

ara("banana", sa, "an")

```
1  int[] ara(String t, int[] sa, String p) {
2    int lo = 0, hi = sa.length;
3    while (lo < hi) {   // ilk >= p
▶      int mid = (lo + hi) / 2;
▶      if (t.substring(sa[mid])
▶            .compareTo(p) < 0) lo = mid + 1;
7      else hi = mid;
8    }
9    int bas = lo;
10   hi = sa.length;
11   while (lo < hi) {   // p ile başlamayan
12     int mid = (lo + hi) / 2;
13     if (t.startsWith(p, sa[mid]))
14       lo = mid + 1;
15     else hi = mid;
16   }
17   return new int[] { bas, lo };
18 }
```

lo=hi →

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

mid = 0: "a" < "an" → lo = mid + 1. Aralık [1, 1).

Adım 4 / 8

DEĞİŞKENLER

`lo = 1` `hi = 1` `mid = 0` `bas = –`

ara("banana", sa, "an")

```
1  int[] ara(String t, int[] sa, String p) {
2    int lo = 0, hi = sa.length;
3    while (lo < hi) {   // ilk >= p
4      int mid = (lo + hi) / 2;
5      if (t.substring(sa[mid])
6            .compareTo(p) < 0) lo = mid + 1;
7      else hi = mid;
8    }
▶    int bas = lo;
▶    hi = sa.length;
11   while (lo < hi) {   // p ile başlamayan
12     int mid = (lo + hi) / 2;
13     if (t.startsWith(p, sa[mid]))
14       lo = mid + 1;
15     else hi = mid;
16   }
17   return new int[] { bas, lo };
18 }
```

	sıra	SA	sonek
	0	5	a
lo →	1	3	ana
	2	1	anana
	3	0	banana
	4	4	na
	5	2	nana
hi →			(dizi sonu)

İlk döngü bitti: "an"dan küçük olmayan ilk sonek sıra 1'de → **bas = 1**. İkinci arama için hi yeniden dizi sonuna çekilir.

Adım 5 / 8

DEĞİŞKENLER

`lo = 1` `hi = 6` `mid = –` `bas = 1`

ara("banana", sa, "an")

```
1  int[] ara(String t, int[] sa, String p) {
2    int lo = 0, hi = sa.length;
3    while (lo < hi) {   // ilk >= p
4      int mid = (lo + hi) / 2;
5      if (t.substring(sa[mid])
6            .compareTo(p) < 0) lo = mid + 1;
7      else hi = mid;
8    }
9    int bas = lo;
10   hi = sa.length;
11   while (lo < hi) {   // p ile başlamayan
▶      int mid = (lo + hi) / 2;
▶      if (t.startsWith(p, sa[mid]))
14       lo = mid + 1;
▶      else hi = mid;
16   }
17   return new int[] { bas, lo };
18 }
```

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

mid = 3: "banana", "an" ile başlamaz → hi = mid.

Adım 6 / 8

DEĞİŞKENLER

`lo = 1` `hi = 3` `mid = 3` `bas = 1`

ara("banana", sa, "an")

```
1  int[] ara(String t, int[] sa, String p) {
2    int lo = 0, hi = sa.length;
3    while (lo < hi) {   // ilk >= p
4      int mid = (lo + hi) / 2;
5      if (t.substring(sa[mid])
6            .compareTo(p) < 0) lo = mid + 1;
7      else hi = mid;
8    }
9    int bas = lo;
10   hi = sa.length;
11   while (lo < hi) {   // p ile başlamayan
▶      int mid = (lo + hi) / 2;
▶      if (t.startsWith(p, sa[mid]))
▶        lo = mid + 1;
15     else hi = mid;
16   }
17   return new int[] { bas, lo };
18 }
```

lo=hi →

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

mid = 2: "anana", "an" ile başlar → lo = mid + 1.

Adım 7 / 8

DEĞİŞKENLER

`lo = 3` `hi = 3` `mid = 2` `bas = 1`

ara("banana", sa, "an")

```
1  int[] ara(String t, int[] sa, String p) {
2    int lo = 0, hi = sa.length;
3    while (lo < hi) {   // ilk >= p
4      int mid = (lo + hi) / 2;
5      if (t.substring(sa[mid])
6            .compareTo(p) < 0) lo = mid + 1;
7      else hi = mid;
8    }
9    int bas = lo;
10   hi = sa.length;
11   while (lo < hi) {   // p ile başlamayan
12     int mid = (lo + hi) / 2;
13     if (t.startsWith(p, sa[mid]))
14       lo = mid + 1;
15     else hi = mid;
16   }
▶    return new int[] { bas, lo };
18 }
```

lo=hi →

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

Sonuç [1, 3): sıra 1 ve 2 → SA değerleri **3 ve 1**. "an", banana içinde 3 ve 1 konumlarında geçer; toplam 5 karşılaştırma yetti (her ikili arama en çok $\lfloor \log_2 6 \rfloor + 1 = 3$).

Adım 8 / 8

DEĞİŞKENLER

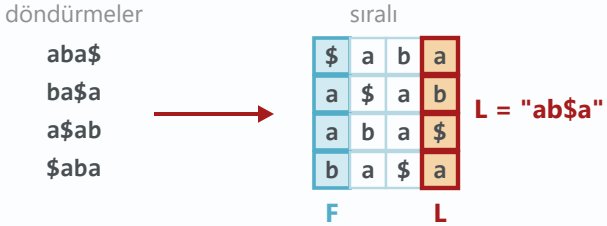
`lo = 3` `hi = 3` `mid = –` `bas = 1`

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 hâlleri** (cyclic rotations) oluşturulur.
2. Döndürmeler **alfabetik** olarak sıralanır.
3. Sıralı döndürmelerin **son karakterlerinden** yeni bir dizgi oluşturulur (L sütunu).
4. Orijinal dizginin sıralı listedeki **satır numarası** saklanır (ya da sona tekil bir `$` eklenir).



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

BWT Örneği: BANANA

Girdi: **BANANA**

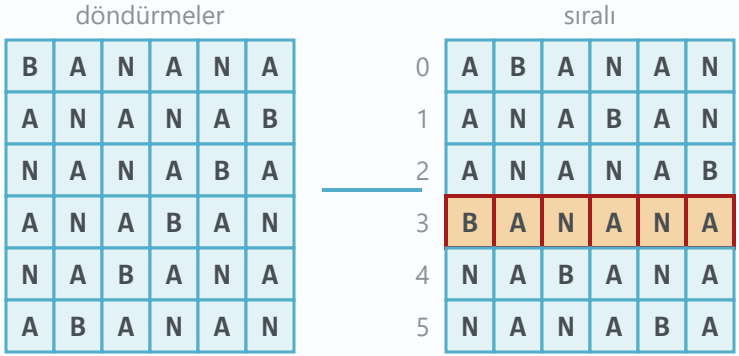
döndürmeler

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

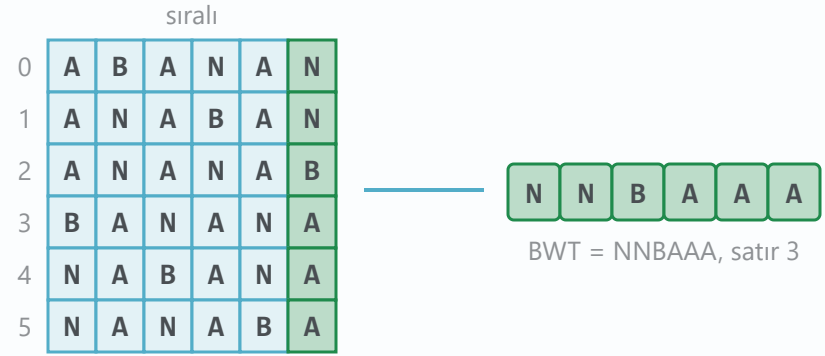
Girdi: **BANANA**



2. Alfabetik sıralama: ABANAN, ANABAN, ANANAB, BANANA, NABANA, NANABA. Orijinal dizgi **3 numaralı satırda** (0'dan sayarak).

BWT Örneği: BANANA

Girdi: **BANANA**



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

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	← L
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	

- 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 sonek dizisinden **L[i] = T[SA[i] − 1]** ile O(n)'de elde edilir (T sonuna `$` eklenmiş metin).

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 = banana**$** ($ en küçük, tekil) → BWT = **annb$aa**.
- **F** = sırala(L) = $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 başlanır, L[i] okunur, LF(i) satırına gidilir:

satırlar 0 → 1 → 5 → 2 → 6 → 3 → 4
okunan L: a n a n a b → ters çevir → **banana**

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

bwtTers("annb$aa")

```
1  String bwtTers(String L) {   // $ ile
2    int n = L.length();
3    int[] sira = new int[n];
4    int[] say = new int[256];
5    int[] C = new int[256];
▶    for (int i = 0; i < n; i++)
▶      sira[i] = say[L.charAt(i)]++;
8    for (int c = 1; c < 256; c++)
9      C[c] = C[c - 1] + say[c - 1];
10   StringBuilder s = new StringBuilder();
11   int i = 0;                    // $ satırı
12   for (int k = 0; k < n - 1; k++) {
13     char c = L.charAt(i);
14     s.insert(0, c);
15     i = C[c] + sira[i];         // LF(i)
16   }
17   return s.toString();
18 }
```

i	L	sıra	F
0	a	0	$
1	n	0	a
2	n	1	a
3	b	0	a
4	$	0	b
5	a	1	n
6	a	2	n

	$	a	b	n
C[c]				

C[c]: c'den küçük karakter sayısı

s

L = "annb$aa" (banana$'ın BWT'si). sira[i]: L[i] karakterinin L'de kaçıncı kez geçtiği (0'dan). Örn. L[6] = a, üçüncü a → 2.

Adım 1 / 9

DEĞİŞKENLER

n = 7 | k = – | i = – | c = –

bwtTers("annb$aa")

```
1  String bwtTers(String L) {   // $ ile
2    int n = L.length();
3    int[] sira = new int[n];
4    int[] say = new int[256];
5    int[] C = new int[256];
6    for (int i = 0; i < n; i++)
7      sira[i] = say[L.charAt(i)]++;
▶    for (int c = 1; c < 256; c++)
▶      C[c] = C[c - 1] + say[c - 1];
10   StringBuilder s = new StringBuilder();
11   int i = 0;                    // $ satırı
12   for (int k = 0; k < n - 1; k++) {
13     char c = L.charAt(i);
14     s.insert(0, c);
15     i = C[c] + sira[i];         // LF(i)
16   }
17   return s.toString();
18 }
```

i	L	sıra	F
0	a	0	$
1	n	0	a
2	n	1	a
3	b	0	a
4	$	0	b
5	a	1	n
6	a	2	n

C[c]	$	a	b	n
	0	1	4	5

C[c]: c'den küçük karakter sayısı

s

C[c] = c'den küçük karakter sayısı = c'nin F sütununda başladığı satır: $ → 0, a → 1, b → 4, n → 5.

Adım 2 / 9

DEĞİŞKENLER

n = 7 | k = – | i = – | c = –

bwtTers("annb$aa")

```
1  String bwtTers(String L) {   // $ ile
2    int n = L.length();
3    int[] sira = new int[n];
4    int[] say = new int[256];
5    int[] C = new int[256];
6    for (int i = 0; i < n; i++)
7      sira[i] = say[L.charAt(i)]++;
8    for (int c = 1; c < 256; c++)
9      C[c] = C[c - 1] + say[c - 1];
10   StringBuilder s = new StringBuilder();
11   int i = 0;                    // $ satırı
12   for (int k = 0; k < n - 1; k++) {
▶      char c = L.charAt(i);
▶      s.insert(0, c);
▶      i = C[c] + sira[i];         // LF(i)
16   }
17   return s.toString();
18 }
```

i	L	sıra	F
0	a	0	$
1	n	0	a
2	n	1	a
3	b	0	a
4	$	0	b
5	a	1	n
6	a	2	n

C[c]	$	a	b	n
	0	1	4	5

C[c]: c'den küçük karakter sayısı

s: [ ][ ][ ][ ][ ][a]

i = 0: L[0] = 'a' metinde öncekine eklenir → s = "a".
LF(0) = C['a'] + sira[0] = 1 + 0 = **1**.

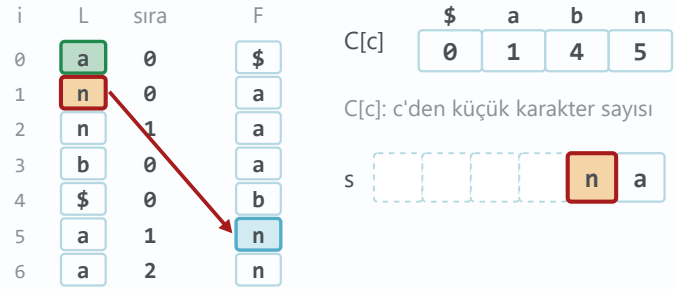
Adım 3 / 9

DEĞİŞKENLER

`n = 7` `k = 0` `i = 1` `c = 'a'`

bwtTers("annb$aa")

```
1  String bwtTers(String L) {   // $ ile
2    int n = L.length();
3    int[] sira = new int[n];
4    int[] say = new int[256];
5    int[] C = new int[256];
6    for (int i = 0; i < n; i++)
7      sira[i] = say[L.charAt(i)]++;
8    for (int c = 1; c < 256; c++)
9      C[c] = C[c - 1] + say[c - 1];
10   StringBuilder s = new StringBuilder();
11   int i = 0;                    // $ satırı
12   for (int k = 0; k < n - 1; k++) {
▶      char c = L.charAt(i);
▶      s.insert(0, c);
▶      i = C[c] + sira[i];         // LF(i)
16   }
17   return s.toString();
18 }
```



i = 1: L[1] = 'n' metinde öncekine eklenir → s = "na".
LF(1) = C['n'] + sira[1] = 5 + 0 = **5**.

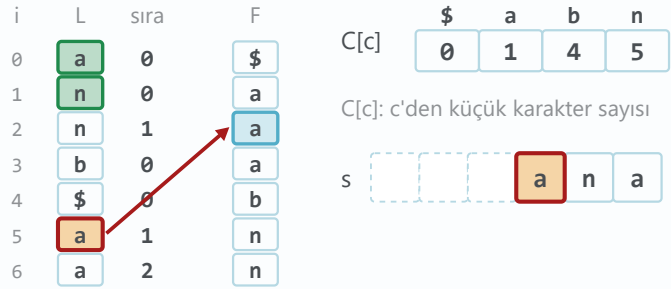
Adım 4 / 9

DEĞİŞKENLER

`n = 7` `k = 1` `i = 5` `c = 'n'`

bwtTers("annb$aa")

```
1  String bwtTers(String L) {   // $ ile
2    int n = L.length();
3    int[] sira = new int[n];
4    int[] say = new int[256];
5    int[] C = new int[256];
6    for (int i = 0; i < n; i++)
7      sira[i] = say[L.charAt(i)]++;
8    for (int c = 1; c < 256; c++)
9      C[c] = C[c - 1] + say[c - 1];
10   StringBuilder s = new StringBuilder();
11   int i = 0;                    // $ satırı
12   for (int k = 0; k < n - 1; k++) {
▶      char c = L.charAt(i);
▶      s.insert(0, c);
▶      i = C[c] + sira[i];         // LF(i)
16   }
17   return s.toString();
18 }
```



i = 5: L[5] = 'a' metinde öncekine eklenir → s = "ana". LF(5) = C['a'] + sira[5] = 1 + 1 = **2**.

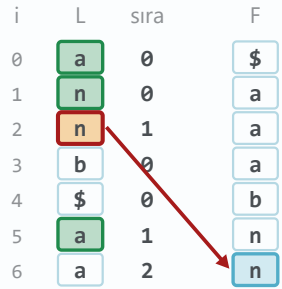
Adım 5 / 9

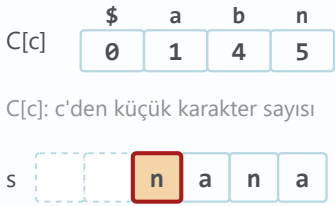
DEĞİŞKENLER

`n = 7` `k = 2` `i = 2` `c = 'a'`

bwtTers("annb$aa")

```
1  String bwtTers(String L) {   // $ ile
2    int n = L.length();
3    int[] sira = new int[n];
4    int[] say = new int[256];
5    int[] C = new int[256];
6    for (int i = 0; i < n; i++)
7      sira[i] = say[L.charAt(i)]++;
8    for (int c = 1; c < 256; c++)
9      C[c] = C[c - 1] + say[c - 1];
10   StringBuilder s = new StringBuilder();
11   int i = 0;                    // $ satırı
12   for (int k = 0; k < n - 1; k++) {
▶      char c = L.charAt(i);
▶      s.insert(0, c);
▶      i = C[c] + sira[i];         // LF(i)
16   }
17   return s.toString();
18 }
```





i = 2: L[2] = 'n' metinde öncekine eklenir → s = "nana". LF(2) = C['n'] + sira[2] = 5 + 1 = **6**.

Adım 6 / 9

DEĞİŞKENLER

`n = 7` `k = 3` `i = 6` `c = 'n'`

bwtTers("annb$aa")

```
1  String bwtTers(String L) {   // $ ile
2    int n = L.length();
3    int[] sira = new int[n];
4    int[] say = new int[256];
5    int[] C = new int[256];
6    for (int i = 0; i < n; i++)
7      sira[i] = say[L.charAt(i)]++;
8    for (int c = 1; c < 256; c++)
9      C[c] = C[c - 1] + say[c - 1];
10   StringBuilder s = new StringBuilder();
11   int i = 0;                    // $ satırı
12   for (int k = 0; k < n - 1; k++) {
▶      char c = L.charAt(i);
▶      s.insert(0, c);
▶      i = C[c] + sira[i];         // LF(i)
16   }
17   return s.toString();
18 }
```

i	L	sıra	F
0	a	0	$
1	n	0	a
2	n	1	a
3	b	0	a
4	$	0	b
5	a	1	n
6	a	2	n

C[c]	$	a	b	n
	0	1	4	5

C[c]: c'den küçük karakter sayısı

s		a	n	a	n	a

i = 6: L[6] = 'a' metinde öncekine eklenir → s = "anana". LF(6) = C['a'] + sira[6] = 1 + 2 = **3**.

Adım 7 / 9

DEĞİŞKENLER

`n = 7` `k = 4` `i = 3` `c = 'a'`

bwtTers("annb$aa")

```
1  String bwtTers(String L) {   // $ ile
2    int n = L.length();
3    int[] sira = new int[n];
4    int[] say = new int[256];
5    int[] C = new int[256];
6    for (int i = 0; i < n; i++)
7      sira[i] = say[L.charAt(i)]++;
8    for (int c = 1; c < 256; c++)
9      C[c] = C[c - 1] + say[c - 1];
10   StringBuilder s = new StringBuilder();
11   int i = 0;                    // $ satırı
12   for (int k = 0; k < n - 1; k++) {
▶      char c = L.charAt(i);
▶      s.insert(0, c);
▶      i = C[c] + sira[i];         // LF(i)
16   }
17   return s.toString();
18 }
```

i	L	sıra	F
0	a	0	$
1	n	0	a
2	n	1	a
3	b	0	a
4	$	0	b
5	a	1	n
6	a	2	n

C[c]	$	a	b	n
	0	1	4	5

C[c]: c'den küçük karakter sayısı

s	b	a	n	a	n	a

i = 3: L[3] = 'b' metinde öncekine eklenir → s = "banana". LF(3) = C['b'] + sira[3] = 4 + 0 = **4**.

Adım 8 / 9

DEĞİŞKENLER

n = 7 | k = 5 | i = 4 | c = 'b'

bwtTers("annb$aa")

```
1  String bwtTers(String L) {   // $ ile
2    int n = L.length();
3    int[] sira = new int[n];
4    int[] say = new int[256];
5    int[] C = new int[256];
6    for (int i = 0; i < n; i++)
7      sira[i] = say[L.charAt(i)]++;
8    for (int c = 1; c < 256; c++)
9      C[c] = C[c - 1] + say[c - 1];
10   StringBuilder s = new StringBuilder();
11   int i = 0;                    // $ satırı
12   for (int k = 0; k < n - 1; k++) {
13     char c = L.charAt(i);
14     s.insert(0, c);
15     i = C[c] + sira[i];         // LF(i)
16   }
▶    return s.toString();
18 }
```

i	L	sıra	F
0	a	0	$
1	n	0	a
2	n	1	a
3	b	0	a
4	$	0	b
5	a	1	n
6	a	2	n

C[c]	$	a	b	n
	0	1	4	5

C[c]: c'den küçük karakter sayısı

s	b	a	n	a	n	a

Satırlar 0 → 1 → 5 → 2 → 6 → 3 → 4 izlendi; sonraki satır $ ile biteceği için döngü n − 1 = 6 turda durur. Sonuç: **banana**.

Adım 9 / 9

DEĞİŞKENLER

`n = 7` `k = 6` `i = 4` `c = 'b'`

Özet

Konu	Hatırlanacaklar
Sözlüksel sıra	ilk farklı karakter karar verir; önek önce gelir; kod sırası ≠ 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 ≡ DFA ≡ düzenli ifade
Levenshtein	$1 + \min$(sil, ekle, değiştir), eşitse sol üst; $O(m \cdot n)$
Benzerlik	Hamming eşit uzunlukta $O(n)$; LCS ve ortak alt dizgi $O(m \cdot n)$; yalnız ekle/sil uzaklığı $= m + n - 2 \cdot LCS$
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

ÖZET

Etkileşimli Simülatörler

Dizgi Sıralama: LSD Basamak Sıralaması

Virgülle ayrılmış dizgiler sağdan sola, karakter karakter kovalara dağıtılarak sıralanır; her geçişte kovalar ve dizi gösterilir.

Dizgi Ayrıştırma: FSM

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

Dizgi Benzerliği: Levenshtein

İki dizgi için (ör. kitten · sitting) DP tablosu hücre hücre dolar; ekleme, silme, değiştirme maliyetleri gösterilir.

Dizgi Dönüşümü: BWT

Tüm döndürmeler üretilir, sıralanır ve son sütun okunarak Burrows–Wheeler dönüşümü adım adım elde edilir.

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 çevirmek ya da sıfırla aynı uzunluğa getirmek gerekir.

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. Doğrusu: `\.` ve tembel `.*?`. `(a+)+$` gibi ifadeler üstel sayıda geri dönüşe yol açar.

DP tablosunda indis kayması

Tablo (m+1) × (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. sob, bat, cab, bot, tab dizgilerine LSD basamak sıralaması uygulayın; her geçişten sonraki diziyi yazın.
3. `^\d{3}-\d{4}$` ifadesi "555-1234", "5551234", "55-12345" dizgilerinden hangileriyle eşleşir?
4. Levenshtein("sunday", "saturday") kaçtır?
5. Hamming("karolin", "kathrin") kaçtır? Levenshtein uzaklığı bundan büyük olabilir mi?
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 (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 5.2: İleri Dizgi Algoritmaları

Sonraki bölüm: Bölüm 5.3: Trie Veri Yapısı · Terim sözlüğü