Bölüm 9: Dizgi Eşleştirme ve Sıkıştırma

Algoritmalar

Kaba kuvvet | KMP ve LPS tablosu | Boyer-Moore | Rabin-Karp | Karşılaştırma | RLE | LZW

Sercan KÜLCÜ · Giresun Üniversitesi · Bilgisayar Mühendisliği

© Sercan Külcü. Tüm hakları saklıdır. · Sunumun etkileşimli HTML sürümü:
https://sercankulcu.github.io/files/algorithms/slides/Bolum_09_Dizgi_Eslestirme_ve_Sikistirma.html

GİRİŞ

Öğrenme Çıktıları

Kavramlar

Metin, örüntü, kayma, önek ve sonek kavramlarını tanımlamak; dizgi eşleştirme problemini biçimsel olarak ifade etmek.

Eşleştirme

Kaba kuvvet, KMP, Boyer-Moore ve Rabin-Karp algoritmalarını bir örnek üzerinde adım adım çalıştırmak ve Java ile yazmak.

Analiz

LPS tablosu, kötü karakter tablosu ve kayan hash'in zaman kazancını açıklamak; en iyi, ortalama ve en kötü durumları karşılaştırmak.

Sıkıştırma

RLE ve LZW ile kayıpsız sıkıştırma yapmak, sıkıştırılmış veriyi geri açmak ve sıkıştırma oranını hesaplamak.

GİRİŞ

Dizgi Algoritmaları

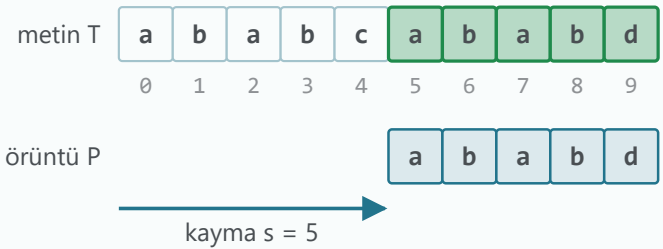
- Metinlerle dolu bir dünyada yaşıyoruz: **e-postalar, iletiler, sosyal medya paylaşımları, haber metinleri**...
- Bilgisayarlarımızda her gün sayısız metinle karşılaşıyoruz. Peki bu metinler **nasıl düzenlenir ve analiz edilir?**
- **Dizgi (string) algoritmaları** metinlerde **arama**, **değiştirme** ve **karşılaştırma** gibi işlemleri gerçekleştirir.
- İki temel konu: **dizgi eşleştirme** (bir örüntüyü metinde bulmak) ve **dizgi sıkıştırma** (metni daha az yerde saklamak).

Nerelerde?

Editörlerde Ctrl+F, `grep`, arama motorları, virüs imzası tarama, intihal tespiti, DNA dizilerinde gen arama, ZIP/GIF/PNG gibi dosya biçimleri.

Temel Kavramlar ve Gösterim

- **Alfabe** Σ: kullanılan karakterler kümesi (ör. {a, b, …, z}, {A, C, G, T}).
- **Metin** (text) T[0..n−1] ve **örüntü** (pattern) P[0..m−1], genellikle m ≤ n.
- P, T içinde **s kaymasında** (shift) geçer ⇔ T[s..s+m−1] = P. Geçerli kaymalar 0 ≤ s ≤ n − m.
- **Önek** (prefix): P[0..k−1]; **sonek** (suffix): P[m−k..m−1]. **Uygun** (proper) önek/sonek, dizginin kendisine eşit olmayandır.
- Örnek: "ababd" için önekler a, ab, aba, abab; sonekler d, bd, abd, babd.



Dizgi eşleştirme problemi: P'nin geçtiği tüm (ya da ilk) s değerlerini bulmak.

GİRİŞ

Bölümün Haritası

Kaba Kuvvet (Brute Force)
Metindeki her konum, örüntüyle eşleşip eşleşmediği için kontrol edilir. En fazla sayıda karşılaştırma gerektirebilir.

Knuth-Morris-Pratt (KMP)
Başlangıçta bir tablo (LPS) oluşturur; metinde hiç geri gitmeden, karşılaştırma sayısını azaltarak hızlı çalışır.

Boyer-Moore
Sağdan sola karşılaştırır; kötü karakter ve iyi sonek kurallarıyla uzun atlamalar yapar. Uzun örüntülerde çok etkili.

Rabin-Karp
Hash (özet) değerlerini karşılaştırır; pencereyi kaydırırken hash'i O(1)'de günceller. Olasılıksal hızlıdır.

RLE (Run Length Encoding)
Çalışma uzunluğu kodlaması: ardışık tekrar eden değerler tek değer + tekrar sayısı olarak saklanır.

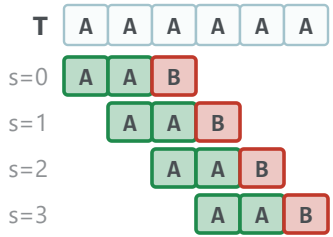
LZW (Lempel-Ziv-Welch)
Sözlük tabanlı sıkıştırma: tekrar eden örüntüler dinamik olarak büyüyen bir sözlükteki kısa kodlarla temsil edilir (GIF).

Kaba Kuvvet (Naive) Algoritması

- Metin içinde belirli bir örüntüyü arayan **en basit** algoritmadır; "naive" (saf) adı bu yüzdendir.
- **İşleyiş:**
  1. Her s = 0, 1, …, n − m kayması için örüntü metnin altına hizalanır.
  2. Karakterler **soldan sağa** karşılaştırılır; ilk uyuşmazlıkta durulur.
  3. m karakterin hepsi eşleşirse s **rapor edilir**; aksi hâlde örüntü **1 konum** kaydırılır.
- Ön işleme ve ek bellek **gerektirmez**.

Karmaşıklık

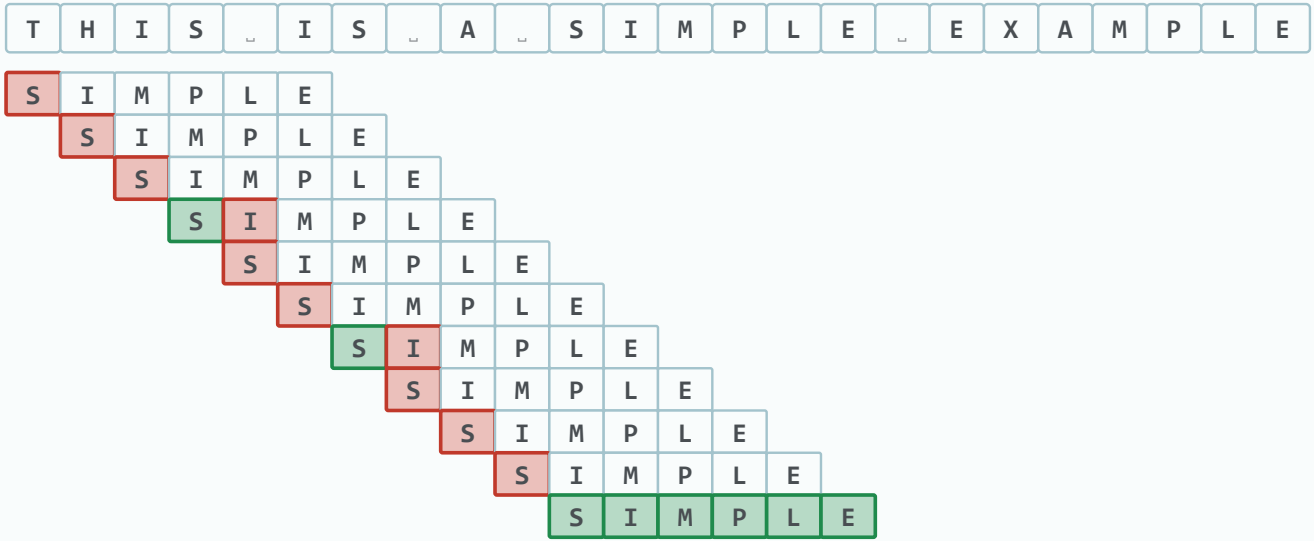
En kötü durum: (n − m + 1) · m karşılaştırma → **O(n · m)**.
En iyi durum: her hizada ilk karakter uyuşmaz → O(n).
n: metin uzunluğu, m: örüntü uzunluğu.



4 hiza × 3 = 12 karşılaştırma

KABA KUVVET

Örnek: "THIS IS A SIMPLE EXAMPLE"



Her satır bir kayma: yeşil eşleşen, kırmızı uyuşmayan karşılaştırma. SIMPLE, 11. hizada (s = 10) bulunur; toplam **18** karşılaştırma yapılır (n = 24, m = 6).

"HELLO WORLD" İçinde "LO": Kod İzleme

```
1  int kabaKuvvet(String t, String p) {
▶    int n = t.length(), m = p.length();
3    for (int i = 0; i <= n - m; i++) {
4      int j = 0;
5      while (j < m &&
6        t.charAt(i + j) == p.charAt(j))
7        j++;
8      if (j == m) return i;
9    }
10   return -1;
11 }
```

t	H	E	L	L	O	␣	W	O	R	L	D
	0	1	2	3	4	5	6	7	8	9	10

n = 11, m = 2. Örüntü i = 0, 1, ..., 9 kaymalarında denenecek.

Adım 1 / 8

DEĞİŞKENLER

`n = 11` `m = 2` `i = –` `j = –` `karş. = 0`

"HELLO WORLD" İçinde "LO": Kod İzleme

```
1  int kabaKuvvet(String t, String p) {
2    int n = t.length(), m = p.length();
3    for (int i = 0; i <= n - m; i++) {
4      int j = 0;
▶      while (j < m &&
▶        t.charAt(i + j) == p.charAt(j))
7        j++;
8      if (j == m) return i;
9    }
10   return -1;
11 }
```



t[0] = 'H' ≠ p[0] = 'L' → uyuşmazlık; örüntü 1 sağa kayar.

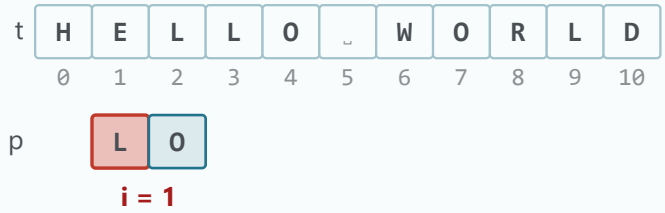
Adım 2 / 8

DEĞİŞKENLER

`n = 11` `m = 2` `i = 0` `j = 0` `karş. = 1`

"HELLO WORLD" İçinde "LO": Kod İzleme

```
1  int kabaKuvvet(String t, String p) {
2    int n = t.length(), m = p.length();
3    for (int i = 0; i <= n - m; i++) {
4      int j = 0;
▶      while (j < m &&
▶        t.charAt(i + j) == p.charAt(j))
7        j++;
8      if (j == m) return i;
9    }
10   return -1;
11 }
```



t[1] = 'E' ≠ p[0] = 'L' → uyuşmazlık; örüntü 1 sağa kayar.

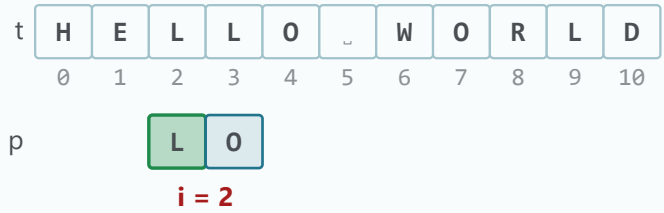
Adım 3 / 8

DEĞİŞKENLER

`n = 11` `m = 2` `i = 1` `j = 0` `karş. = 2`

"HELLO WORLD" İçinde "LO": Kod İzleme

```
1  int kabaKuvvet(String t, String p) {
2    int n = t.length(), m = p.length();
3    for (int i = 0; i <= n - m; i++) {
4      int j = 0;
▶      while (j < m &&
▶        t.charAt(i + j) == p.charAt(j))
▶        j++;
8      if (j == m) return i;
9    }
10   return -1;
11 }
```



t[2] = 'L' ile p[0] = 'L' eşleşti → j++.

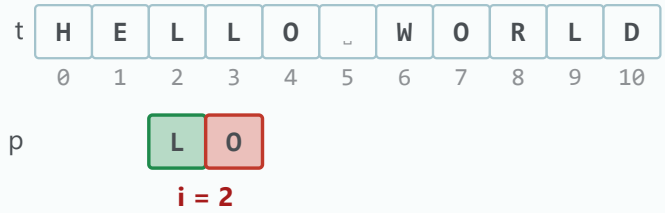
Adım 4 / 8

DEĞİŞKENLER

n = 11 | m = 2 | i = 2 | j = 1 | karş. = 3

"HELLO WORLD" İçinde "LO": Kod İzleme

```
1  int kabaKuvvet(String t, String p) {
2    int n = t.length(), m = p.length();
3    for (int i = 0; i <= n - m; i++) {
4      int j = 0;
▶      while (j < m &&
▶        t.charAt(i + j) == p.charAt(j))
7        j++;
8      if (j == m) return i;
9    }
10   return -1;
11 }
```



t[3] = 'L' ≠ p[1] = 'O' → uyuşmazlık; örüntü 1 sağa kayar.

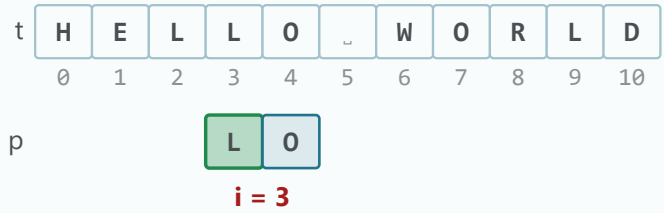
Adım 5 / 8

DEĞİŞKENLER

`n = 11` `m = 2` `i = 2` `j = 1` `karş. = 4`

"HELLO WORLD" İçinde "LO": Kod İzleme

```
1  int kabaKuvvet(String t, String p) {
2    int n = t.length(), m = p.length();
3    for (int i = 0; i <= n - m; i++) {
4      int j = 0;
▶      while (j < m &&
▶        t.charAt(i + j) == p.charAt(j))
▶        j++;
8      if (j == m) return i;
9    }
10   return -1;
11 }
```



t[3] = 'L' ile p[0] = 'L' eşleşti → j++.

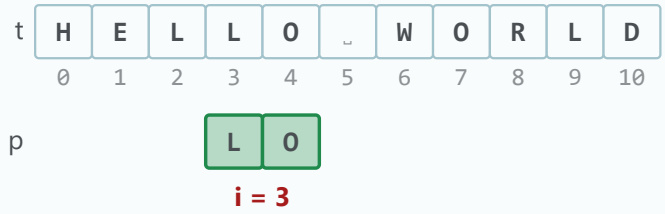
Adım 6 / 8

DEĞİŞKENLER

`n = 11` `m = 2` `i = 3` `j = 1` `karş. = 5`

"HELLO WORLD" İçinde "LO": Kod İzleme

```
1  int kabaKuvvet(String t, String p) {
2    int n = t.length(), m = p.length();
3    for (int i = 0; i <= n - m; i++) {
4      int j = 0;
▶      while (j < m &&
▶        t.charAt(i + j) == p.charAt(j))
▶        j++;
8      if (j == m) return i;
9    }
10   return -1;
11 }
```



t[4] = 'O' ile p[1] = 'O' eşleşti → j++.

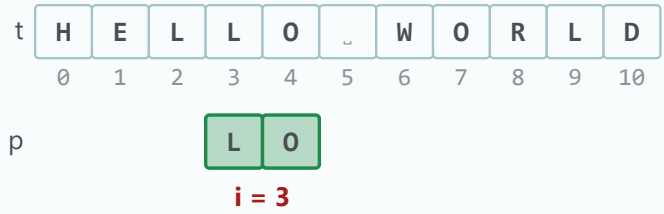
Adım 7 / 8

DEĞİŞKENLER

`n = 11` `m = 2` `i = 3` `j = 2` `karş. = 6`

"HELLO WORLD" İçinde "LO": Kod İzleme

```
1  int kabaKuvvet(String t, String p) {
2    int n = t.length(), m = p.length();
3    for (int i = 0; i <= n - m; i++) {
4      int j = 0;
5      while (j < m &&
6        t.charAt(i + j) == p.charAt(j))
7        j++;
▶      if (j == m) return i;
9    }
10   return -1;
11 }
```



j == m: örüntü **i = 3** konumunda bulundu. Toplam 6 karşılaştırma.

Adım 8 / 8

DEĞİŞKENLER

`n = 11` `m = 2` `i = 3` `j = 2` `karş. = 6`

Kaba Kuvvet: En İyi, Ortalama, En Kötü

Durum	Ne zaman?	Karşılaştırma	Zaman
En iyi	P[0] metnin hiçbir karakteriyle eşleşmez (ör. P = "xyz", T = "aaaa…")	$n - m + 1$	$O(n)$
Ortalama	Rastgele metin, büyük alfabe: uyuşmazlık genellikle ilk 1–2 karakterde	$\approx n \cdot \sigma/(\sigma-1)$	$O(n)$
En kötü	T = "aaaaaaaaaa", P = "aaab": her hizada m karşılaştırma	$(n - m + 1) \cdot m$	$O(n \cdot m)$

Sayısal örnek: n = 10, m = 4, T = a^{10}, P = aaab → 7 hiza × 4 = **28** karşılaştırma. Uyuşmazlık hep son karakterde çıkar.

Asıl kusur: uyuşmazlıkta öğrenilen bilgi (eşleşen karakterler) **atılır**; metinde geri gidilip aynı karakterler tekrar okunur. KMP bu bilgiyi kullanır.

Knuth-Morris-Pratt (KMP) Algoritması

- Metin içinde belirli bir örüntüyü bulmak için kullanılır.
- **Donald Knuth**, **Vaughan Pratt** ve (bağımsız olarak) **James H. Morris** tarafından geliştirilmiş, 1977'de birlikte yayımlanmıştır.
- Temel fikir: uyuşmazlık olduğunda örüntünün **zaten eşleşmiş kısmı** hakkında bildiklerimizi kullanmak; metin işaretçisi **asla geri gitmez**.
- Bu bilgi örüntüden önceden hesaplanan **LPS tablosunda** (önek fonksiyonu, π) saklanır.

Karmaşıklık

Ön işleme: O(m)
Arama: O(n)
Toplam: **O(n + m)**, en kötü durumda da
Ek bellek: O(m)

KMP: İşleyiş

1. Ön işleme

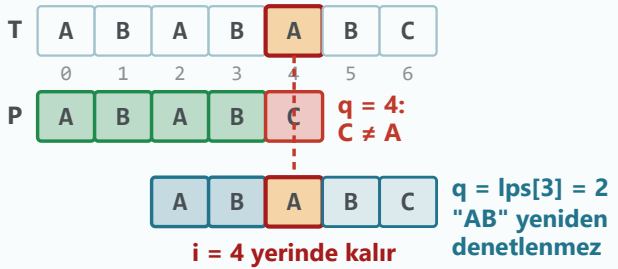
Örüntünün her konumu için **en uzun uygun önek = sonek** (Longest Proper Prefix which is also Suffix, LPS) uzunluğu bir tabloya yazılır. Uyuşmazlıkta örüntüde nereye dönüleceğini bu tablo söyler.

2. Örüntü arama

Metin soldan sağa bir kez taranır. Eşleşmeyen karakterde örüntü işaretçisi q, tablodan okunan değere (q = lps[q − 1]) geri çekilir; metin işaretçisi i yerinde kalır.

3. Eşleşme kontrolü

q, m'ye ulaştığında örüntünün tüm karakterleri eşleşmiştir: konum i − m + 1 rapor edilir. Tüm geçişler isteniyorsa q = lps[m − 1] ile devam edilir.



Neden doğru? P[0..q−1] metinle eşleşmişse ve P[q] uyuşmuyorsa, örüntünün bir sonraki olası hizası, P[0..q−1]'in hem öneki hem soneki olan en uzun parçanın üst üste geleceği hizadır. Arada kalan hizalar **kesinlikle** eşleşemez.

LPS Tablosu (Önek Fonksiyonu)

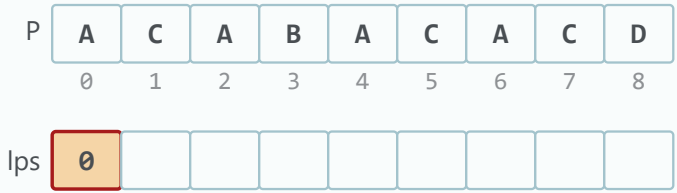
lps[q] = P[0..q] dizgisinin, kendisine eşit olmayan ve aynı zamanda soneki olan **en uzun öneki**nin uzunluğu.

- lps[0] = 0 (tek karakterin uygun öneki yok).
- Örnek: Metin "ababcababcababab", örüntü **"ababd"**.
- Örüntü tablosu: a: 0, b: 0, a: 1, b: 2, d: 0.

q	P[q]	lps[q]	Gerekçe (P[0..q])
0	`a`	**0**	"a" — uygun önek yok
1	`b`	**0**	"ab" — a ≠ b
2	`a`	**1**	"aba" — "a" hem önek hem sonek
3	`b`	**2**	"abab" — "ab" hem önek hem sonek
4	`d`	**0**	"ababd" — d ile biten önek yok

LPS Tablosunu Kurmak: "ACABACACD"

```
1  int[] lpsHesapla(String p) {
2    int m = p.length(), k = 0;
3    int[] lps = new int[m];
4    for (int q = 1; q < m; q++) {
5      while (k > 0 &&
6        p.charAt(k) != p.charAt(q))
7        k = lps[k - 1];
8      if (p.charAt(k) == p.charAt(q))
9        k++;
10     lps[q] = k;
11   }
12   return lps;
13 }
```



lps[0] = 0 her zaman. k = 0: şu an eşleşen önek uzunluğu.

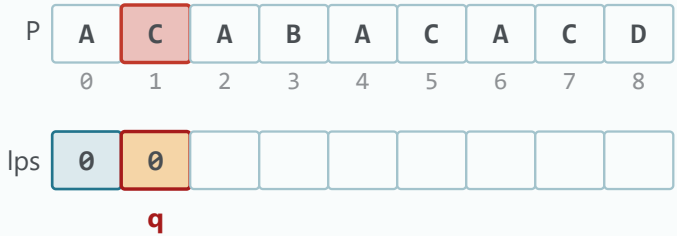
Adım 1 / 13

DEĞİŞKENLER

m = 9 | q = – | k = 0

LPS Tablosunu Kurmak: "ACABACACD"

```
1  int[] lpsHesapla(String p) {
2    int m = p.length(), k = 0;
3    int[] lps = new int[m];
4    for (int q = 1; q < m; q++) {
5      while (k > 0 &&
6        p.charAt(k) != p.charAt(q))
7        k = lps[k - 1];
▶      if (p.charAt(k) == p.charAt(q))
9        k++;
▶      lps[q] = k;
11   }
12   return lps;
13 }
```



p[0] = 'A' ≠ p[1] = 'C' → k = 0, lps[1] = 0.

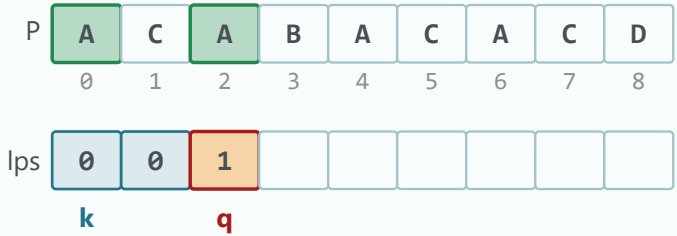
Adım 2 / 13

DEĞİŞKENLER

`m = 9` `q = 1` `k = 0`

LPS Tablosunu Kurmak: "ACABACACD"

```
1  int[] lpsHesapla(String p) {
2    int m = p.length(), k = 0;
3    int[] lps = new int[m];
4    for (int q = 1; q < m; q++) {
5      while (k > 0 &&
6        p.charAt(k) != p.charAt(q))
7        k = lps[k - 1];
▶      if (p.charAt(k) == p.charAt(q))
▶        k++;
▶      lps[q] = k;
11   }
12   return lps;
13 }
```



p[0] = 'A' = p[2] = 'A' → k = 1, lps[2] = 1.

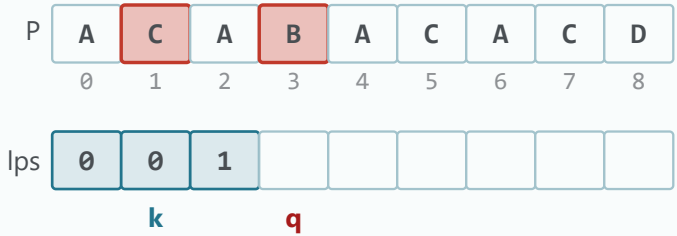
Adım 3 / 13

DEĞİŞKENLER

`m = 9` `q = 2` `k = 1`

LPS Tablosunu Kurmak: "ACABACACD"

```
1  int[] lpsHesapla(String p) {
2    int m = p.length(), k = 0;
3    int[] lps = new int[m];
4    for (int q = 1; q < m; q++) {
▶      while (k > 0 &&
▶        p.charAt(k) != p.charAt(q))
▶        k = lps[k - 1];
8      if (p.charAt(k) == p.charAt(q))
9        k++;
10     lps[q] = k;
11   }
12   return lps;
13 }
```



p[1] = 'C' ≠ p[3] = 'B' → geri dön: k = lps[0] = 0.

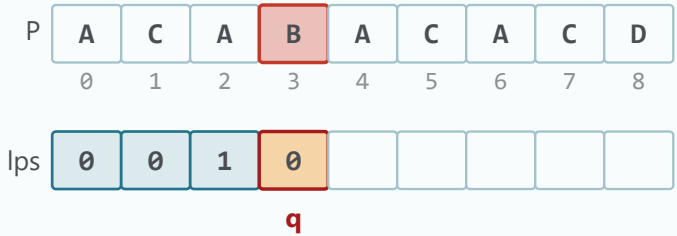
Adım 4 / 13

DEĞİŞKENLER

`m = 9` `q = 3` `k = 0`

LPS Tablosunu Kurmak: "ACABACACD"

```
1  int[] lpsHesapla(String p) {
2    int m = p.length(), k = 0;
3    int[] lps = new int[m];
4    for (int q = 1; q < m; q++) {
5      while (k > 0 &&
6        p.charAt(k) != p.charAt(q))
7        k = lps[k - 1];
       if (p.charAt(k) == p.charAt(q))
9        k++;
       lps[q] = k;
11   }
12   return lps;
13 }
```



p[0] = 'A' ≠ p[3] = 'B' → k = 0, lps[3] = 0.

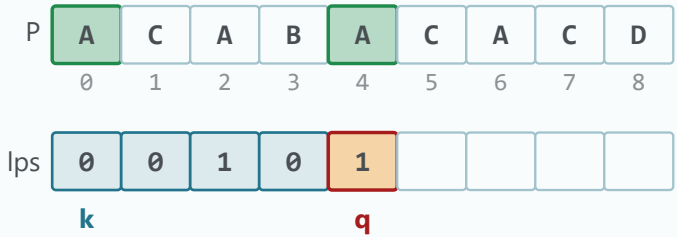
Adım 5 / 13

DEĞİŞKENLER

`m = 9` `q = 3` `k = 0`

LPS Tablosunu Kurmak: "ACABACACD"

```
1  int[] lpsHesapla(String p) {
2    int m = p.length(), k = 0;
3    int[] lps = new int[m];
4    for (int q = 1; q < m; q++) {
5      while (k > 0 &&
6        p.charAt(k) != p.charAt(q))
7        k = lps[k - 1];
▶      if (p.charAt(k) == p.charAt(q))
▶        k++;
▶      lps[q] = k;
11   }
12   return lps;
13 }
```



p[0] = 'A' = p[4] = 'A' → k = 1, lps[4] = 1.

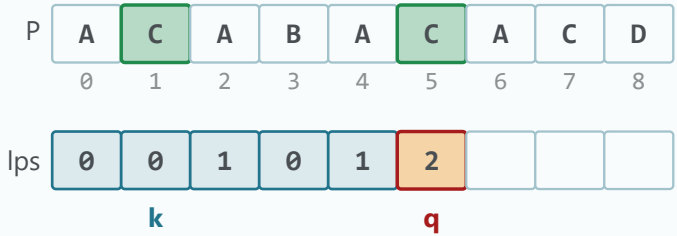
Adım 6 / 13

DEĞİŞKENLER

`m = 9` `q = 4` `k = 1`

LPS Tablosunu Kurmak: "ACABACACD"

```
1  int[] lpsHesapla(String p) {
2    int m = p.length(), k = 0;
3    int[] lps = new int[m];
4    for (int q = 1; q < m; q++) {
5      while (k > 0 &&
6        p.charAt(k) != p.charAt(q))
7        k = lps[k - 1];
▶      if (p.charAt(k) == p.charAt(q))
▶        k++;
▶      lps[q] = k;
11   }
12   return lps;
13 }
```



p[1] = 'C' = p[5] = 'C' → k = 2, lps[5] = 2. "AC" hem önek hem sonek.

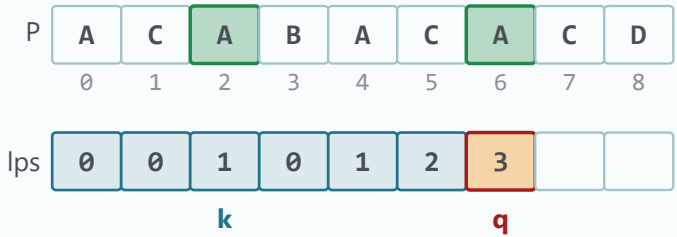
Adım 7 / 13

DEĞİŞKENLER

`m = 9` `q = 5` `k = 2`

LPS Tablosunu Kurmak: "ACABACACD"

```
1  int[] lpsHesapla(String p) {
2    int m = p.length(), k = 0;
3    int[] lps = new int[m];
4    for (int q = 1; q < m; q++) {
5      while (k > 0 &&
6        p.charAt(k) != p.charAt(q))
7        k = lps[k - 1];
▶      if (p.charAt(k) == p.charAt(q))
▶        k++;
▶      lps[q] = k;
11   }
12   return lps;
13 }
```



p[2] = 'A' = p[6] = 'A' → k = 3, lps[6] = 3. "ACA" hem önek hem sonek.

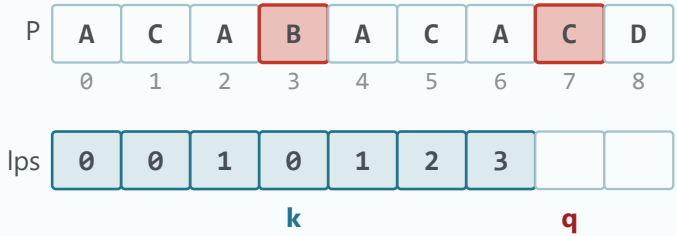
Adım 8 / 13

DEĞİŞKENLER

`m = 9` `q = 6` `k = 3`

LPS Tablosunu Kurmak: "ACABACACD"

```
1  int[] lpsHesapla(String p) {
2    int m = p.length(), k = 0;
3    int[] lps = new int[m];
4    for (int q = 1; q < m; q++) {
▶      while (k > 0 &&
▶        p.charAt(k) != p.charAt(q))
▶        k = lps[k - 1];
8      if (p.charAt(k) == p.charAt(q))
9        k++;
10     lps[q] = k;
11   }
12   return lps;
13 }
```



p[3] = 'B' ≠ p[7] = 'C' → geri dön: k = lps[2] = 1.

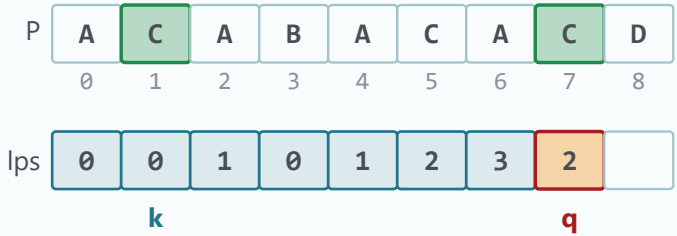
Adım 9 / 13

DEĞİŞKENLER

`m = 9` `q = 7` `k = 1`

LPS Tablosunu Kurmak: "ACABACACD"

```
1  int[] lpsHesapla(String p) {
2    int m = p.length(), k = 0;
3    int[] lps = new int[m];
4    for (int q = 1; q < m; q++) {
5      while (k > 0 &&
6        p.charAt(k) != p.charAt(q))
7        k = lps[k - 1];
▶      if (p.charAt(k) == p.charAt(q))
▶        k++;
▶      lps[q] = k;
11   }
12   return lps;
13 }
```



p[1] = 'C' = p[7] = 'C' → k = 2, lps[7] = 2. "AC" hem önek hem sonek.

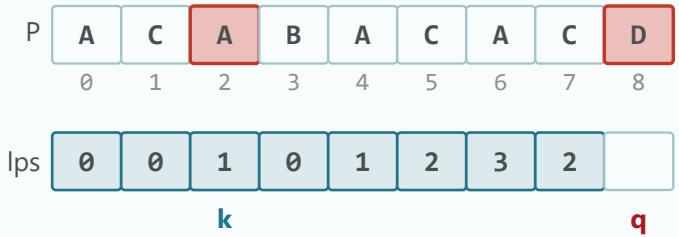
Adım 10 / 13

DEĞİŞKENLER

`m = 9` `q = 7` `k = 2`

LPS Tablosunu Kurmak: "ACABACACD"

```
1  int[] lpsHesapla(String p) {
2    int m = p.length(), k = 0;
3    int[] lps = new int[m];
4    for (int q = 1; q < m; q++) {
▶      while (k > 0 &&
▶        p.charAt(k) != p.charAt(q))
▶        k = lps[k - 1];
8      if (p.charAt(k) == p.charAt(q))
9        k++;
10     lps[q] = k;
11   }
12   return lps;
13 }
```



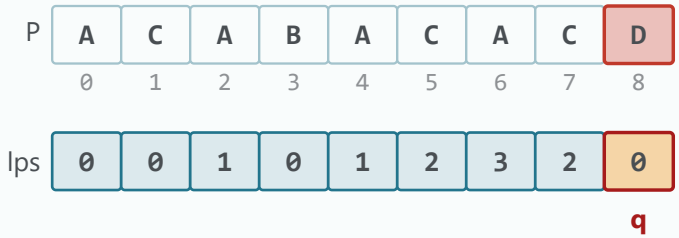
p[2] = 'A' ≠ p[8] = 'D' → geri dön: k = lps[1] = 0.

DEĞİŞKENLER

`m = 9` `q = 8` `k = 0`

LPS Tablosunu Kurmak: "ACABACACD"

```
1  int[] lpsHesapla(String p) {
2    int m = p.length(), k = 0;
3    int[] lps = new int[m];
4    for (int q = 1; q < m; q++) {
5      while (k > 0 &&
6        p.charAt(k) != p.charAt(q))
7        k = lps[k - 1];
       if (p.charAt(k) == p.charAt(q))
9        k++;
       lps[q] = k;
11   }
12   return lps;
13 }
```



p[0] = 'A' ≠ p[8] = 'D' → k = 0, lps[8] = 0.

Adım 12 / 13

DEĞİŞKENLER

`m = 9` `q = 8` `k = 0`

LPS Tablosunu Kurmak: "ACABACACD"

```
1  int[] lpsHesapla(String p) {
2    int m = p.length(), k = 0;
3    int[] lps = new int[m];
4    for (int q = 1; q < m; q++) {
5      while (k > 0 &&
6        p.charAt(k) != p.charAt(q))
7        k = lps[k - 1];
8      if (p.charAt(k) == p.charAt(q))
9        k++;
10     lps[q] = k;
11   }
▶    return lps;
13 }
```

	0	1	2	3	4	5	6	7	8
P	A	C	A	B	A	C	A	C	D
lps	0	0	1	0	1	2	3	2	0

Sonuç: lps = [0, 0, 1, 0, 1, 2, 3, 2, 0]. Her q için en fazla sabit sayıda iş (amortize) → O(m).

Adım 13 / 13

DEĞİŞKENLER

`m = 9` `q = 9` `k = 0`

KMP Aramasının Üç Evresi

Evre 1 · İlk karakteri arama

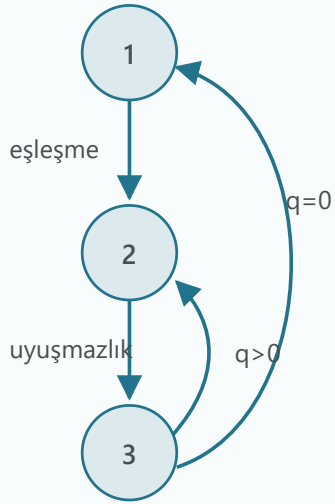
Örüntünün ilk sembolüyle eşleşme bulunana kadar metin işaretçisi (i) ilerletilir.

Evre 2 · Birlikte ilerleme

Semboller eşleştikçe iki işaretçi de ilerler. Örüntünün sonuna ulaşılırsa (metin konumu − örüntü konumu) döndürülür.

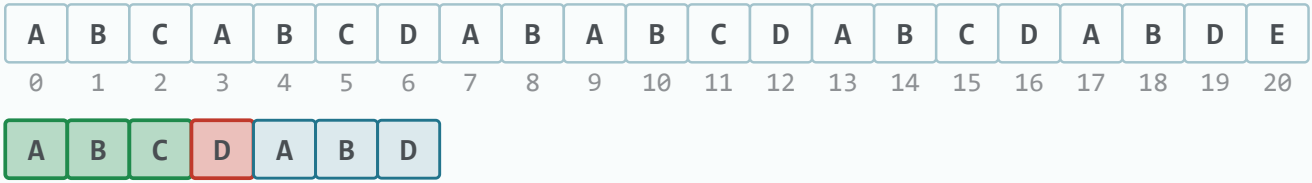
Evre 3 · LPS ile geri çekilme

Uyuşmazlıkta yalnızca örüntü işaretçisi (q) LPS ile geri çekilir; eşleşme ya da 0 konumuna ulaşılınca Evre 2'ye (q = 0 ise Evre 1'e) dönülür.



KMP Örneği: "ABCDABD" Aranıyor

lps("ABCDABD") = [0, 0, 0, 0, 1, 2, 0]. yeşil eşleşen, kırmızı uyuşmayan, mavi LPS sayesinde bilinen (tekrar karşılaştırılmayan) karakterler.



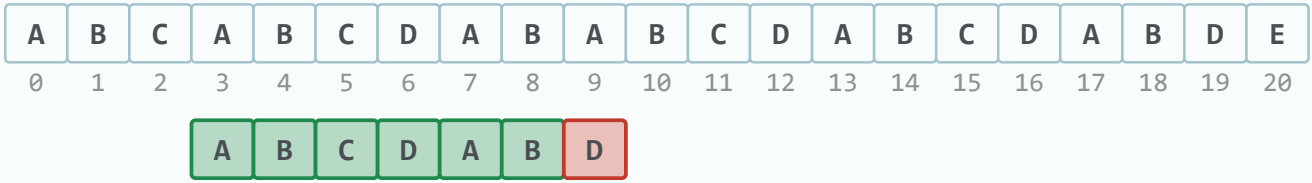
s = 0: ABC eşleşti, t[3] = 'A' ≠ p[3] = 'D'. q = 3 → q = lps[2] = **0**; örüntü 3 kayar, t[3] yeniden p[0] ile karşılaştırılır.

Adım 1 / 5

Toplam 24 karakter karşılaştırması (n = 21); kaba kuvvet aynı metinde 37 karşılaştırma yapar.

KMP Örneği: "ABCDABD" Aranıyor

lps("ABCDABD") = [0, 0, 0, 0, 1, 2, 0]. yeşil eşleşen, kırmızı uyuşmayan, mavi LPS sayesinde bilinen (tekrar karşılaştırılmayan) karakterler.

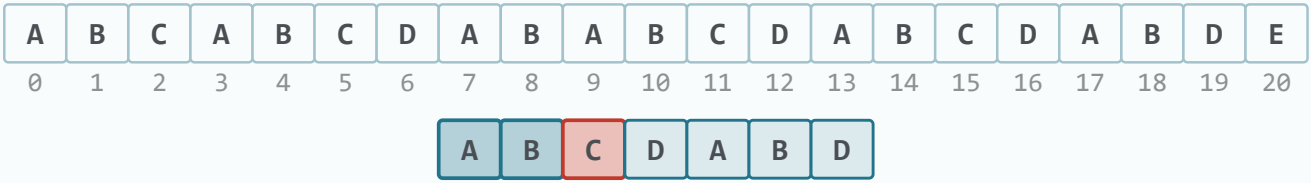


s = 3: ABCDAB eşleşti, t[9] = 'A' ≠ 'D'. q = 6 → q = lps[5] = **2**: "AB" zaten eşleşmiş bilinir, örüntü 4 kayar.

Toplam 24 karakter karşılaştırması (n = 21); kaba kuvvet aynı metinde 37 karşılaştırma yapar.

KMP Örneği: "ABCDABD" Aranıyor

lps("ABCDABD") = [0, 0, 0, 0, 1, 2, 0]. yeşil eşleşen, kırmızı uyuşmayan, mavi LPS sayesinde bilinen (tekrar karşılaştırılmayan) karakterler.

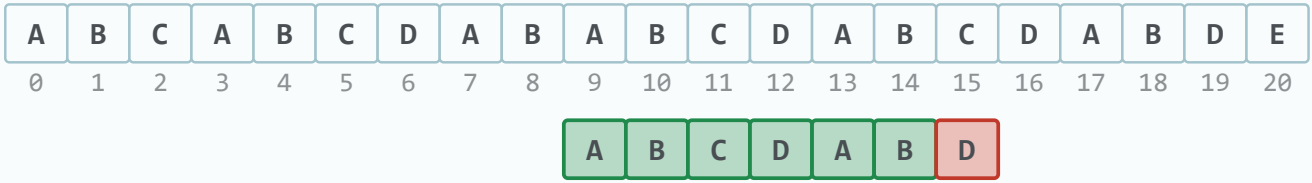


s = 7: AB yeniden karşılaştırılmaz. t[9] = 'A' ≠ p[2] = 'C' → q = lps[1] = **0**.

Toplam 24 karakter karşılaştırması (n = 21); kaba kuvvet aynı metinde 37 karşılaştırma yapar.

KMP Örneği: "ABCDABD" Aranıyor

lps("ABCDABD") = [0, 0, 0, 0, 1, 2, 0]. yeşil eşleşen, kırmızı uyuşmayan, mavi LPS sayesinde bilinen (tekrar karşılaştırılmayan) karakterler.

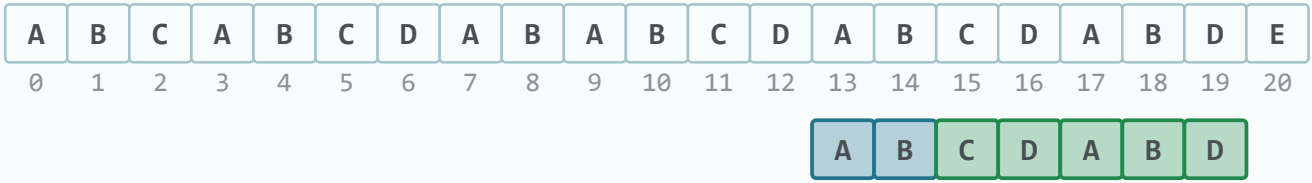


s = 9: ABCDAB eşleşti, t[15] = 'C' ≠ 'D'. q = 6 → q = lps[5] = **2**; örüntü 4 kayar.

Toplam 24 karakter karşılaştırması (n = 21); kaba kuvvet aynı metinde 37 karşılaştırma yapar.

KMP Örneği: "ABCDABD" Aranıyor

lps("ABCDABD") = [0, 0, 0, 0, 1, 2, 0]. yeşil eşleşen, kırmızı uyuşmayan, mavi LPS sayesinde bilinen (tekrar karşılaştırılmayan) karakterler.



s = 13: AB bilinir; CDABD da eşleşir → örüntü **13. konumda** bulundu. Metin işaretçisi hiç geri gitmedi.

Adım 5 / 5

Toplam 24 karakter karşılaştırması (n = 21); kaba kuvvet aynı metinde 37 karşılaştırma yapar.

KMP · KODU ADIM ADIM ÇALIŞTIR

kmpAra("ababcababcababababd", "ababd")

```
1  int kmpAra(String t, String p) {
▶    int n = t.length(), m = p.length();
▶    int[] lps = lpsHesapla(p);
▶    int q = 0;  // eşleşen uzunluk
5    for (int i = 0; i < n; i++) {
6      while (q > 0 &&
7        p.charAt(q) != t.charAt(i))
8        q = lps[q - 1];
9      if (p.charAt(q) == t.charAt(i))
10       q++;
11     if (q == m)
12       return i - m + 1;
13   }
14   return -1;
15 }
```

t	a	b	a	b	c	a	b	a	b	c	a	b	a	b	a	b	d
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

lps = [0, 0, 1, 2, 0]

n = 17, m = 5, lps = [0, 0, 1, 2, 0], q = 0.

Adım 1 / 24

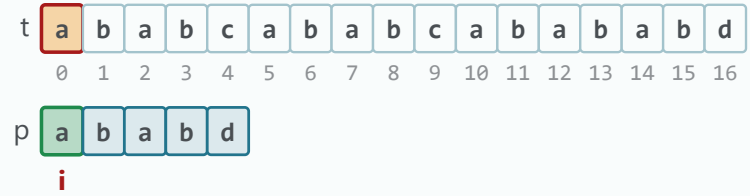
DEĞİŞKENLER

n = **17** m = **5** i = **–** q = **0**

KMP · KODU ADIM ADIM ÇALIŞTIR

kmpAra("ababcababcababababd", "ababd")

```
1  int kmpAra(String t, String p) {
2    int n = t.length(), m = p.length();
3    int[] lps = lpsHesapla(p);
4    int q = 0;  // eşleşen uzunluk
5    for (int i = 0; i < n; i++) {
6      while (q > 0 &&
7        p.charAt(q) != t.charAt(i))
8        q = lps[q - 1];
▶      if (p.charAt(q) == t.charAt(i))
▶        q++;
11     if (q == m)
12       return i - m + 1;
13   }
14   return -1;
15 }
```



lps = [0, 0, 1, 2, 0]

p[0] = t[0] = 'a' → q = 1.

Adım 2 / 24

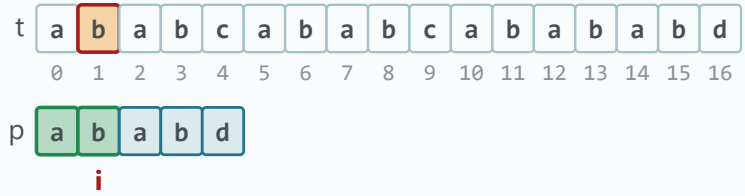
DEĞİŞKENLER

n = 17 | m = 5 | i = 0 | q = 1

KMP · KODU ADIM ADIM ÇALIŞTIR

kmpAra("ababcababcababababd", "ababd")

```
1  int kmpAra(String t, String p) {
2    int n = t.length(), m = p.length();
3    int[] lps = lpsHesapla(p);
4    int q = 0;  // eşleşen uzunluk
5    for (int i = 0; i < n; i++) {
6      while (q > 0 &&
7        p.charAt(q) != t.charAt(i))
8        q = lps[q - 1];
▶      if (p.charAt(q) == t.charAt(i))
▶        q++;
11     if (q == m)
12       return i - m + 1;
13   }
14   return -1;
15 }
```



lps = [0, 0, 1, 2, 0]

p[1] = t[1] = 'b' → q = 2.

Adım 3 / 24

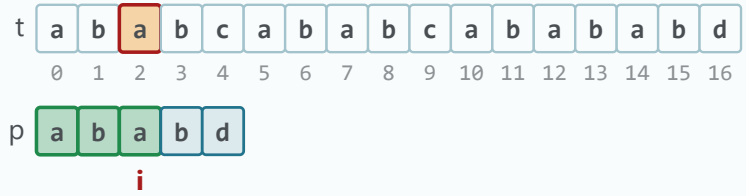
DEĞİŞKENLER

n = 17 | m = 5 | i = 1 | q = 2

KMP · KODU ADIM ADIM ÇALIŞTIR

kmpAra("ababcababcababababd", "ababd")

```
1  int kmpAra(String t, String p) {
2    int n = t.length(), m = p.length();
3    int[] lps = lpsHesapla(p);
4    int q = 0;  // eşleşen uzunluk
5    for (int i = 0; i < n; i++) {
6      while (q > 0 &&
7        p.charAt(q) != t.charAt(i))
8        q = lps[q - 1];
▶      if (p.charAt(q) == t.charAt(i))
▶        q++;
11     if (q == m)
12       return i - m + 1;
13   }
14   return -1;
15 }
```



lps = [0, 0, 1, 2, 0]

p[2] = t[2] = 'a' → q = 3.

Adım 4 / 24

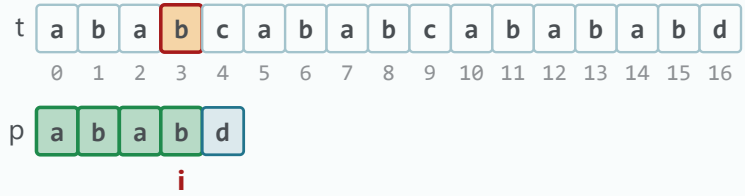
DEĞİŞKENLER

n = 17 | m = 5 | i = 2 | q = 3

KMP · KODU ADIM ADIM ÇALIŞTIR

kmpAra("ababcababcababababd", "ababd")

```
1  int kmpAra(String t, String p) {
2    int n = t.length(), m = p.length();
3    int[] lps = lpsHesapla(p);
4    int q = 0;  // eşleşen uzunluk
5    for (int i = 0; i < n; i++) {
6      while (q > 0 &&
7        p.charAt(q) != t.charAt(i))
8        q = lps[q - 1];
▶      if (p.charAt(q) == t.charAt(i))
▶        q++;
11     if (q == m)
12       return i - m + 1;
13   }
14   return -1;
15 }
```



lps = [0, 0, 1, 2, 0]

p[3] = t[3] = 'b' → q = 4.

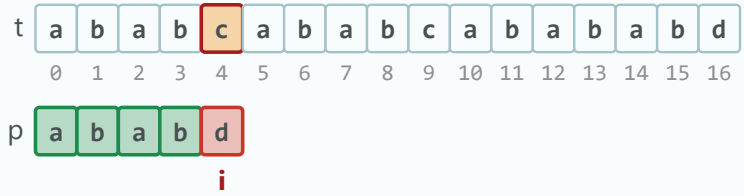
Adım 5 / 24

DEĞİŞKENLER

n = 17 | m = 5 | i = 3 | q = 4

kmpAra("ababcababcababababd", "ababd")

```
1  int kmpAra(String t, String p) {
2    int n = t.length(), m = p.length();
3    int[] lps = lpsHesapla(p);
4    int q = 0;  // eşleşen uzunluk
5    for (int i = 0; i < n; i++) {
▶      while (q > 0 &&
▶        p.charAt(q) != t.charAt(i))
▶        q = lps[q - 1];
9      if (p.charAt(q) == t.charAt(i))
10       q++;
11     if (q == m)
12       return i - m + 1;
13   }
14   return -1;
15 }
```



lps = [0, 0, 1, 2, 0]

p[4] = 'd' ≠ t[4] = 'c' → q = lps[3] = 2 (i yerinde kalır).

Adım 6 / 24

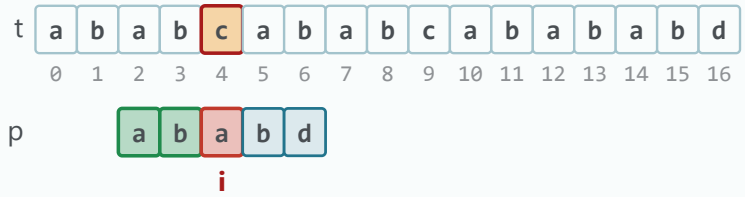
DEĞİŞKENLER

`n = 17` `m = 5` `i = 4` `q = 2`

KMP · KODU ADIM ADIM ÇALIŞTIR

kmpAra("ababcababcababababd", "ababd")

```
1  int kmpAra(String t, String p) {
2    int n = t.length(), m = p.length();
3    int[] lps = lpsHesapla(p);
4    int q = 0;  // eşleşen uzunluk
5    for (int i = 0; i < n; i++) {
▶      while (q > 0 &&
▶        p.charAt(q) != t.charAt(i))
▶        q = lps[q - 1];
9      if (p.charAt(q) == t.charAt(i))
10       q++;
11     if (q == m)
12       return i - m + 1;
13   }
14   return -1;
15 }
```



lps = [0, 0, 1, 2, 0]

p[2] = 'a' ≠ t[4] = 'c' → q = lps[1] = 0 (i yerinde kalır).

Adım 7 / 24

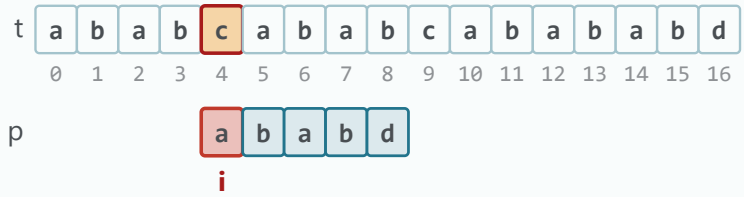
DEĞİŞKENLER

n = 17 | m = 5 | i = 4 | q = 0

KMP · KODU ADIM ADIM ÇALIŞTIR

kmpAra("ababcababcababababd", "ababd")

```
1  int kmpAra(String t, String p) {
2    int n = t.length(), m = p.length();
3    int[] lps = lpsHesapla(p);
4    int q = 0;  // eşleşen uzunluk
5    for (int i = 0; i < n; i++) {
6      while (q > 0 &&
7        p.charAt(q) != t.charAt(i))
8        q = lps[q - 1];
▶      if (p.charAt(q) == t.charAt(i))
10       q++;
11     if (q == m)
12       return i - m + 1;
13   }
14   return -1;
15 }
```



lps = [0, 0, 1, 2, 0]

p[0] = 'a' ≠ t[4] = 'c' → q = 0 kalır, i ilerler.

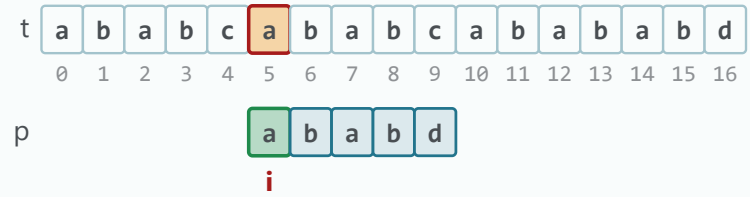
Adım 8 / 24

DEĞİŞKENLER

n = 17 | m = 5 | i = 4 | q = 0

kmpAra("ababcababcababababd", "ababd")

```
1  int kmpAra(String t, String p) {
2    int n = t.length(), m = p.length();
3    int[] lps = lpsHesapla(p);
4    int q = 0;  // eşleşen uzunluk
5    for (int i = 0; i < n; i++) {
6      while (q > 0 &&
7        p.charAt(q) != t.charAt(i))
8        q = lps[q - 1];
▶      if (p.charAt(q) == t.charAt(i))
▶        q++;
11     if (q == m)
12       return i - m + 1;
13   }
14   return -1;
15 }
```



lps = [0, 0, 1, 2, 0]

p[0] = t[5] = 'a' → q = 1.

Adım 9 / 24

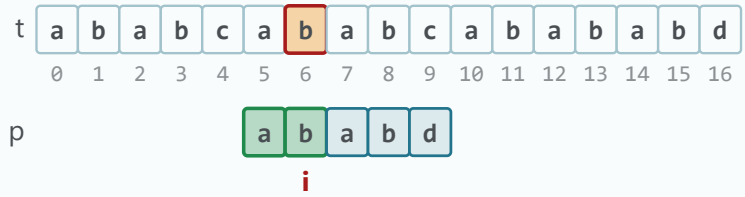
DEĞİŞKENLER

`n = 17` `m = 5` `i = 5` `q = 1`

KMP · KODU ADIM ADIM ÇALIŞTIR

kmpAra("ababcababcababababd", "ababd")

```
1  int kmpAra(String t, String p) {
2    int n = t.length(), m = p.length();
3    int[] lps = lpsHesapla(p);
4    int q = 0;  // eşleşen uzunluk
5    for (int i = 0; i < n; i++) {
6      while (q > 0 &&
7        p.charAt(q) != t.charAt(i))
8        q = lps[q - 1];
▶      if (p.charAt(q) == t.charAt(i))
▶        q++;
11     if (q == m)
12       return i - m + 1;
13   }
14   return -1;
15 }
```



lps = [0, 0, 1, 2, 0]

p[1] = t[6] = 'b' → q = 2.

Adım 10 / 24

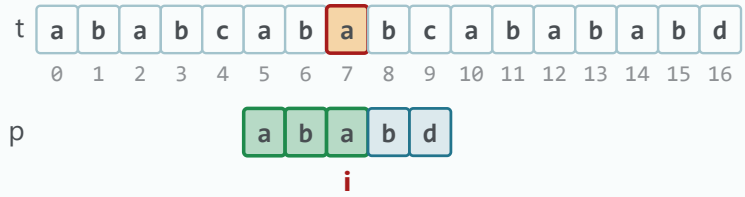
DEĞİŞKENLER

n = 17 | m = 5 | i = 6 | q = 2

KMP · KODU ADIM ADIM ÇALIŞTIR

kmpAra("ababcababcababababd", "ababd")

```
1  int kmpAra(String t, String p) {
2    int n = t.length(), m = p.length();
3    int[] lps = lpsHesapla(p);
4    int q = 0;  // eşleşen uzunluk
5    for (int i = 0; i < n; i++) {
6      while (q > 0 &&
7        p.charAt(q) != t.charAt(i))
8        q = lps[q - 1];
▶      if (p.charAt(q) == t.charAt(i))
▶        q++;
11     if (q == m)
12       return i - m + 1;
13   }
14   return -1;
15 }
```



lps = [0, 0, 1, 2, 0]

p[2] = t[7] = 'a' → q = 3.

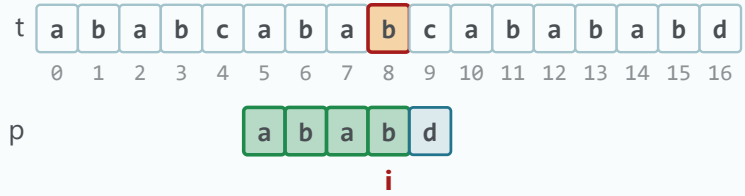
Adım 11 / 24

DEĞİŞKENLER

n = 17 | m = 5 | i = 7 | q = 3

kmpAra("ababcababcababababd", "ababd")

```
1  int kmpAra(String t, String p) {
2    int n = t.length(), m = p.length();
3    int[] lps = lpsHesapla(p);
4    int q = 0;  // eşleşen uzunluk
5    for (int i = 0; i < n; i++) {
6      while (q > 0 &&
7        p.charAt(q) != t.charAt(i))
8        q = lps[q - 1];
▶      if (p.charAt(q) == t.charAt(i))
▶        q++;
11     if (q == m)
12       return i - m + 1;
13   }
14   return -1;
15 }
```



lps = [0, 0, 1, 2, 0]

p[3] = t[8] = 'b' → q = 4.

Adım 12 / 24

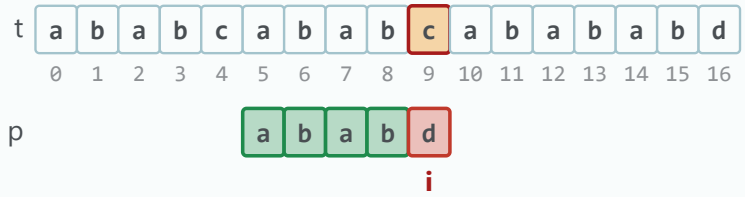
DEĞİŞKENLER

n = 17 | m = 5 | i = 8 | q = 4

KMP · KODU ADIM ADIM ÇALIŞTIR

kmpAra("ababcababcababababd", "ababd")

```
1  int kmpAra(String t, String p) {
2    int n = t.length(), m = p.length();
3    int[] lps = lpsHesapla(p);
4    int q = 0;  // eşleşen uzunluk
5    for (int i = 0; i < n; i++) {
▶      while (q > 0 &&
▶        p.charAt(q) != t.charAt(i))
▶        q = lps[q - 1];
9      if (p.charAt(q) == t.charAt(i))
10       q++;
11     if (q == m)
12       return i - m + 1;
13   }
14   return -1;
15 }
```



lps = [0, 0, 1, 2, 0]

p[4] = 'd' ≠ t[9] = 'c' → q = lps[3] = 2 (i yerinde kalır).

Adım 13 / 24

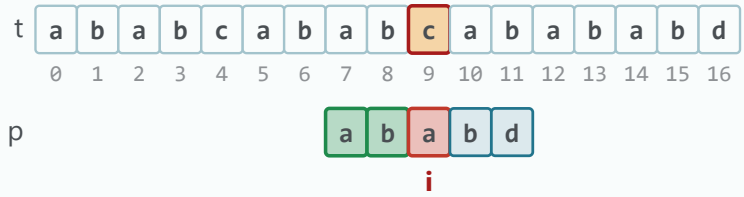
DEĞİŞKENLER

n = 17 | m = 5 | i = 9 | q = 2

KMP · KODU ADIM ADIM ÇALIŞTIR

kmpAra("ababcababcababababd", "ababd")

```
1  int kmpAra(String t, String p) {
2    int n = t.length(), m = p.length();
3    int[] lps = lpsHesapla(p);
4    int q = 0;  // eşleşen uzunluk
5    for (int i = 0; i < n; i++) {
▶      while (q > 0 &&
▶        p.charAt(q) != t.charAt(i))
▶        q = lps[q - 1];
9      if (p.charAt(q) == t.charAt(i))
10       q++;
11     if (q == m)
12       return i - m + 1;
13   }
14   return -1;
15 }
```



lps = [0, 0, 1, 2, 0]

p[2] = 'a' ≠ t[9] = 'c' → q = lps[1] = 0 (i yerinde kalır).

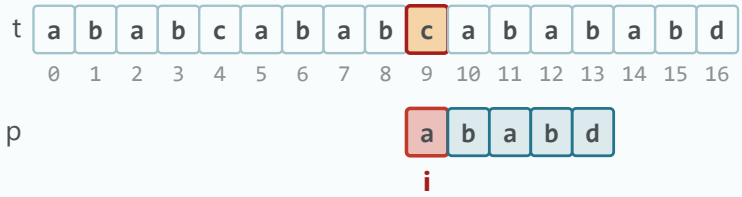
Adım 14 / 24

DEĞİŞKENLER

n = 17 | m = 5 | i = 9 | q = 0

kmpAra("ababcababcababababd", "ababd")

```
1  int kmpAra(String t, String p) {
2    int n = t.length(), m = p.length();
3    int[] lps = lpsHesapla(p);
4    int q = 0;  // eşleşen uzunluk
5    for (int i = 0; i < n; i++) {
6      while (q > 0 &&
7        p.charAt(q) != t.charAt(i))
8        q = lps[q - 1];
▶      if (p.charAt(q) == t.charAt(i))
10       q++;
11     if (q == m)
12       return i - m + 1;
13   }
14   return -1;
15 }
```



lps = [0, 0, 1, 2, 0]

p[0] = 'a' ≠ t[9] = 'c' → q = 0 kalır, i ilerler.

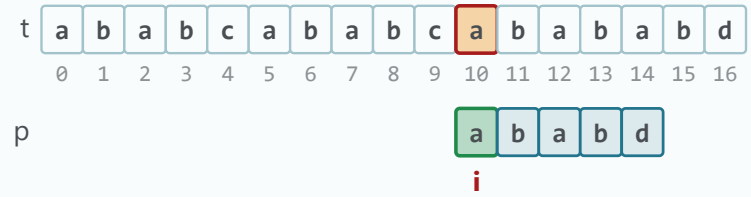
Adım 15 / 24

DEĞİŞKENLER

`n = 17` `m = 5` `i = 9` `q = 0`

kmpAra("ababcababcababababd", "ababd")

```
1  int kmpAra(String t, String p) {
2    int n = t.length(), m = p.length();
3    int[] lps = lpsHesapla(p);
4    int q = 0;  // eşleşen uzunluk
5    for (int i = 0; i < n; i++) {
6      while (q > 0 &&
7        p.charAt(q) != t.charAt(i))
8        q = lps[q - 1];
▶      if (p.charAt(q) == t.charAt(i))
▶        q++;
11     if (q == m)
12       return i - m + 1;
13   }
14   return -1;
15 }
```



lps = [0, 0, 1, 2, 0]

p[0] = t[10] = 'a' → q = 1.

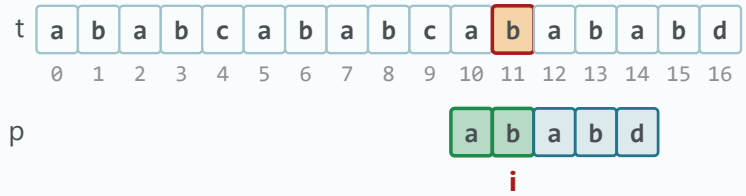
Adım 16 / 24

DEĞİŞKENLER

n = 17 | m = 5 | i = 10 | q = 1

kmpAra("ababcababcababababd", "ababd")

```
1  int kmpAra(String t, String p) {
2    int n = t.length(), m = p.length();
3    int[] lps = lpsHesapla(p);
4    int q = 0;  // eşleşen uzunluk
5    for (int i = 0; i < n; i++) {
6      while (q > 0 &&
7        p.charAt(q) != t.charAt(i))
8        q = lps[q - 1];
▶      if (p.charAt(q) == t.charAt(i))
▶        q++;
11     if (q == m)
12       return i - m + 1;
13   }
14   return -1;
15 }
```



lps = [0, 0, 1, 2, 0]

p[1] = t[11] = 'b' → q = 2.

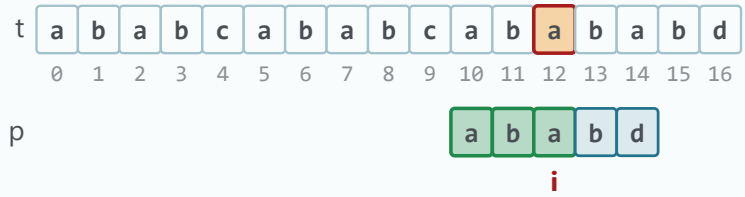
Adım 17 / 24

DEĞİŞKENLER

n = 17 | m = 5 | i = 11 | q = 2

kmpAra("ababcababcababababd", "ababd")

```
1  int kmpAra(String t, String p) {
2    int n = t.length(), m = p.length();
3    int[] lps = lpsHesapla(p);
4    int q = 0;  // eşleşen uzunluk
5    for (int i = 0; i < n; i++) {
6      while (q > 0 &&
7        p.charAt(q) != t.charAt(i))
8        q = lps[q - 1];
▶      if (p.charAt(q) == t.charAt(i))
▶        q++;
11     if (q == m)
12       return i - m + 1;
13   }
14   return -1;
15 }
```



lps = [0, 0, 1, 2, 0]

p[2] = t[12] = 'a' → q = 3.

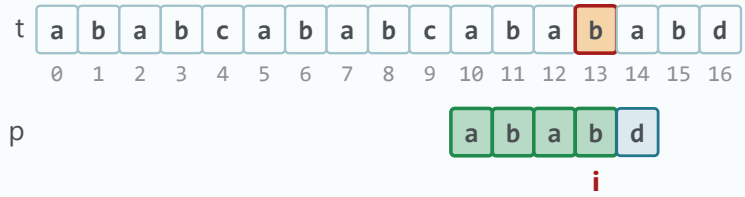
Adım 18 / 24

DEĞİŞKENLER

n = **17** | m = **5** | i = **12** | q = **3**

kmpAra("ababcababcababababd", "ababd")

```
1  int kmpAra(String t, String p) {
2    int n = t.length(), m = p.length();
3    int[] lps = lpsHesapla(p);
4    int q = 0;  // eşleşen uzunluk
5    for (int i = 0; i < n; i++) {
6      while (q > 0 &&
7        p.charAt(q) != t.charAt(i))
8        q = lps[q - 1];
▶      if (p.charAt(q) == t.charAt(i))
▶        q++;
11     if (q == m)
12       return i - m + 1;
13   }
14   return -1;
15 }
```



lps = [0, 0, 1, 2, 0]

p[3] = t[13] = 'b' → q = 4.

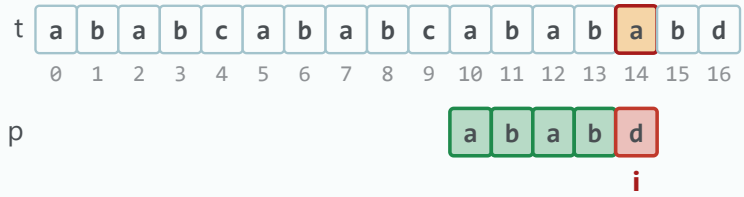
Adım 19 / 24

DEĞİŞKENLER

`n = 17` `m = 5` `i = 13` `q = 4`

kmpAra("ababcababcababababd", "ababd")

```
1  int kmpAra(String t, String p) {
2    int n = t.length(), m = p.length();
3    int[] lps = lpsHesapla(p);
4    int q = 0;  // eşleşen uzunluk
5    for (int i = 0; i < n; i++) {
▶      while (q > 0 &&
▶        p.charAt(q) != t.charAt(i))
▶        q = lps[q - 1];
9      if (p.charAt(q) == t.charAt(i))
10       q++;
11     if (q == m)
12       return i - m + 1;
13   }
14   return -1;
15 }
```



lps = [0, 0, 1, 2, 0]

p[4] = 'd' ≠ t[14] = 'a' → q = lps[3] = 2 (i yerinde kalır).

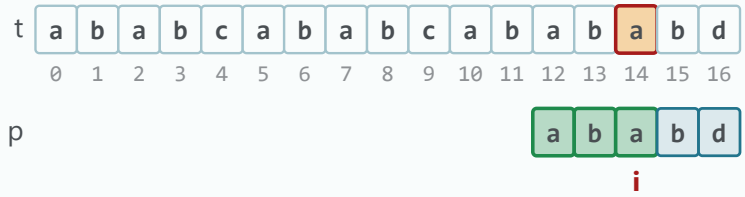
Adım 20 / 24

DEĞİŞKENLER

`n = 17` `m = 5` `i = 14` `q = 2`

kmpAra("ababcababcababababd", "ababd")

```
1  int kmpAra(String t, String p) {
2    int n = t.length(), m = p.length();
3    int[] lps = lpsHesapla(p);
4    int q = 0;  // eşleşen uzunluk
5    for (int i = 0; i < n; i++) {
6      while (q > 0 &&
7        p.charAt(q) != t.charAt(i))
8        q = lps[q - 1];
▶      if (p.charAt(q) == t.charAt(i))
▶        q++;
11     if (q == m)
12       return i - m + 1;
13   }
14   return -1;
15 }
```



lps = [0, 0, 1, 2, 0]

p[2] = t[14] = 'a' → q = 3.

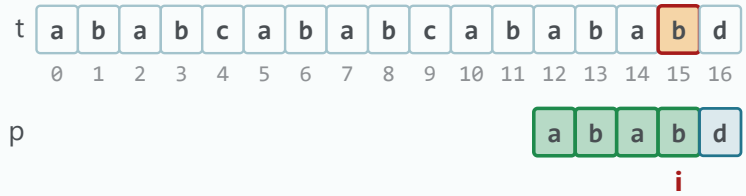
Adım 21 / 24

DEĞİŞKENLER

`n = 17` `m = 5` `i = 14` `q = 3`

kmpAra("ababcababcababababd", "ababd")

```
1  int kmpAra(String t, String p) {
2    int n = t.length(), m = p.length();
3    int[] lps = lpsHesapla(p);
4    int q = 0;  // eşleşen uzunluk
5    for (int i = 0; i < n; i++) {
6      while (q > 0 &&
7        p.charAt(q) != t.charAt(i))
8        q = lps[q - 1];
▶      if (p.charAt(q) == t.charAt(i))
▶        q++;
11     if (q == m)
12       return i - m + 1;
13   }
14   return -1;
15 }
```



lps = [0, 0, 1, 2, 0]

p[3] = t[15] = 'b' → q = 4.

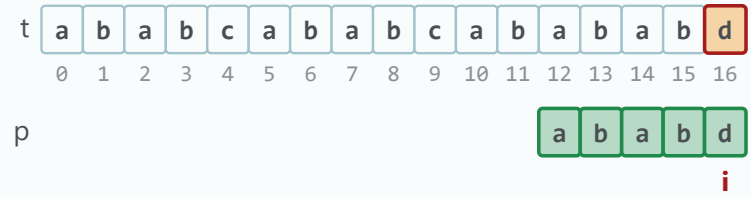
Adım 22 / 24

DEĞİŞKENLER

`n = 17` `m = 5` `i = 15` `q = 4`

kmpAra("ababcababcababababd", "ababd")

```
1  int kmpAra(String t, String p) {
2    int n = t.length(), m = p.length();
3    int[] lps = lpsHesapla(p);
4    int q = 0;  // eşleşen uzunluk
5    for (int i = 0; i < n; i++) {
6      while (q > 0 &&
7        p.charAt(q) != t.charAt(i))
8        q = lps[q - 1];
▶      if (p.charAt(q) == t.charAt(i))
▶        q++;
11     if (q == m)
12       return i - m + 1;
13   }
14   return -1;
15 }
```



lps = [0, 0, 1, 2, 0]

p[4] = t[16] = 'd' → q = 5.

Adım 23 / 24

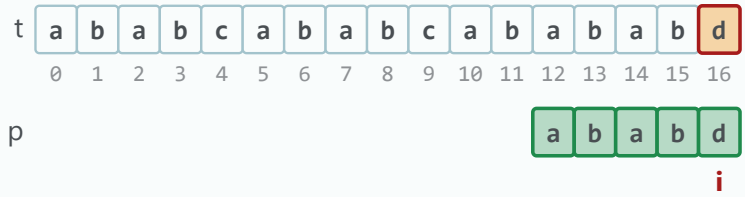
DEĞİŞKENLER

n = **17** | m = **5** | i = **16** | q = **5**

KMP · KODU ADIM ADIM ÇALIŞTIR

kmpAra("ababcababcababababd", "ababd")

```
1  int kmpAra(String t, String p) {
2    int n = t.length(), m = p.length();
3    int[] lps = lpsHesapla(p);
4    int q = 0;  // eşleşen uzunluk
5    for (int i = 0; i < n; i++) {
6      while (q > 0 &&
7        p.charAt(q) != t.charAt(i))
8        q = lps[q - 1];
9      if (p.charAt(q) == t.charAt(i))
10       q++;
▶      if (q == m)
▶        return i - m + 1;
13   }
14   return -1;
15 }
```



lps = [0, 0, 1, 2, 0]

q == m → **i − m + 1 = 12** döndürülür. Metin yalnızca bir kez tarandı.

Adım 24 / 24

DEĞİŞKENLER

`n = 17` `m = 5` `i = 16` `q = 5`

KMP: Neden O(n + m)?

- Dış döngü i'yi **n kez** artırır; i hiç azalmaz.
- q her adımda en fazla **1 artar** → toplam artış ≤ n.
- `while` içindeki her geri çekilme q'yu **en az 1 azaltır**; q negatif olamaz → toplam geri çekilme ≤ toplam artış ≤ n.
- Arama: en fazla **2n** karşılaştırma → O(n). Aynı amortize argüman LPS için O(m) verir.

Artıları

+ En kötü durumda bile doğrusal
+ Metinde geri gitmez: akış (stream) verisine uygun

Eksileri

– Pratikte Boyer-Moore'dan genelde yavaş
– Küçük alfabede (DNA) atlama kazancı sınırlı

Boyer-Moore Algoritması

- Metin içinde belirli bir örüntüyü arar; **Robert S. Boyer** ve **J Strother Moore** tarafından 1977'de geliştirilmiştir.
- Örüntü soldan sağa kaydırılır ama karakterler **sağdan sola** karşılaştırılır.
- Uyuşmazlıkta iki kuraldan büyük olan kadar kaydırılır: **kötü karakter** ve **iyi sonek**.
- Metindeki karakterlerin çoğu **hiç okunmadan** atlanabilir: uzun örüntü ve geniş alfabede (doğal dil metni) çok hızlıdır; `grep` ve editörlerin aramaları bu ailedendir.

Karmaşıklık

En iyi: O(n/m) — her denemede ilk karşılaştırma uyuşmaz ve m atlanır.
Ortalama: alt-doğrusal, pratikte O(n)'den hızlı.
En kötü: yalnız kötü karakterle O(n · m); iyi sonek + Galil kuralıyla O(n + m).

Boyer-Moore: İşleyiş

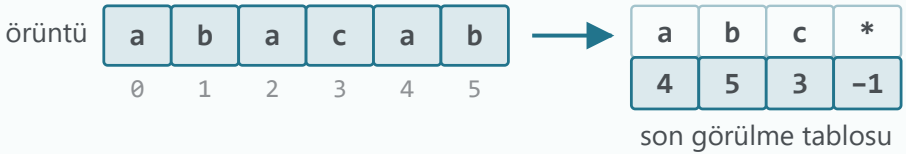
1. **Ön işleme:** örüntüden kaydırma tabloları kurulur: her karakterin örüntüdeki **son görülme** konumu (kötü karakter) ve her sonek için kaydırma miktarı (iyi sonek).
2. **Arama:** örüntü metnin altına hizalanır; karşılaştırma örüntünün **son karakterinden başlayıp sola** doğru ilerler.
3. **Eşleşme kontrolü:** tüm karakterler eşleşirse konum rapor edilir.
4. **Kötü karakter kuralı:** uyuşmayan metin karakterinin örüntüdeki **en sağdaki** konumu dikkate alınarak kaydırma yapılır.
5. **İyi sonek kuralı:** eşleşmiş sonekin örüntü içinde başka bir yerde bulunup bulunmadığına göre kaydırma yapılır.



Kırmızı sayılar karşılaştırma sırasıdır: 12 karakterlik metinde yalnızca 6 karşılaştırma.

Kaydırma = max(kötü karakter kaydırması, iyi sonek kaydırması). İkisi de "güvenli"dir: atlanan hiçbir hizada eşleşme olamaz.

Kötü Karakter (Bad Character) Kuralı

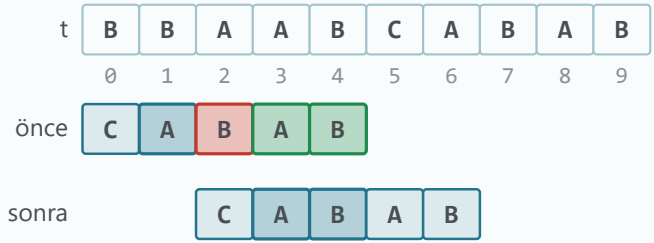


- p[j] ≠ t[s + j] olsun; c = t[s + j] **kötü karakter**.
- c örüntüde **j'nin solunda** geçiyorsa: örüntü, en sağdaki c t[s + j]'nin altına gelecek şekilde kaydırılır.
- c örüntüde **hiç yoksa** (son = −1): örüntü c'nin tamamen ötesine kaydırılır (j + 1).
- Formül: **s += max(1, j − son[c])**. c, j'nin sağında son kez geçiyorsa j − son[c] ≤ 0 olur; bu yüzden en az 1.

```
int[] sonGorulme(String p) {
  int[] son = new int[256];
  Arrays.fill(son, -1);
  for (int i = 0; i < p.length(); i++)
    son[p.charAt(i)] = i;
  return son;
}
```

İyi Sonek (Good Suffix) Kuralı

- Uyuşmazlıktan önce eşleşen sonek u = P[j+1..m−1] metinde bulunmuştur: **"iyi sonek"**.
- **Durum 1:** u örüntüde başka yerde (önünde farklı bir karakterle) geçiyorsa, örüntü, o geçiş metindeki u'nun altına gelecek şekilde kaydırılır.
- **Durum 2:** yoksa, u'nun bir sonekine eşit olan en uzun P **öneki** hizalanır.
- **Durum 3:** hiçbiri yoksa örüntü m kaydırılır.
- Tablo O(m) zamanda önceden hesaplanır (KMP'nin LPS fikrine benzer, sonekler için).



"AB" eşleşti, t[2] = 'A' ≠ p[2] = 'B'. Kötü karakter: son[A] = 3 > j → yalnız 1. İyi sonek: "AB" p[1..2]'de de var (önünde 'C' ≠ 'B') → **2** kaydır.

Örnek: "LO LOELLO O HELLO" İçinde "HELLO"

c	H	E	L	O	diğer
son[c]	0	1	3	4	−1

Kural: s += max(1, j − son[c]). yeşil eşleşen, kırmızı kötü karakter.



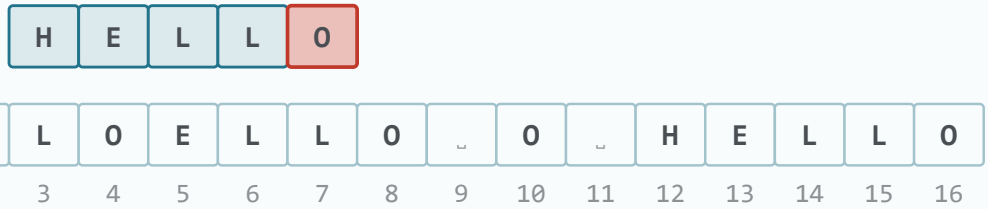


Hiza 1 (s = 0): 2 karakter eşleşti, t[2] = '␣' ≠ p[2] = 'L'. '␣' örüntüde yok → j − (−1) = 3 → **3 kaydır**.

Örnek: "LO LOELLO O HELLO" İçinde "HELLO"

c	H	E	L	O	diğer
son[c]	0	1	3	4	−1

Kural: s += max(1, j − son[c]). yeşil eşleşen, kırmızı kötü karakter.



Hiza 2 (s = 3): t[7] = 'L' ≠ p[4] = 'O'. son['L'] = 3 → j − 3 = 1 → **1 kaydır**.

Örnek: "LO LOELLO O HELLO" İçinde "HELLO"

c	H	E	L	O	diğer
son[c]	0	1	3	4	−1

Kural: s += max(1, j − son[c]). yeşil eşleşen, kırmızı kötü karakter.



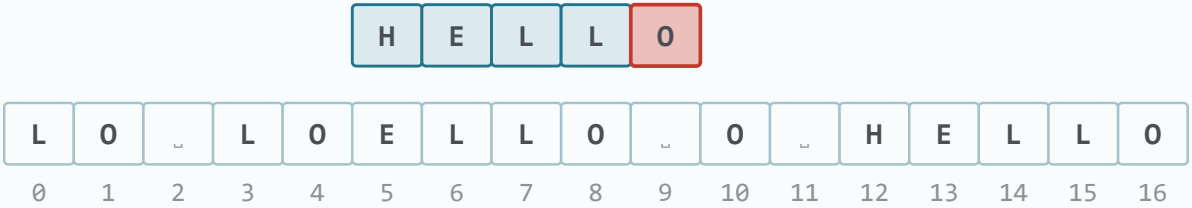


Hiza 3 (s = 4): 4 karakter eşleşti, t[4] = 'O' ≠ p[0] = 'H'. son['O'] = 4 ≥ j → en az 1 → **1 kaydır**.

Örnek: "LO LOELLO O HELLO" İçinde "HELLO"

c	H	E	L	O	diğer
son[c]	0	1	3	4	−1

Kural: s += max(1, j − son[c]). yeşil eşleşen, kırmızı kötü karakter.



Hiza 4 (s = 5): t[9] = '␣' ≠ p[4] = 'O'. '␣' örüntüde yok → j − (−1) = 5 → **5 kaydır**.

Örnek: "LO LOELLO O HELLO" İçinde "HELLO"

c	H	E	L	O	diğer
son[c]	0	1	3	4	−1

Kural: s += max(1, j − son[c]). yeşil eşleşen, kırmızı kötü karakter.



Hiza 5 (s = 10): t[14] = 'L' ≠ p[4] = 'O'. son['L'] = 3 → j − 3 = 1 → **1 kaydır**.

Örnek: "LO LOELLO O HELLO" İçinde "HELLO"

c	H	E	L	O	diğer
son[c]	0	1	3	4	−1

Kural: s += max(1, j − son[c]). yeşil eşleşen, kırmızı kötü karakter.



Hiza 6 (s = 11): t[15] = 'L' ≠ p[4] = 'O'. son['L'] = 3 → j − 3 = 1 → **1 kaydır**.

Örnek: "LO LOELLO O HELLO" İçinde "HELLO"

c	H	E	L	O	diğer
son[c]	0	1	3	4	−1

Kural: s += max(1, j − son[c]). yeşil eşleşen, kırmızı kötü karakter.



Hiza 7 (s = 12): 5 karakterin hepsi sağdan sola eşleşti → **HELLO, s = 12**'de bulundu. Toplam 7 hiza, 17 karşılaştırma.

Horspool Basitleştirmesi ve Kaydırma Tablosu

- Horspool (1980): uyuşmazlık nerede olursa olsun, kaydırma **hizanın son karakterinin** altındaki metin karakterine göre yapılır.
- Kaydırma tablosu: kayd[c] = m − 1 − (c'nin P[0..m−2] içindeki en sağ konumu); yoksa **m**.
- HELLO: H → 4, E → 3, L → 1, **O → 5** (son karakter tabloya katılmaz; O başka yerde yok), diğer → 5.
- Aynı örnekte 5 hiza yeter (temel kötü karakter kuralıyla 7).

Hiza	s	Karşılaştırma	son kar.	kaydır
1	0	t[2] = '␣' ≠ p[2]	'O'	5
2	5	t[9] = '␣' ≠ p[4]	'␣'	5
3	10	t[14] = 'L' ≠ p[4]	'L'	1
4	11	t[15] = 'L' ≠ p[4]	'L'	1
5	12	tümü eşleşti	—	**bulundu**

horspool("LO LOELLO O HELLO", "HELLO")

```
1  int horspool(String t, String p) {
2    int n = t.length(), m = p.length();
3    int[] kayd = new int[256];
4    Arrays.fill(kayd, m);
5    for (int i = 0; i < m - 1; i++)
6      kayd[p.charAt(i)] = m - 1 - i;
7    int s = 0;
8    while (s <= n - m) {
9      int j = m - 1;
10     while (j >= 0 &&
11       p.charAt(j) == t.charAt(s + j))
12       j--;
13     if (j < 0) return s;
14     s += kayd[t.charAt(s + m - 1)];
15   }
16   return -1;
17 }
```

t	L	O	␣	L	O	E	L	L	O	␣	O	␣	H	E	L	L	O
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

kayd	H	E	L	O	diğer
	5	5	5	5	5

n = 17, m = 5. `kayd` dizisinin 256 hücresi önce m = 5 ile dolar: örüntüde olmayan bir karakter görülürse örüntü tümüyle onun ötesine atlar.

Adım 1 / 11

DEĞİŞKENLER

n = **17** m = **5** s = **–** j = **–**

horspool("LO LOELLO O HELLO", "HELLO")

```
1  int horspool(String t, String p) {
2    int n = t.length(), m = p.length();
3    int[] kayd = new int[256];
4    Arrays.fill(kayd, m);
▶    for (int i = 0; i < m - 1; i++)
▶      kayd[p.charAt(i)] = m - 1 - i;
7    int s = 0;
8    while (s <= n - m) {
9      int j = m - 1;
10     while (j >= 0 &&
11       p.charAt(j) == t.charAt(s + j))
12       j--;
13     if (j < 0) return s;
14     s += kayd[t.charAt(s + m - 1)];
15   }
16   return -1;
17 }
```

t	L	O	␣	L	O	E	L	L	O	␣	O	␣	H	E	L	L	O
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

	H	E	L	O	diğer
kayd	4	3	1	5	5

Son karakter hariç P[0..3] taranır: H → 4, E → 3, L önce 2 sonra **1** (en sağdaki geçiş kalır). O yalnızca sonda olduğu için 5'te kalır.

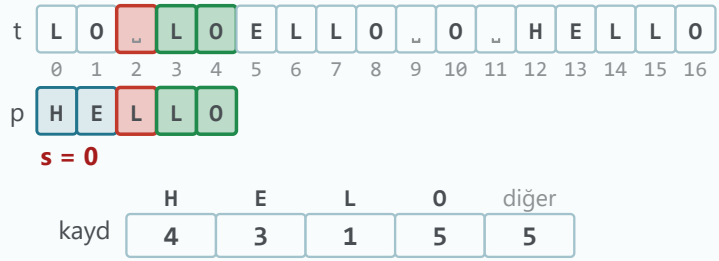
Adım 2 / 11

DEĞİŞKENLER

`n = 17` `m = 5` `s = –` `j = –`

horspool("LO LOELLO O HELLO", "HELLO")

```
1  int horspool(String t, String p) {
2    int n = t.length(), m = p.length();
3    int[] kayd = new int[256];
4    Arrays.fill(kayd, m);
5    for (int i = 0; i < m - 1; i++)
6      kayd[p.charAt(i)] = m - 1 - i;
7    int s = 0;
8    while (s <= n - m) {
▶      int j = m - 1;
▶      while (j >= 0 &&
▶        p.charAt(j) == t.charAt(s + j))
▶        j--;
13     if (j < 0) return s;
14     s += kayd[t.charAt(s + m - 1)];
15   }
16   return -1;
17 }
```



s = 0: 2 karakter sağdan eşleşti, sonra t[2] = '␣' ≠ p[2] = 'L' → uyuşmazlık.

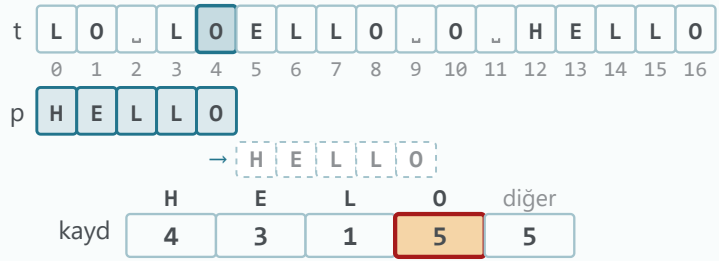
Adım 3 / 11

DEĞİŞKENLER

`n = 17` `m = 5` `s = 0` `j = 2`

horspool("LO LOELLO O HELLO", "HELLO")

```
1  int horspool(String t, String p) {
2    int n = t.length(), m = p.length();
3    int[] kayd = new int[256];
4    Arrays.fill(kayd, m);
5    for (int i = 0; i < m - 1; i++)
6      kayd[p.charAt(i)] = m - 1 - i;
7    int s = 0;
8    while (s <= n - m) {
9      int j = m - 1;
10     while (j >= 0 &&
11       p.charAt(j) == t.charAt(s + j))
12       j--;
13     if (j < 0) return s;
▶      s += kayd[t.charAt(s + m - 1)];
15   }
16   return -1;
17 }
```



Kaydırma, uyuşmazlığın yerine değil hizanın **son** karakterine bakar: t[4] = 'O' → kayd = 5 → s = 5.

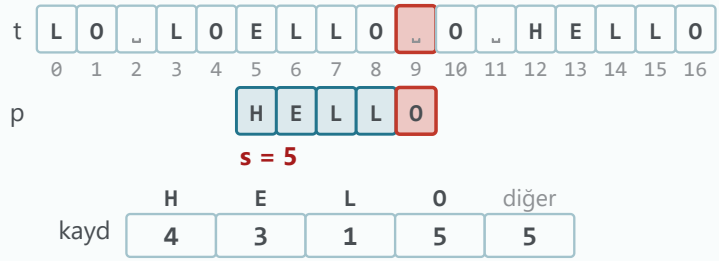
Adım 4 / 11

DEĞİŞKENLER

`n = 17` `m = 5` `s = 5` `j = 2`

horspool("LO LOELLO O HELLO", "HELLO")

```
1  int horspool(String t, String p) {
2    int n = t.length(), m = p.length();
3    int[] kayd = new int[256];
4    Arrays.fill(kayd, m);
5    for (int i = 0; i < m - 1; i++)
6      kayd[p.charAt(i)] = m - 1 - i;
7    int s = 0;
8    while (s <= n - m) {
▶      int j = m - 1;
▶      while (j >= 0 &&
▶        p.charAt(j) == t.charAt(s + j))
▶        j--;
13     if (j < 0) return s;
14     s += kayd[t.charAt(s + m - 1)];
15   }
16   return -1;
17 }
```



s = 5: t[9] = '␣' ≠ p[4] = 'O' → uyuşmazlık.

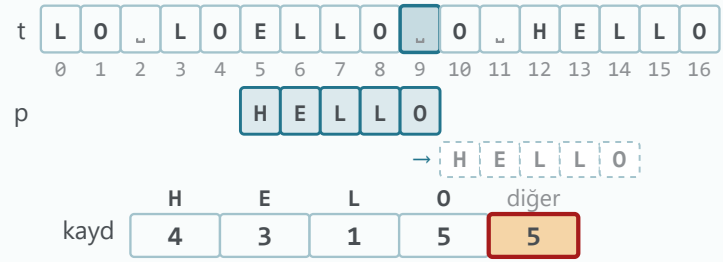
Adım 5 / 11

DEĞİŞKENLER

`n = 17` `m = 5` `s = 5` `j = 4`

horspool("LO LOELLO O HELLO", "HELLO")

```
1  int horspool(String t, String p) {
2    int n = t.length(), m = p.length();
3    int[] kayd = new int[256];
4    Arrays.fill(kayd, m);
5    for (int i = 0; i < m - 1; i++)
6      kayd[p.charAt(i)] = m - 1 - i;
7    int s = 0;
8    while (s <= n - m) {
9      int j = m - 1;
10     while (j >= 0 &&
11       p.charAt(j) == t.charAt(s + j))
12       j--;
13     if (j < 0) return s;
▶      s += kayd[t.charAt(s + m - 1)];
15   }
16   return -1;
17 }
```



Hizanın son karakteri t[9] = '␣' örüntüde yok → kayd = 5: örüntü tümüyle onun ötesine atlar, s = 10.

Adım 6 / 11

DEĞİŞKENLER

`n = 17` `m = 5` `s = 10` `j = 4`

horspool("LO LOELLO O HELLO", "HELLO")

```
1  int horspool(String t, String p) {
2    int n = t.length(), m = p.length();
3    int[] kayd = new int[256];
4    Arrays.fill(kayd, m);
5    for (int i = 0; i < m - 1; i++)
6      kayd[p.charAt(i)] = m - 1 - i;
7    int s = 0;
8    while (s <= n - m) {
▶      int j = m - 1;
▶      while (j >= 0 &&
▶        p.charAt(j) == t.charAt(s + j))
▶        j--;
13     if (j < 0) return s;
14     s += kayd[t.charAt(s + m - 1)];
15   }
16   return -1;
17 }
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
t	L	O	␣	L	O	E	L	L	O	␣	O	␣	H	E	L	L	O
p											H	E	L	L	O		

s = 10

	H	E	L	O	diğer
kayd	4	3	1	5	5

s = 10: t[14] = 'L' ≠ p[4] = 'O' → uyuşmazlık.

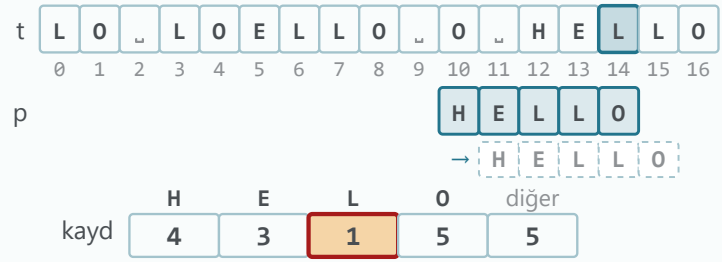
Adım 7 / 11

DEĞİŞKENLER

n = 17 | m = 5 | s = 10 | j = 4

horspool("LO LOELLO O HELLO", "HELLO")

```
1  int horspool(String t, String p) {
2    int n = t.length(), m = p.length();
3    int[] kayd = new int[256];
4    Arrays.fill(kayd, m);
5    for (int i = 0; i < m - 1; i++)
6      kayd[p.charAt(i)] = m - 1 - i;
7    int s = 0;
8    while (s <= n - m) {
9      int j = m - 1;
10     while (j >= 0 &&
11       p.charAt(j) == t.charAt(s + j))
12       j--;
13     if (j < 0) return s;
▶      s += kayd[t.charAt(s + m - 1)];
15   }
16   return -1;
17 }
```



Kaydırma, uyuşmazlığın yerine değil hizanın **son** karakterine bakar: t[14] = 'L' → kayd = 1 → s = 11.

Adım 8 / 11

DEĞİŞKENLER

`n = 17` `m = 5` `s = 11` `j = 4`

horspool("LO LOELLO O HELLO", "HELLO")

```
1  int horspool(String t, String p) {
2    int n = t.length(), m = p.length();
3    int[] kayd = new int[256];
4    Arrays.fill(kayd, m);
5    for (int i = 0; i < m - 1; i++)
6      kayd[p.charAt(i)] = m - 1 - i;
7    int s = 0;
8    while (s <= n - m) {
▶      int j = m - 1;
▶      while (j >= 0 &&
▶        p.charAt(j) == t.charAt(s + j))
▶        j--;
13     if (j < 0) return s;
14     s += kayd[t.charAt(s + m - 1)];
15   }
16   return -1;
17 }
```

t	L	O	␣	L	O	E	L	L	O	␣	O	␣	H	E	L	L	O
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

p												H	E	L	L	O

s = 11

kayd	H	E	L	O	diğer
	4	3	1	5	5

s = 11: t[15] = 'L' ≠ p[4] = 'O' → uyuşmazlık.

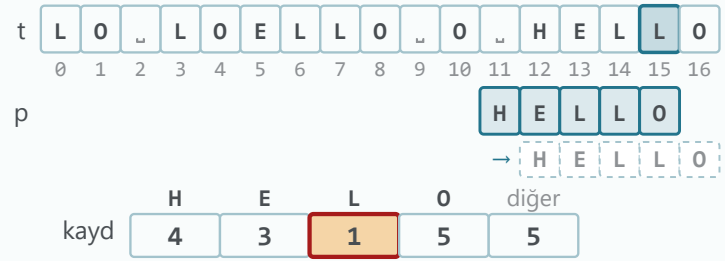
Adım 9 / 11

DEĞİŞKENLER

`n = 17` `m = 5` `s = 11` `j = 4`

horspool("LO LOELLO O HELLO", "HELLO")

```
1  int horspool(String t, String p) {
2    int n = t.length(), m = p.length();
3    int[] kayd = new int[256];
4    Arrays.fill(kayd, m);
5    for (int i = 0; i < m - 1; i++)
6      kayd[p.charAt(i)] = m - 1 - i;
7    int s = 0;
8    while (s <= n - m) {
9      int j = m - 1;
10     while (j >= 0 &&
11       p.charAt(j) == t.charAt(s + j))
12       j--;
13     if (j < 0) return s;
▶      s += kayd[t.charAt(s + m - 1)];
15   }
16   return -1;
17 }
```



Kaydırma, uyuşmazlığın yerine değil hizanın **son** karakterine bakar: t[15] = 'L' → kayd = 1 → s = 12.

Adım 10 / 11

DEĞİŞKENLER

`n = 17` `m = 5` `s = 12` `j = 4`

horspool("LO LOELLO O HELLO", "HELLO")

```
1  int horspool(String t, String p) {
2    int n = t.length(), m = p.length();
3    int[] kayd = new int[256];
4    Arrays.fill(kayd, m);
5    for (int i = 0; i < m - 1; i++)
6      kayd[p.charAt(i)] = m - 1 - i;
7    int s = 0;
8    while (s <= n - m) {
9      int j = m - 1;
10     while (j >= 0 &&
11       p.charAt(j) == t.charAt(s + j))
12       j--;
▶      if (j < 0) return s;
14     s += kayd[t.charAt(s + m - 1)];
15   }
16   return -1;
17 }
```



	H	E	L	O	diğer
kayd	4	3	1	5	5

Beş karakter sağdan sola eşleşti, j = −1 → **s = 12** döndürülür. 5 hiza, toplam 11 karşılaştırma.

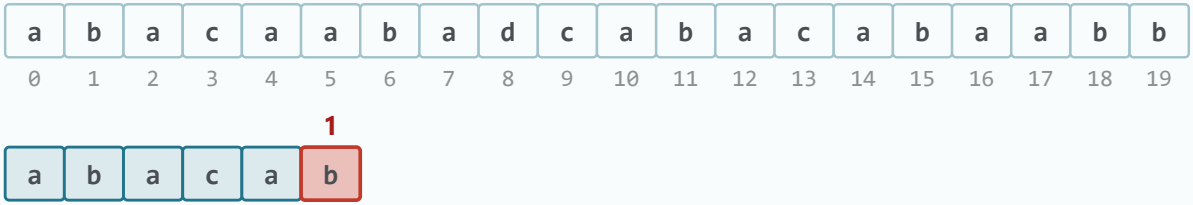
Adım 11 / 11

DEĞİŞKENLER

`n = 17` `m = 5` `s = 12` `j = −1`

Örnek: Son Görülme Tablosu ile Arama

Örüntü **abacab**, son görülme: a = 4, b = 5, c = 3, * = −1. Hücre üstündeki sayı: karşılaştırma sayacı.

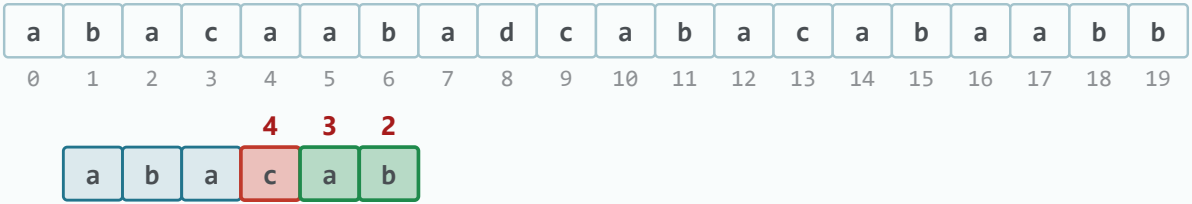


Hiza 1 (s = 0): t[5] = 'a' ≠ p[5] = 'b'. j − son['a'] = 5 − 4 = 1 → 1 kaydır.

10 hiza, toplam 20 karşılaştırma (n = 20, m = 6): 'd' gibi örüntüde olmayan bir karakter 6 konumluk atlama sağlar.

Örnek: Son Görülme Tablosu ile Arama

Örüntü **abacab**, son görülme: a = 4, b = 5, c = 3, * = −1. Hücre üstündeki sayı: karşılaştırma sayacı.

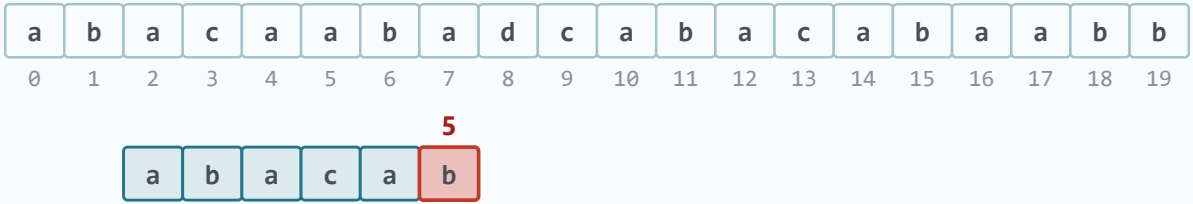


Hiza 2 (s = 1): t[4] = 'a' ≠ p[3] = 'c'. son['a'] = 4 ≥ j = 3 → 1 → 1 kaydır.

10 hiza, toplam 20 karşılaştırma (n = 20, m = 6): 'd' gibi örüntüde olmayan bir karakter 6 konumluk atlama sağlar.

Örnek: Son Görülme Tablosu ile Arama

Örüntü **abacab**, son görülme: a = 4, b = 5, c = 3, * = −1. Hücre üstündeki sayı: karşılaştırma sayacı.

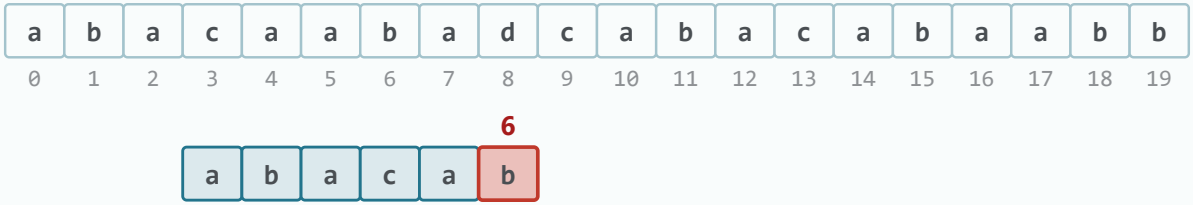


Hiza 3 (s = 2): t[7] = 'a' ≠ p[5] = 'b'. j − son['a'] = 5 − 4 = 1 → 1 kaydır.

10 hiza, toplam 20 karşılaştırma (n = 20, m = 6): 'd' gibi örüntüde olmayan bir karakter 6 konumluk atlama sağlar.

Örnek: Son Görülme Tablosu ile Arama

Örüntü **abacab**, son görülme: a = 4, b = 5, c = 3, * = −1. Hücre üstündeki sayı: karşılaştırma sayacı.

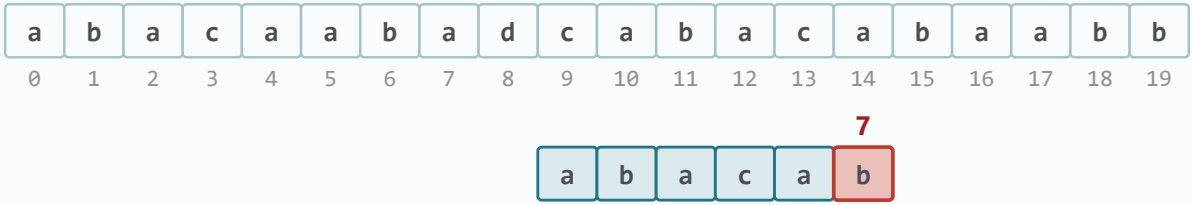


Hiza 4 (s = 3): t[8] = 'd' ≠ p[5] = 'b'. 'd' örüntüde yok → j + 1 = 6 → 6 kaydır.

10 hiza, toplam 20 karşılaştırma (n = 20, m = 6): 'd' gibi örüntüde olmayan bir karakter 6 konumluk atlama sağlar.

Örnek: Son Görülme Tablosu ile Arama

Örüntü **abacab**, son görülme: a = 4, b = 5, c = 3, * = −1. Hücre üstündeki sayı: karşılaştırma sayacı.

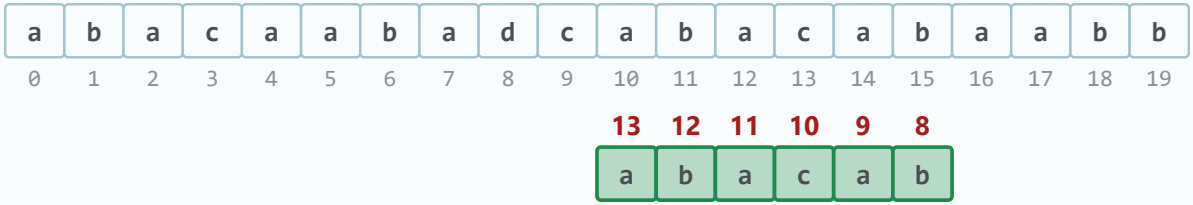


Hiza 5 (s = 9): t[14] = 'a' ≠ p[5] = 'b'. j − son['a'] = 5 − 4 = 1 → 1 kaydır.

10 hiza, toplam 20 karşılaştırma (n = 20, m = 6): 'd' gibi örüntüde olmayan bir karakter 6 konumluk atlama sağlar.

Örnek: Son Görülme Tablosu ile Arama

Örüntü **abacab**, son görülme: a = 4, b = 5, c = 3, * = −1. Hücre üstündeki sayı: karşılaştırma sayacı.



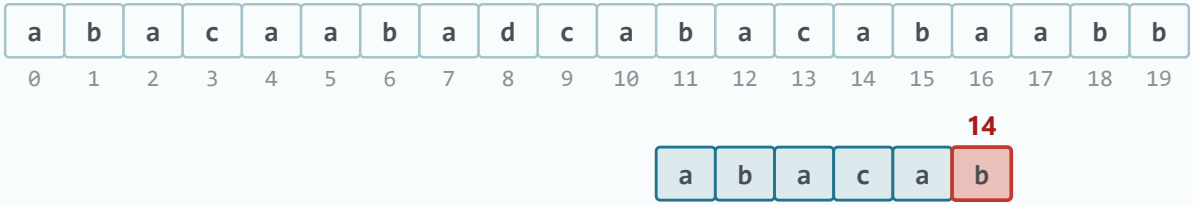
Hiza 6 (s = 10): 6 karşılaştırmanın hepsi eşleşti → **eşleşme, s = 10**. Tüm geçişler arandığı için 1 kaydırılıp devam edilir.

Adım 6 / 10

10 hiza, toplam 20 karşılaştırma (n = 20, m = 6): 'd' gibi örüntüde olmayan bir karakter 6 konumluk atlama sağlar.

Örnek: Son Görülme Tablosu ile Arama

Örüntü **abacab**, son görülme: a = 4, b = 5, c = 3, * = −1. Hücre üstündeki sayı: karşılaştırma sayacı.

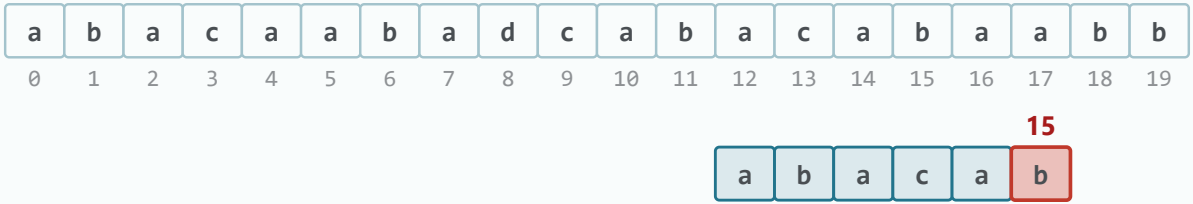


Hiza 7 (s = 11): t[16] = 'a' ≠ p[5] = 'b'. j − son['a'] = 5 − 4 = 1 → 1 kaydır.

10 hiza, toplam 20 karşılaştırma (n = 20, m = 6): 'd' gibi örüntüde olmayan bir karakter 6 konumluk atlama sağlar.

Örnek: Son Görülme Tablosu ile Arama

Örüntü **abacab**, son görülme: a = 4, b = 5, c = 3, * = −1. Hücre üstündeki sayı: karşılaştırma sayacı.

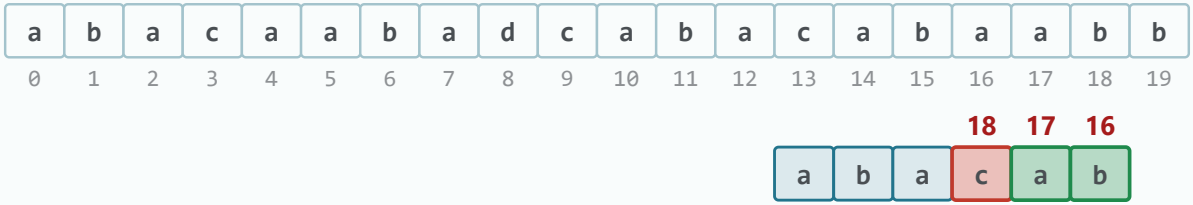


Hiza 8 (s = 12): t[17] = 'a' ≠ p[5] = 'b'. j − son['a'] = 5 − 4 = 1 → 1 kaydır.

10 hiza, toplam 20 karşılaştırma (n = 20, m = 6): 'd' gibi örüntüde olmayan bir karakter 6 konumluk atlama sağlar.

Örnek: Son Görülme Tablosu ile Arama

Örüntü **abacab**, son görülme: a = 4, b = 5, c = 3, * = −1. Hücre üstündeki sayı: karşılaştırma sayacı.

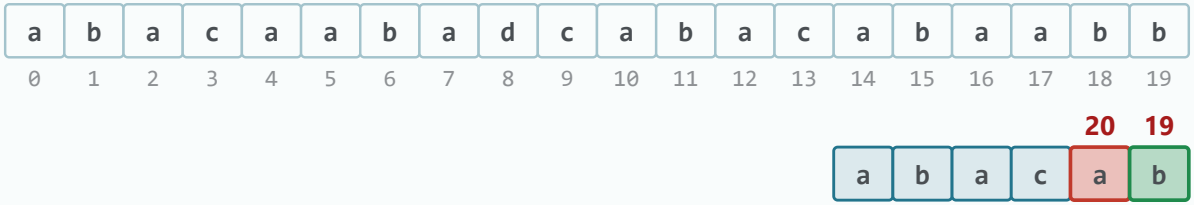


Hiza 9 (s = 13): t[16] = 'a' ≠ p[3] = 'c'. son['a'] = 4 ≥ j = 3 → 1 → 1 kaydır.

10 hiza, toplam 20 karşılaştırma (n = 20, m = 6): 'd' gibi örüntüde olmayan bir karakter 6 konumluk atlama sağlar.

Örnek: Son Görülme Tablosu ile Arama

Örüntü **abacab**, son görülme: a = 4, b = 5, c = 3, * = −1. Hücre üstündeki sayı: karşılaştırma sayacı.



Hiza 10 (s = 14): t[18] = 'b' ≠ p[4] = 'a'. son['b'] = 5 ≥ j = 4 → 1 → 1 kaydır.

Adım 10 / 10

10 hiza, toplam 20 karşılaştırma (n = 20, m = 6): 'd' gibi örüntüde olmayan bir karakter 6 konumluk atlama sağlar.

BOYER-MOORE · KODU ADIM ADIM ÇALIŞTIR

boyerMoore("merhaba naber", "abe")

```
1  int boyerMoore(String t, String p) {
▶    int n = t.length(), m = p.length();
▶    int[] son = sonGorulme(p);
▶    int s = 0;
5    while (s <= n - m) {
6      int j = m - 1;
7      while (j >= 0 &&
8        p.charAt(j) == t.charAt(s + j))
9        j--;
10     if (j < 0) return s;
11     s += Math.max(1,
12       j - son[t.charAt(s + j)]);
13   }
14   return -1;
15 }
```

t	m	e	r	h	a	b	a	␣	n	a	b	e	r
	0	1	2	3	4	5	6	7	8	9	10	11	12

son: a = 0, b = 1, e = 2, diğer = −1

n = 13, m = 3. Son görülme tablosu kuruldu; s = 0.

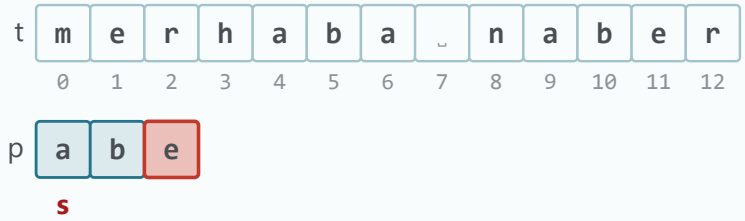
Adım 1 / 9

DEĞİŞKENLER

n = 13 | m = 3 | s = 0 | j = –

boyerMoore("merhaba naber", "abe")

```
1  int boyerMoore(String t, String p) {
2    int n = t.length(), m = p.length();
3    int[] son = sonGorulme(p);
4    int s = 0;
5    while (s <= n - m) {
6      int j = m - 1;
7      while (j >= 0 &&
8        p.charAt(j) == t.charAt(s + j))
9        j--;
10     if (j < 0) return s;
▶      s += Math.max(1,
▶        j - son[t.charAt(s + j)]);
13   }
14   return -1;
15 }
```



son: a = 0, b = 1, e = 2, diğer = −1

p[2] = 'e' ≠ t[2] = 'r'; son['r'] = -1 → s += max(1, 2 − (-1)) = 3.

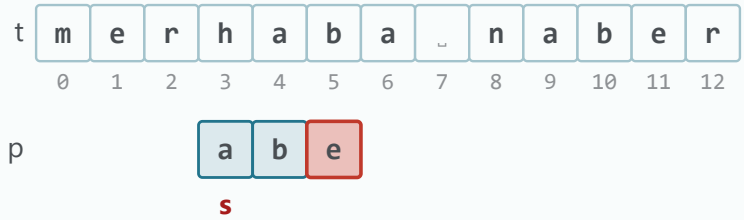
Adım 2 / 9

DEĞİŞKENLER

`n = 13` `m = 3` `s = 3` `j = 2`

boyerMoore("merhaba naber", "abe")

```
1  int boyerMoore(String t, String p) {
2    int n = t.length(), m = p.length();
3    int[] son = sonGorulme(p);
4    int s = 0;
5    while (s <= n - m) {
6      int j = m - 1;
7      while (j >= 0 &&
8        p.charAt(j) == t.charAt(s + j))
9        j--;
10     if (j < 0) return s;
▶      s += Math.max(1,
▶        j - son[t.charAt(s + j)]);
13   }
14   return -1;
15 }
```



son: a = 0, b = 1, e = 2, diğer = −1

p[2] = 'e' ≠ t[5] = 'b'; son['b'] = 1 → s += max(1, 2 − (1)) = 1.

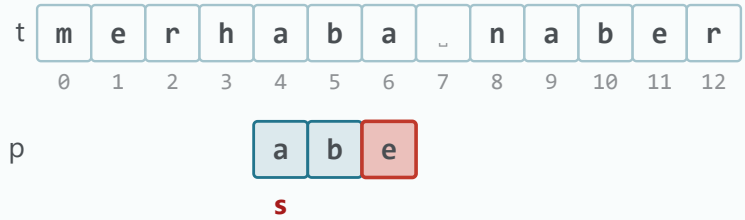
Adım 3 / 9

DEĞİŞKENLER

`n = 13` `m = 3` `s = 4` `j = 2`

boyerMoore("merhaba naber", "abe")

```
1  int boyerMoore(String t, String p) {
2    int n = t.length(), m = p.length();
3    int[] son = sonGorulme(p);
4    int s = 0;
5    while (s <= n - m) {
6      int j = m - 1;
7      while (j >= 0 &&
8        p.charAt(j) == t.charAt(s + j))
9        j--;
10     if (j < 0) return s;
▶      s += Math.max(1,
▶        j - son[t.charAt(s + j)]);
13   }
14   return -1;
15 }
```



son: a = 0, b = 1, e = 2, diğer = −1

p[2] = 'e' ≠ t[6] = 'a'; son['a'] = 0 → s += max(1, 2 − (0)) = 2.

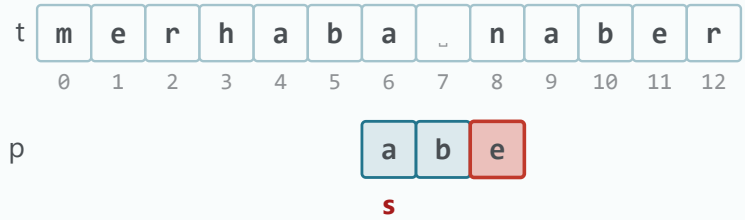
Adım 4 / 9

DEĞİŞKENLER

`n = 13` `m = 3` `s = 6` `j = 2`

boyerMoore("merhaba naber", "abe")

```
1  int boyerMoore(String t, String p) {
2    int n = t.length(), m = p.length();
3    int[] son = sonGorulme(p);
4    int s = 0;
5    while (s <= n - m) {
6      int j = m - 1;
7      while (j >= 0 &&
8        p.charAt(j) == t.charAt(s + j))
9        j--;
10     if (j < 0) return s;
▶      s += Math.max(1,
▶        j - son[t.charAt(s + j)]);
13   }
14   return -1;
15 }
```



son: a = 0, b = 1, e = 2, diğer = −1

p[2] = 'e' ≠ t[8] = 'n'; son['n'] = -1 → s += max(1, 2 − (-1)) = 3.

Adım 5 / 9

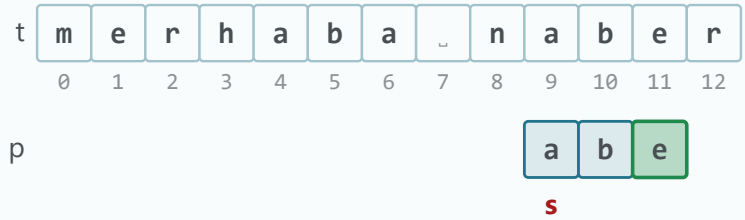
DEĞİŞKENLER

`n = 13` `m = 3` `s = 9` `j = 2`

BOYER-MOORE · KODU ADIM ADIM ÇALIŞTIR

boyerMoore("merhaba naber", "abe")

```
1  int boyerMoore(String t, String p) {
2    int n = t.length(), m = p.length();
3    int[] son = sonGorulme(p);
4    int s = 0;
5    while (s <= n - m) {
6      int j = m - 1;
▶      while (j >= 0 &&
▶        p.charAt(j) == t.charAt(s + j))
▶        j--;
10     if (j < 0) return s;
11     s += Math.max(1,
12       j - son[t.charAt(s + j)]);
13   }
14   return -1;
15 }
```



son: a = 0, b = 1, e = 2, diğer = −1

p[2] = t[11] = 'e' → j-- .

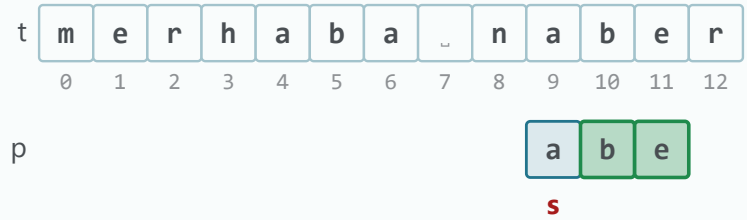
Adım 6 / 9

DEĞİŞKENLER

n = 13 | m = 3 | s = 9 | j = 1

boyerMoore("merhaba naber", "abe")

```
1  int boyerMoore(String t, String p) {
2    int n = t.length(), m = p.length();
3    int[] son = sonGorulme(p);
4    int s = 0;
5    while (s <= n - m) {
6      int j = m - 1;
▶      while (j >= 0 &&
▶        p.charAt(j) == t.charAt(s + j))
▶        j--;
10     if (j < 0) return s;
11     s += Math.max(1,
12       j - son[t.charAt(s + j)]);
13   }
14   return -1;
15 }
```



son: a = 0, b = 1, e = 2, diğer = −1

p[1] = t[10] = 'b' → j-- .

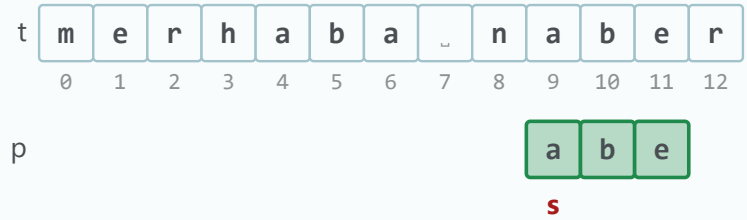
Adım 7 / 9

DEĞİŞKENLER

`n = 13` `m = 3` `s = 9` `j = 0`

boyerMoore("merhaba naber", "abe")

```
1  int boyerMoore(String t, String p) {
2    int n = t.length(), m = p.length();
3    int[] son = sonGorulme(p);
4    int s = 0;
5    while (s <= n - m) {
6      int j = m - 1;
▶      while (j >= 0 &&
▶        p.charAt(j) == t.charAt(s + j))
▶        j--;
10     if (j < 0) return s;
11     s += Math.max(1,
12       j - son[t.charAt(s + j)]);
13   }
14   return -1;
15 }
```



son: a = 0, b = 1, e = 2, diğer = −1

p[0] = t[9] = 'a' → j-- .

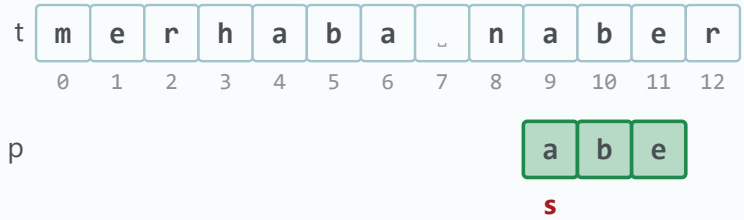
Adım 8 / 9

DEĞİŞKENLER

`n = 13` `m = 3` `s = 9` `j = −1`

boyerMoore("merhaba naber", "abe")

```
1  int boyerMoore(String t, String p) {
2    int n = t.length(), m = p.length();
3    int[] son = sonGorulme(p);
4    int s = 0;
5    while (s <= n - m) {
6      int j = m - 1;
7      while (j >= 0 &&
8        p.charAt(j) == t.charAt(s + j))
9        j--;
▶      if (j < 0) return s;
11     s += Math.max(1,
12       j - son[t.charAt(s + j)]);
13   }
14   return -1;
15 }
```



son: a = 0, b = 1, e = 2, diğer = −1

j < 0 → örüntü **s = 9** konumunda bulundu ("n**aber**").

Adım 9 / 9

DEĞİŞKENLER

`n = 13` `m = 3` `s = 9` `j = −1`

Boyer-Moore: Analiz ve Örnek

- **En iyi:** metin karakterleri örüntüde yok → her hizada 1 karşılaştırma, m atlama: ≈ n/m karşılaştırma.
- **En kötü (yalnız kötü karakter):** T = aaaa…a, P = baaa…a → her hizada m karşılaştırma, 1 kaydırma: O(n · m).
- **Örüntü uzadıkça hızlanır** (kaba kuvvet ve KMP'nin tersine).
- Küçük alfabede (DNA, ikili veri) kötü karakter kuralı zayıftır; iyi sonek kuralı önem kazanır.

Örnek

Metin: `abccbaabccbaabcbcabbabcabc`
Örüntü: `abcbcabbabcabc` (m = 14)
Son görülme tablosu: **a: 11, b: 12, c: 13**
Sonuç: örüntü **12. konumda** bulunur.

Rabin-Karp Algoritması

- Metin içinde belirli bir örüntüyü bulmak için kullanılır; **Michael O. Rabin** ve **Richard M. Karp** tarafından 1987'de geliştirilmiştir.
- Karakterleri tek tek karşılaştırmak yerine örüntünün ve her m uzunluklu metin penceresinin **hash** (özet) değerini karşılaştırır.
- **Kayan hash** (rolling hash): pencere bir sağa kayınca yeni hash, eskisinden **O(1)**'de hesaplanır.
- Hash eşitliği eşleşmeyi garanti etmez: **sahte eşleşme** (spurious hit) olabilir → karakter bazında doğrulama gerekir.

Karmaşıklık

En iyi: hash'ler nadiren çakışır → $O(n + m)$
Ortalama: pratikte $O(n + m)$
En kötü: her pencerede çakışma (sahte eşleşme), sürekli karakter kontrolü → $O(n \cdot m)$

Rabin-Karp: İşleyiş ve Kayan Hash

1. **Hash:** örüntünün ve metnin ilk penceresinin hash değerleri hesaplanır.
2. **Eşleşme kontrolü:** hash değerleri eşleşen alt dizgiler aday kabul edilir.
3. **Doğrulama:** aday pencere karakter bazında örüntüyle karşılaştırılır.
4. **Kaydırma ve yeniden hesaplama:** eşleşme yoksa pencere bir sağa kaydırılır, hash güncellenir.
5. **Tekrarlama:** tüm metin boyunca adımlar tekrarlanır.

Polinom hash (taban B, asal mod Q):
$h(t[s..s+m-1]) = (t[s]\cdot B^{m-1} + t[s+1]\cdot B^{m-2} + \ldots + t[s+m-1]) \bmod Q$
Kaydırma: $h_{s+1} = (B \cdot (h_s - t[s] \cdot B^{m-1}) + t[s+m]) \bmod Q$ — çıkan karakter düşülür, kalanlar bir basamak sola kayar, giren karakter eklenir.

Basit Örnek: Toplam Hash ve Sahte Eşleşme

Oyuncak hash: harf değerlerinin toplamı. "VUATS" metninde "TS" aranıyor.



5 + 1 = 6

örüntü "TS"

h = 2 + 3 = 5

U=1 T=2 S=3 A=4 V=5

Pencere "VU": 5 + 1 = 6 ≠ 5 → hash farklı, karakter karşılaştırması yapılmaz.

Toplam hash zayıftır (harflerin sırasına duyarsız: "ST" de 5 verir). Bu yüzden konum ağırlıklı **polinom hash** ve büyük asal Q kullanılır.

Basit Örnek: Toplam Hash ve Sahte Eşleşme

Oyuncak hash: harf değerlerinin toplamı. "VUATS" metninde "TS" aranıyor.



1 + 4 = 5

örüntü "TS"

h = 2 + 3 = 5

U=1 T=2 S=3 A=4 V=5

Pencere "UA": 1 + 4 = **5** = 5 → hash eşit ama "UA" ≠ "TS": **sahte eşleşme** (spurious hit); doğrulama boşa gider.

Toplam hash zayıftır (harflerin sırasına duyarsız: "ST" de 5 verir). Bu yüzden konum ağırlıklı **polinom hash** ve büyük asal Q kullanılır.

Basit Örnek: Toplam Hash ve Sahte Eşleşme

Oyuncak hash: harf değerlerinin toplamı. "VUATS" metninde "TS" aranıyor.



4 + 2 = 6

örüntü "TS"

h = 2 + 3 = 5

U=1 T=2 S=3 A=4 V=5

Pencere "AT": 4 + 2 = 6 ≠ 5 → geç. Yeni hash = eski − çıkan (U=1) + giren (T=2): O(1).

Toplam hash zayıftır (harflerin sırasına duyarsız: "ST" de 5 verir). Bu yüzden konum ağırlıklı **polinom hash** ve büyük asal Q kullanılır.

Basit Örnek: Toplam Hash ve Sahte Eşleşme

Oyuncak hash: harf değerlerinin toplamı. "VUATS" metninde "TS" aranıyor.



örüntü "TS"

h = 2 + 3 = 5

U=1 T=2 S=3 A=4 V=5

Pencere "TS": 2 + 3 = 5 → hash eşit, karakter karşılaştırması da tutar: **eşleşme** (s = 3).

Adım 4 / 4

Toplam hash zayıftır (harflerin sırasına duyarsız: "ST" de 5 verir). Bu yüzden konum ağırlıklı **polinom hash** ve büyük asal Q kullanılır.

rabinKarp("abracadabra", "cad")

```
1  int rabinKarp(String t, String p) {
▶    int n = t.length(), m = p.length();
▶    int B = 256, Q = 101, h = 1;
▶    for (int i = 0; i < m - 1; i++)
▶      h = (h * B) % Q;  // B^(m-1) mod Q
6    int ph = 0, th = 0;
7    for (int i = 0; i < m; i++) {
8      ph = (B * ph + p.charAt(i)) % Q;
9      th = (B * th + t.charAt(i)) % Q;
10   }
11   for (int s = 0; s <= n - m; s++) {
12     if (ph == th && esit(t, p, s))
13       return s;
14     if (s < n - m) {
15       th = (B * (th - t.charAt(s) * h)
16             + t.charAt(s + m)) % Q;
17       if (th < 0) th += Q;
18     }
19   }
20   return -1;
21 }
```

t	a	b	r	a	c	a	d	a	b	r	a
	0	1	2	3	4	5	6	7	8	9	10

a=97 b=98 c=99 d=100 r=114

n = 11, m = 3, B = 256, Q = 101. h = 256^2 mod 101 = 88 (çıkan karakterin ağırlığı).

Adım 1 / 7

DEĞİŞKENLER

`n = 11` `m = 3` `h = 88` `ph = –` `th = –` `s = –`

rabinKarp("abracadabra", "cad")

```
1  int rabinKarp(String t, String p) {
2    int n = t.length(), m = p.length();
3    int B = 256, Q = 101, h = 1;
4    for (int i = 0; i < m - 1; i++)
5      h = (h * B) % Q;  // B^(m-1) mod Q
▶    int ph = 0, th = 0;
▶    for (int i = 0; i < m; i++) {
▶      ph = (B * ph + p.charAt(i)) % Q;
▶      th = (B * th + t.charAt(i)) % Q;
10   }
11   for (int s = 0; s <= n - m; s++) {
12     if (ph == th && esit(t, p, s))
13       return s;
14     if (s < n - m) {
15       th = (B * (th - t.charAt(s) * h)
16             + t.charAt(s + m)) % Q;
17       if (th < 0) th += Q;
18     }
19   }
20   return -1;
21 }
```

t	a	b	r	a	c	a	d	a	b	r	a
	0	1	2	3	4	5	6	7	8	9	10

pencere s = 0

ph = h("cad") = 11, th = h("abr") = 4

ph = (99·256² + 97·256 + 100) mod 101 = **11**; ilk pencere "abr" için th = **4**.

Adım 2 / 7

DEĞİŞKENLER

`n = 11` `m = 3` `h = 88` `ph = 11` `th = 4` `s = –`

rabinKarp("abracadabra", "cad")

```
1  int rabinKarp(String t, String p) {
2    int n = t.length(), m = p.length();
3    int B = 256, Q = 101, h = 1;
4    for (int i = 0; i < m - 1; i++)
5      h = (h * B) % Q;  // B^(m-1) mod Q
6    int ph = 0, th = 0;
7    for (int i = 0; i < m; i++) {
8      ph = (B * ph + p.charAt(i)) % Q;
9      th = (B * th + t.charAt(i)) % Q;
10   }
11   for (int s = 0; s <= n - m; s++) {
▶      if (ph == th && esit(t, p, s))
13       return s;
▶      if (s < n - m) {
▶        th = (B * (th - t.charAt(s) * h)
▶              + t.charAt(s + m)) % Q;
▶        if (th < 0) th += Q;
18     }
19   }
20   return -1;
21 }
```

t	a	b	r	a	c	a	d	a	b	r	a
	0	1	2	3	4	5	6	7	8	9	10

pencere s = 0

th("abr") = 4 ≠ ph = 11

Pencere "abr": th = 4 ≠ 11 → karakterlere bakılmaz. Kaydır: 'a' çıkar, 'a' girer → yeni th = **30**.

Adım 3 / 7

DEĞİŞKENLER

n = 11 | m = 3 | h = 88 | ph = 11 | th = 30 | s = 0

rabinKarp("abracadabra", "cad")

```
1  int rabinKarp(String t, String p) {
2    int n = t.length(), m = p.length();
3    int B = 256, Q = 101, h = 1;
4    for (int i = 0; i < m - 1; i++)
5      h = (h * B) % Q;  // B^(m-1) mod Q
6    int ph = 0, th = 0;
7    for (int i = 0; i < m; i++) {
8      ph = (B * ph + p.charAt(i)) % Q;
9      th = (B * th + t.charAt(i)) % Q;
10   }
11   for (int s = 0; s <= n - m; s++) {
▶      if (ph == th && esit(t, p, s))
13       return s;
▶      if (s < n - m) {
▶        th = (B * (th - t.charAt(s) * h)
▶              + t.charAt(s + m)) % Q;
▶        if (th < 0) th += Q;
18     }
19   }
20   return -1;
21 }
```

t	a	b	r	a	c	a	d	a	b	r	a
	0	1	2	3	4	5	6	7	8	9	10

pencere s = 1

th("bra") = 30 ≠ ph = 11

Pencere "bra": th = 30 ≠ 11 → karakterlere bakılmaz. Kaydır: 'b' çıkar, 'c' girer → yeni th = **17**.

Adım 4 / 7

DEĞİŞKENLER

n = 11 | m = 3 | h = 88 | ph = 11 | th = 17 | s = 1

rabinKarp("abracadabra", "cad")

```
1  int rabinKarp(String t, String p) {
2    int n = t.length(), m = p.length();
3    int B = 256, Q = 101, h = 1;
4    for (int i = 0; i < m - 1; i++)
5      h = (h * B) % Q;  // B^(m-1) mod Q
6    int ph = 0, th = 0;
7    for (int i = 0; i < m; i++) {
8      ph = (B * ph + p.charAt(i)) % Q;
9      th = (B * th + t.charAt(i)) % Q;
10   }
11   for (int s = 0; s <= n - m; s++) {
▶      if (ph == th && esit(t, p, s))
13       return s;
▶      if (s < n - m) {
▶        th = (B * (th - t.charAt(s) * h)
▶              + t.charAt(s + m)) % Q;
▶        if (th < 0) th += Q;
18     }
19   }
20   return -1;
21 }
```

t	a	b	r	a	c	a	d	a	b	r	a
	0	1	2	3	4	5	6	7	8	9	10

pencere s = 2

th("rac") = 17 ≠ ph = 11

Pencere "rac": th = 17 ≠ 11 → karakterlere bakılmaz. Kaydır: 'r' çıkar, 'a' girer → yeni th = **41**.

Adım 5 / 7

DEĞİŞKENLER

n = 11 | m = 3 | h = 88 | ph = 11 | th = 41 | s = 2

rabinKarp("abracadabra", "cad")

```
 1 int rabinKarp(String t, String p) {
 2   int n = t.length(), m = p.length();
 3   int B = 256, Q = 101, h = 1;
 4   for (int i = 0; i < m - 1; i++)
 5     h = (h * B) % Q;  // B^(m-1) mod Q
 6   int ph = 0, th = 0;
 7   for (int i = 0; i < m; i++) {
 8     ph = (B * ph + p.charAt(i)) % Q;
 9     th = (B * th + t.charAt(i)) % Q;
10   }
11   for (int s = 0; s <= n - m; s++) {
▶    if (ph == th && esit(t, p, s))
13     return s;
▶    if (s < n - m) {
▶      th = (B * (th - t.charAt(s) * h)
▶            + t.charAt(s + m)) % Q;
▶      if (th < 0) th += Q;
18   }
19   }
20   return -1;
21 }
```

t	a	b	r	a	c	a	d	a	b	r	a
	0	1	2	3	4	5	6	7	8	9	10

pencere s = 3

th("aca") = 41 ≠ ph = 11

Pencere "aca": th = 41 ≠ 11 → karakterlere bakılmaz. Kaydır: 'a' çıkar, 'd' girer → yeni th = **11**.

Adım 6 / 7

DEĞİŞKENLER

`n = 11` `m = 3` `h = 88` `ph = 11` `th = 11` `s = 3`

rabinKarp("abracadabra", "cad")

```
1  int rabinKarp(String t, String p) {
2    int n = t.length(), m = p.length();
3    int B = 256, Q = 101, h = 1;
4    for (int i = 0; i < m - 1; i++)
5      h = (h * B) % Q;  // B^(m-1) mod Q
6    int ph = 0, th = 0;
7    for (int i = 0; i < m; i++) {
8      ph = (B * ph + p.charAt(i)) % Q;
9      th = (B * th + t.charAt(i)) % Q;
10   }
11   for (int s = 0; s <= n - m; s++) {
▶      if (ph == th && esit(t, p, s))
▶        return s;
14     if (s < n - m) {
15       th = (B * (th - t.charAt(s) * h)
16             + t.charAt(s + m)) % Q;
17       if (th < 0) th += Q;
18     }
19   }
20   return -1;
21 }
```

t	a	b	r	a	c	a	d	a	b	r	a
	0	1	2	3	4	5	6	7	8	9	10

pencere s = 4

th = 11 = ph → karakter karşılaştırması

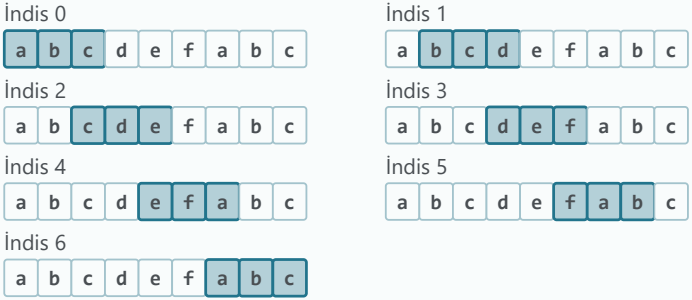
th = 11 = ph → `esit` ile karakter karşılaştırması: "cad" = "cad" → **s = 4 döndürülür**.

Adım 7 / 7

DEĞİŞKENLER

`n = 11` `m = 3` `h = 88` `ph = 11` `th = 11` `s = 4`

Kayan Pencere ve Analiz



n − m + 1 = 7 pencere; her biri bir öncekinden O(1)'de.

- Hash hesapları: O(m) + (n − m) · O(1).
- Q büyük bir asal seçilirse sahte eşleşme olasılığı ≈ 1/Q → beklenen süre **O(n + m)**.
- Kötü Q veya kötü niyetli girdi: her pencere aday → O(n · m).
- **Güçlü yanı:** aynı uzunlukta **çok sayıda örüntü** tek geçişte aranabilir (hash'ler bir kümeye konur): intihal tespiti, belge benzerliği.
- Ayrıca 2B örüntü (görüntü içinde blok) aramaya genellenebilir.

EŞLEŞTİRME ÖZETİ

Dizgi Eşleştirme Algoritmalarının Karşılaştırması

Algoritma	Temel fikir	Ön işleme	Ortalama	En kötü	Ek bellek
Kaba kuvvet	Her konumu dener	yok	O(n)*	O(n · m)	O(1)
KMP	LPS tablosu, metinde geri gitmez	O(m)	O(n + m)	O(n + m)	O(m)
Boyer-Moore	Sağdan sola, kötü karakter + iyi sonek	O(m + σ)	alt-doğrusal	O(n · m)**	O(m + σ)
Rabin-Karp	Kayan hash karşılaştırması	O(m)	O(n + m)	O(n · m)	O(1)

* rastgele metin ve geniş alfabe için; düzenli (tekrarlı) metinde O(n · m)'ye yaklaşır. ** yalnız kötü karakter kuralıyla; iyi sonek + Galil kuralıyla O(n + m). σ: alfabe boyutu.

Garanti gerekiyorsa
KMP: akış verisi, en kötü durum sınırı önemli.

Uzun örüntü, doğal dil
Boyer-Moore / Horspool: pratikte en hızlısı.

Çok sayıda örüntü
Rabin-Karp: tek geçişte birçok örüntü.

SIKIŞTIRMA

Dizgi Sıkıştırma Algoritmaları

- **Amaç:** metni (veya veriyi) daha az bitle saklamak/iletmek.
- **Kayıpsız** (lossless) sıkıştırma: açıldığında **orijinal veri aynen** elde edilir. RLE ve LZW kayıpsızdır (metin, program, PNG/GIF).
- **Kayıplı** (lossy) sıkıştırma: algıda önemsiz ayrıntılar atılır (JPEG, MP3); metin için uygun değildir.
- **Sıkıştırma oranı** = sıkıştırılmış boyut / orijinal boyut (küçük olması iyi).

RLE

Aynı değerlerin ardışık tekrarları tek değer + sayı olarak saklanır.

LZW

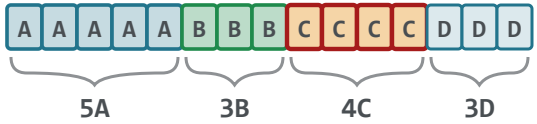
Tekrar eden örüntüler, dinamik büyüyen bir sözlükteki kısa kodlarla temsil edilir.

Çalışma Uzunluğu Kodlaması (RLE)

- Run Length Encoding: ardışık tekrar eden karakterleri (**koşu**, run) **sayı + karakter** çiftiyle değiştirir.
- **Adımlar:** metin soldan sağa taranır → ardışık tekrarlar sayılır → sayı ve karakter birleştirilir → sıkıştırılmış metin olarak saklanır.
- Metin tek bir kez tarandığı için zaman **O(n)**.
- Basit ve etkili; ancak tekrar az ise **etkisiz**, hatta zararlıdır: "ABCD" → "1A1B1C1D" (iki kat!).
- Kullanım: faks (siyah-beyaz satırlar), BMP/PCX/TIFF, basit simgeler.

Örnek

Metin: `AAAAABBBCCCCDDD` (15 karakter)



Sıkıştırılmış: `5A3B4C3D` (8 karakter)

Oran: 8/15 ≈ 0,53

RLE · KODU ADIM ADIM ÇALIŞTIR

rleSikistir("AAAAABBBCCCCDDD")

```
1  String rleSikistir(String s) {
▶    StringBuilder sonuc = new
   StringBuilder();
▶    int n = s.length();
4    for (int i = 0; i < n; i++) {
5      char c = s.charAt(i);
6      int sayac = 1;
7      while (i + 1 < n &&
8        s.charAt(i + 1) == c) {
9        sayac++;
10       i++;
11     }
12     sonuc.append(sayac).append(c);
13   }
14   return sonuc.toString();
15 }
```

A	A	A	A	A	B	B	B	C	C	C	C	D	D	D
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

n = 15, sonuc boş.

Adım 1 / 10

DEĞİŞKENLER

n = **15** | i = **0** | c = **–** | sayac = **–**

ÇIKTI

RLE · KODU ADIM ADIM ÇALIŞTIR

rleSikistir("AAAAABBBCCCCDDD")

```
1  String rleSikistir(String s) {
2    StringBuilder sonuc = new
   StringBuilder();
3    int n = s.length();
4    for (int i = 0; i < n; i++) {
▶      char c = s.charAt(i);
▶      int sayac = 1;
7      while (i + 1 < n &&
8        s.charAt(i + 1) == c) {
9        sayac++;
10       i++;
11     }
12     sonuc.append(sayac).append(c);
13   }
14   return sonuc.toString();
15 }
```

A	A	A	A	A	B	B	B	C	C	C	C	D	D	D
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

i

Yeni koşu: c = 'A', sayac = 1 (i = 0).

Adım 2 / 10

DEĞİŞKENLER

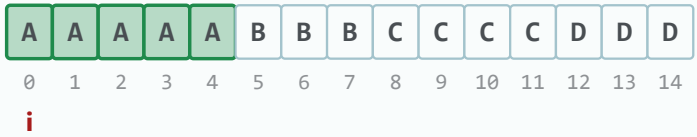
n = 15 | i = 0 | c = A | sayac = 1

ÇIKTI

RLE · KODU ADIM ADIM ÇALIŞTIR

rleSikistir("AAAAABBBCCCCDDD")

```
1  String rleSikistir(String s) {
2    StringBuilder sonuc = new
   StringBuilder();
3    int n = s.length();
4    for (int i = 0; i < n; i++) {
5      char c = s.charAt(i);
6      int sayac = 1;
▶      while (i + 1 < n &&
▶        s.charAt(i + 1) == c) {
▶        sayac++;
▶        i++;
11     }
▶      sonuc.append(sayac).append(c);
13   }
14   return sonuc.toString();
15 }
```



while döngüsü 4 kez döndü: sayac = 5, i = 4. sonuc'a "5A" eklenir.

Adım 3 / 10

DEĞİŞKENLER

n = 15 | i = 4 | c = A | sayac = 5

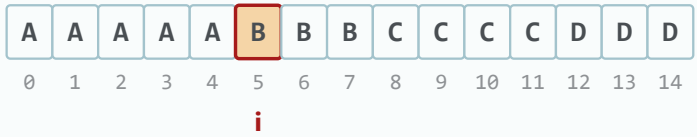
ÇIKTI

5A

RLE · KODU ADIM ADIM ÇALIŞTIR

rleSikistir("AAAAABBBCCCCDDD")

```
1  String rleSikistir(String s) {
2    StringBuilder sonuc = new
   StringBuilder();
3    int n = s.length();
4    for (int i = 0; i < n; i++) {
       char c = s.charAt(i);
       int sayac = 1;
7      while (i + 1 < n &&
8        s.charAt(i + 1) == c) {
9        sayac++;
10       i++;
11     }
12     sonuc.append(sayac).append(c);
13   }
14   return sonuc.toString();
15 }
```



Yeni koşu: c = 'B', sayac = 1 (i = 5).

Adım 4 / 10

DEĞİŞKENLER

n = 15 | i = 5 | c = B | sayac = 1

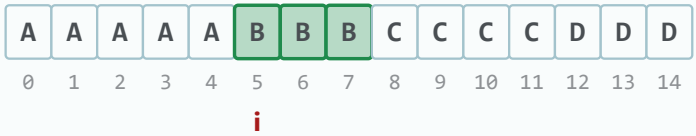
ÇIKTI

```
5A
```

RLE · KODU ADIM ADIM ÇALIŞTIR

rleSikistir("AAAAABBBCCCCDDD")

```
1  String rleSikistir(String s) {
2    StringBuilder sonuc = new
   StringBuilder();
3    int n = s.length();
4    for (int i = 0; i < n; i++) {
5      char c = s.charAt(i);
6      int sayac = 1;
▶      while (i + 1 < n &&
▶        s.charAt(i + 1) == c) {
▶        sayac++;
▶        i++;
11     }
▶      sonuc.append(sayac).append(c);
13   }
14   return sonuc.toString();
15 }
```



while döngüsü 2 kez döndü: sayac = 3, i = 7. sonuc'a "3B" eklenir.

Adım 5 / 10

DEĞİŞKENLER

n = 15 | i = 7 | c = B | sayac = 3

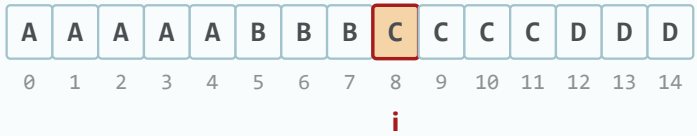
ÇIKTI

5A3B

RLE · KODU ADIM ADIM ÇALIŞTIR

rleSikistir("AAAAABBBCCCCDDD")

```
1  String rleSikistir(String s) {
2    StringBuilder sonuc = new
   StringBuilder();
3    int n = s.length();
4    for (int i = 0; i < n; i++) {
▶      char c = s.charAt(i);
▶      int sayac = 1;
7      while (i + 1 < n &&
8        s.charAt(i + 1) == c) {
9        sayac++;
10       i++;
11     }
12     sonuc.append(sayac).append(c);
13   }
14   return sonuc.toString();
15 }
```



Yeni koşu: c = 'C', sayac = 1 (i = 8).

Adım 6 / 10

DEĞİŞKENLER

n = 15 | i = 8 | c = C | sayac = 1

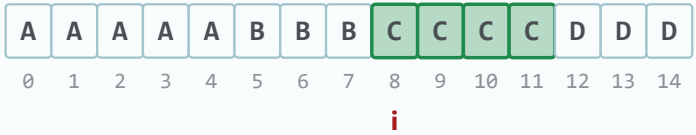
ÇIKTI

```
5A3B
```

RLE · KODU ADIM ADIM ÇALIŞTIR

rleSikistir("AAAAABBBCCCCDDD")

```
1  String rleSikistir(String s) {
2    StringBuilder sonuc = new
   StringBuilder();
3    int n = s.length();
4    for (int i = 0; i < n; i++) {
5      char c = s.charAt(i);
6      int sayac = 1;
▶      while (i + 1 < n &&
▶        s.charAt(i + 1) == c) {
▶        sayac++;
▶        i++;
11     }
▶      sonuc.append(sayac).append(c);
13   }
14   return sonuc.toString();
15 }
```



while döngüsü 3 kez döndü: sayac = 4, i = 11. sonuc'a "4C" eklenir.

Adım 7 / 10

DEĞİŞKENLER

n = 15 | i = 11 | c = C | sayac = 4

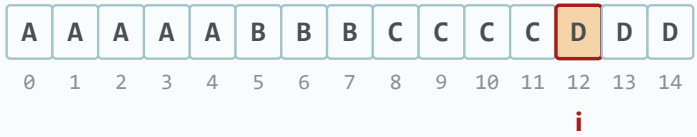
ÇIKTI

```
5A3B4C
```

RLE · KODU ADIM ADIM ÇALIŞTIR

rleSikistir("AAAAABBBCCCCDDD")

```
1  String rleSikistir(String s) {
2    StringBuilder sonuc = new
   StringBuilder();
3    int n = s.length();
4    for (int i = 0; i < n; i++) {
▶      char c = s.charAt(i);
▶      int sayac = 1;
7      while (i + 1 < n &&
8        s.charAt(i + 1) == c) {
9        sayac++;
10       i++;
11     }
12     sonuc.append(sayac).append(c);
13   }
14   return sonuc.toString();
15 }
```



Yeni koşu: c = 'D', sayac = 1 (i = 12).

Adım 8 / 10

DEĞİŞKENLER

n = 15 | i = 12 | c = D | sayac = 1

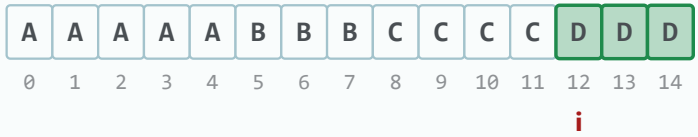
ÇIKTI

```
5A3B4C
```

RLE · KODU ADIM ADIM ÇALIŞTIR

rleSikistir("AAAAABBBCCCCDDD")

```
1  String rleSikistir(String s) {
2    StringBuilder sonuc = new
   StringBuilder();
3    int n = s.length();
4    for (int i = 0; i < n; i++) {
5      char c = s.charAt(i);
6      int sayac = 1;
▶      while (i + 1 < n &&
▶        s.charAt(i + 1) == c) {
▶        sayac++;
▶        i++;
11     }
▶      sonuc.append(sayac).append(c);
13   }
14   return sonuc.toString();
15 }
```



while döngüsü 2 kez döndü: sayac = 3, i = 14. sonuc'a "3D" eklenir.

Adım 9 / 10

DEĞİŞKENLER

n = 15 | i = 14 | c = D | sayac = 3

ÇIKTI

```
5A3B4C3D
```

RLE · KODU ADIM ADIM ÇALIŞTIR

rleSikistir("AAAAABBBCCCCDDD")

```
1  String rleSikistir(String s) {
2    StringBuilder sonuc = new
   StringBuilder();
3    int n = s.length();
4    for (int i = 0; i < n; i++) {
5      char c = s.charAt(i);
6      int sayac = 1;
7      while (i + 1 < n &&
8        s.charAt(i + 1) == c) {
9        sayac++;
10       i++;
11     }
12     sonuc.append(sayac).append(c);
13   }
▶    return sonuc.toString();
15 }
```

A	A	A	A	A	B	B	B	C	C	C	C	D	D	D
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Sonuç: **"5A3B4C3D"** — 15 karakter yerine 8.

Adım 10 / 10

DEĞİŞKENLER

`n = 15` `i = 15` `c = D` `sayac = 3`

ÇIKTI

```
5A3B4C3D
```

RLE

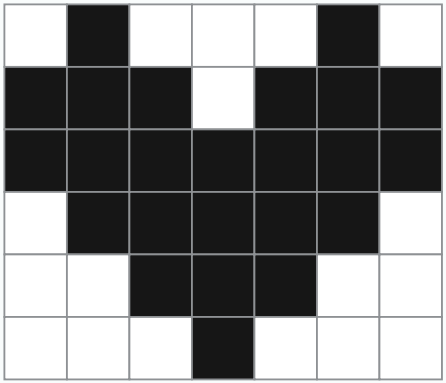
RLE'nin Bayt Düzeyinde Kodlanması

Metin: `aaaaaaaaaabbbbbbeceeceeceeceecddddddddddddddddecb` (**46 bayt**)

Sayı–karakter çiftleri	`(10, a)(6, b)(1, e)(1, c)(1, e)(1, c)(1, e)(1, c)(1, e)(1, c)(1, e)(1, c)(1, e)(1, c)(15, d)(1, e)(1, c)(1, b)`
Bayraksız RLE (her koşu 2 bayt)	`10 97 06 98 01 101 01 99 01 101 01 99 01 101 01 99 01 101 01 99 01 101 01 99 01 101 01 99 15 100 01 101 01 99 01 98` → **36 bayt**
Bayraklı RLE (bayrak = 255)	`255 10 97 255 06 98 101 99 101 99 101 99 101 99 101 99 101 99 255 15 100 101 99 98` → **24 bayt**

Bayraksız RLE'de "ececec…" gibi tekrarsız bölgeler her karakteri 2 bayta çıkarır. Bayraklı RLE yalnızca **uzun koşuları** `255 sayı karakter` üçlüsüyle yazar, diğer karakterleri olduğu gibi bırakır (255 değerinin kendisi de kaçış ile yazılmalıdır).

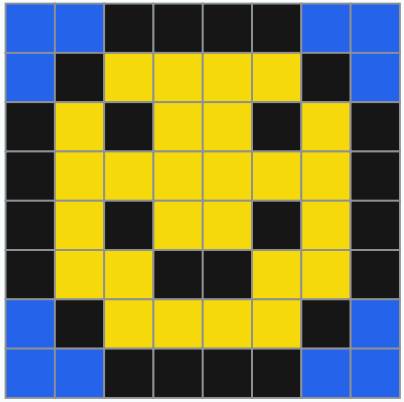
İkili (Siyah-Beyaz) Görüntüde RLE



```
W1B1W3B1W1
B3W1B3
B7
W1B5W1
W2B3W2
W3B1W3
```

- Her satır soldan sağa taranır: W = beyaz, B = siyah, ardından koşu uzunluğu.
- 42 piksel → 18 koşu (her satır ayrı kodlanır). Yalnızca iki renk olduğu için renk adı bile yazılmayabilir: koşular dönüşümlü olarak beyaz/siyah kabul edilir (faks standardı bu fikri kullanır).
- Büyük düz alanlı görüntülerde (çizim, metin taraması) çok etkilidir; fotoğraflarda etkisizdir.

Piksel Düzeyinde RLE (RGB)



Sayı	R G B	Sayı	R G B	Sayı	R G B
2	0 0 255	4	0 0 0	3	0 0 255
1	0 0 0	4	255 255 0	1	0 0 0
1	0 0 255	1	0 0 0	1	255 255 0
1	0 0 0	2	255 255 0	1	0 0 0
1	255 255 0	2	0 0 0	6	255 255 0
2	0 0 0	1	255 255 0	1	0 0 0
2	255 255 0	1	0 0 0	1	255 255 0
2	0 0 0	2	255 255 0	2	0 0 0
2	255 255 0	1	0 0 0	1	0 0 255
1	0 0 0	4	255 255 0	1	0 0 0
3	0 0 255	4	0 0 0	2	0 0 255

Mavi (0, 0, 255), siyah (0, 0, 0), sarı (255, 255, 0). Ham: 64 × 3 = **192 bayt**. RLE: 33 koşu × (1 sayı + 3 renk) = **132 bayt**.

Tablo satır satır (soldan sağa, sonra alt satır) okunur; satır sonunda aynı renk devam ediyorsa koşu bir sonraki satıra taşar (ör. 3 mavi).

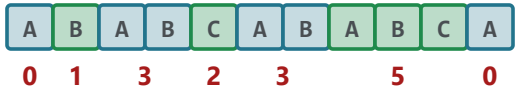
Lempel-Ziv-Welch (LZW) Sıkıştırma

- **Sözlük tabanlı** kayıpsız sıkıştırma: Lempel ve Ziv'in LZ78 yöntemini (1978) Terry Welch geliştirmiştir (1984).
- Tekrar eden dizgileri tanımlar ve onları sözlükteki **kısa kodlarla** (indis) temsil eder.
- Sözlük **dinamik**tir: sıkıştırırken büyür; **gönderilmez**, açıcı aynı sözlüğü kendisi yeniden kurar.
- GIF, TIFF, PDF ve Unix `compress` gibi biçimlerde kullanılır.
- Sözlük büyüdükçe kod genişliği (bit) ve bellek artar, performans düşer: pratikte sözlük sınırlanır (GIF: 4096 kod = 12 bit) ve dolunca sıfırlanır.

Örnek

Metin: `ABABCABABCA`

Başlangıç sözlüğü: A = 0, B = 1, C = 2



Sıkıştırılmış: **0 1 3 2 3 5 0**

11 sembol yerine 7 kod.

LZW · KODU ADIM ADIM ÇALIŞTIR

lzw("ABABCABABCA")

```
1  List<Integer> lzw(String s) {
▶    Map<String, Integer> sozluk =
▶        ilkSozluk(s);  // A=0, B=1, C=2
▶    String w = "";
▶    List<Integer> kod = new ArrayList<>();
6    for (char c : s.toCharArray()) {
7      String wc = w + c;
8      if (sozluk.containsKey(wc)) {
9        w = wc;
10     } else {
11       kod.add(sozluk.get(w));
12       sozluk.put(wc, sozluk.size());
13       w = "" + c;
14     }
15   }
16   if (!w.isEmpty())
17     kod.add(sozluk.get(w));
18   return kod;
19 }
```

A	B	A	B	C	A	B	A	B	C	A
0	1	2	3	4	5	6	7	8	9	10

0	A
1	B
2	C

Sözlük metindeki karakterlerle başlar: A = 0, B = 1, C = 2. w = "".

Adım 1 / 13

DEĞİŞKENLER

c = – w = "" wc = –

ÇIKTI

lzw("ABABCABABCA")

```
1  List<Integer> lzw(String s) {
2    Map<String, Integer> sozluk =
3        ilkSozluk(s);  // A=0, B=1, C=2
4    String w = "";
5    List<Integer> kod = new ArrayList<>();
6    for (char c : s.toCharArray()) {
▶      String wc = w + c;
▶      if (sozluk.containsKey(wc)) {
▶        w = wc;
10     } else {
11       kod.add(sozluk.get(w));
12       sozluk.put(wc, sozluk.size());
13       w = "" + c;
14     }
15   }
16   if (!w.isEmpty())
17     kod.add(sozluk.get(w));
18   return kod;
19 }
```

A	B	A	B	C	A	B	A	B	C	A
0	1	2	3	4	5	6	7	8	9	10

c

0	A
1	B
2	C

"A" sözlükte (0) → w = "A"; daha uzun bir eşleşme aranır.

Adım 2 / 13

DEĞİŞKENLER

c = A w = "A" wc = "A"

ÇIKTI

lzw("ABABCABABCA")

```
1  List<Integer> lzw(String s) {
2    Map<String, Integer> sozluk =
3        ilkSozluk(s);  // A=0, B=1, C=2
4    String w = "";
5    List<Integer> kod = new ArrayList<>();
6    for (char c : s.toCharArray()) {
7      String wc = w + c;
8      if (sozluk.containsKey(wc)) {
9        w = wc;
10     } else {
▶        kod.add(sozluk.get(w));
▶        sozluk.put(wc, sozluk.size());
▶        w = "" + c;
14     }
15   }
16   if (!w.isEmpty())
17     kod.add(sozluk.get(w));
18   return kod;
19 }
```

A	B	A	B	C	A	B	A	B	C	A
0	1	2	3	4	5	6	7	8	9	10

c

0	A
1	B
2	C
3	AB

"AB" sözlükte yok → w = "A" kodu **0** çıktıya; "AB" = 3 sözlüğe eklenir; w = "B".

Adım 3 / 13

DEĞİŞKENLER

`c = B` `w = "B"` `wc = "AB"`

ÇIKTI

```
0
```

lzw("ABABCABABCA")

```
1  List<Integer> lzw(String s) {
2    Map<String, Integer> sozluk =
3        ilkSozluk(s);  // A=0, B=1, C=2
4    String w = "";
5    List<Integer> kod = new ArrayList<>();
6    for (char c : s.toCharArray()) {
7      String wc = w + c;
8      if (sozluk.containsKey(wc)) {
9        w = wc;
10     } else {
▶        kod.add(sozluk.get(w));
▶        sozluk.put(wc, sozluk.size());
▶        w = "" + c;
14     }
15   }
16   if (!w.isEmpty())
17     kod.add(sozluk.get(w));
18   return kod;
19 }
```

A	B	A	B	C	A	B	A	B	C	A
0	1	2	3	4	5	6	7	8	9	10

c

0	A
1	B
2	C
3	AB
4	BA

"BA" sözlükte yok → w = "B" kodu **1** çıktıya; "BA" = 4 sözlüğe eklenir; w = "A".

Adım 4 / 13

DEĞİŞKENLER

`c = A` `w = "A"` `wc = "BA"`

ÇIKTI

```
0 1
```

lzw("ABABCABABCA")

```
1  List<Integer> lzw(String s) {
2    Map<String, Integer> sozluk =
3        ilkSozluk(s);  // A=0, B=1, C=2
4    String w = "";
5    List<Integer> kod = new ArrayList<>();
6    for (char c : s.toCharArray()) {
▶      String wc = w + c;
▶      if (sozluk.containsKey(wc)) {
▶        w = wc;
10     } else {
11       kod.add(sozluk.get(w));
12       sozluk.put(wc, sozluk.size());
13       w = "" + c;
14     }
15   }
16   if (!w.isEmpty())
17     kod.add(sozluk.get(w));
18   return kod;
19 }
```

A	B	A	B	C	A	B	A	B	C	A
0	1	2	3	4	5	6	7	8	9	10

c

0	A
1	B
2	C
3	AB
4	BA

"AB" sözlükte (3) → w = "AB"; daha uzun bir eşleşme aranır.

Adım 5 / 13

DEĞİŞKENLER

`c = B` `w = "AB"` `wc = "AB"`

ÇIKTI

```
0 1
```

lzw("ABABCABABCA")

```
1  List<Integer> lzw(String s) {
2    Map<String, Integer> sozluk =
3        ilkSozluk(s);  // A=0, B=1, C=2
4    String w = "";
5    List<Integer> kod = new ArrayList<>();
6    for (char c : s.toCharArray()) {
7      String wc = w + c;
8      if (sozluk.containsKey(wc)) {
9        w = wc;
10     } else {
▶        kod.add(sozluk.get(w));
▶        sozluk.put(wc, sozluk.size());
▶        w = "" + c;
14     }
15   }
16   if (!w.isEmpty())
17     kod.add(sozluk.get(w));
18   return kod;
19 }
```

A	B	A	B	C	A	B	A	B	C	A
0	1	2	3	4	5	6	7	8	9	10

c

0	A	5	ABC
1	B		
2	C		
3	AB		
4	BA		

"ABC" sözlükte yok → w = "AB" kodu **3** çıktıya; "ABC" = 5 sözlüğe eklenir; w = "C".

Adım 6 / 13

DEĞİŞKENLER

`c = C` `w = "C"` `wc = "ABC"`

ÇIKTI

```
0 1 3
```

lzw("ABABCABABCA")

```
1  List<Integer> lzw(String s) {
2    Map<String, Integer> sozluk =
3        ilkSozluk(s);  // A=0, B=1, C=2
4    String w = "";
5    List<Integer> kod = new ArrayList<>();
6    for (char c : s.toCharArray()) {
7      String wc = w + c;
8      if (sozluk.containsKey(wc)) {
9        w = wc;
10     } else {
▶        kod.add(sozluk.get(w));
▶        sozluk.put(wc, sozluk.size());
▶        w = "" + c;
14     }
15   }
16   if (!w.isEmpty())
17     kod.add(sozluk.get(w));
18   return kod;
19 }
```

A	B	A	B	C	A	B	A	B	C	A
0	1	2	3	4	5	6	7	8	9	10

c

0	A	5	ABC
1	B	6	CA
2	C		
3	AB		
4	BA		

"CA" sözlükte yok → w = "C" kodu **2** çıktıya; "CA" = 6 sözlüğe eklenir; w = "A".

Adım 7 / 13

DEĞİŞKENLER

`c = A` `w = "A"` `wc = "CA"`

ÇIKTI

```
0 1 3 2
```

lzw("ABABCABABCA")

```
1  List<Integer> lzw(String s) {
2    Map<String, Integer> sozluk =
3        ilkSozluk(s);  // A=0, B=1, C=2
4    String w = "";
5    List<Integer> kod = new ArrayList<>();
6    for (char c : s.toCharArray()) {
▶      String wc = w + c;
▶      if (sozluk.containsKey(wc)) {
▶        w = wc;
10     } else {
11       kod.add(sozluk.get(w));
12       sozluk.put(wc, sozluk.size());
13       w = "" + c;
14     }
15   }
16   if (!w.isEmpty())
17     kod.add(sozluk.get(w));
18   return kod;
19 }
```

A	B	A	B	C	A	B	A	B	C	A
0	1	2	3	4	5	6	7	8	9	10

c

0	A	5	ABC
1	B	6	CA
2	C		
3	AB		
4	BA		

"AB" sözlükte (3) → w = "AB"; daha uzun bir eşleşme aranır.

Adım 8 / 13

DEĞİŞKENLER

`c = B` `w = "AB"` `wc = "AB"`

ÇIKTI

```
0 1 3 2
```

lzw("ABABCABABCA")

```
1  List<Integer> lzw(String s) {
2    Map<String, Integer> sozluk =
3        ilkSozluk(s);  // A=0, B=1, C=2
4    String w = "";
5    List<Integer> kod = new ArrayList<>();
6    for (char c : s.toCharArray()) {
7      String wc = w + c;
8      if (sozluk.containsKey(wc)) {
9        w = wc;
10     } else {
▶        kod.add(sozluk.get(w));
▶        sozluk.put(wc, sozluk.size());
▶        w = "" + c;
14     }
15   }
16   if (!w.isEmpty())
17     kod.add(sozluk.get(w));
18   return kod;
19 }
```

A	B	A	B	C	A	B	A	B	C	A
0	1	2	3	4	5	6	7	8	9	10

c

0	A	5	ABC
1	B	6	CA
2	C	7	ABA
3	AB		
4	BA		

"ABA" sözlükte yok → w = "AB" kodu **3** çıktıya; "ABA" = 7 sözlüğe eklenir; w = "A".

Adım 9 / 13

DEĞİŞKENLER

`c = A` `w = "A"` `wc = "ABA"`

ÇIKTI

```
0 1 3 2 3
```

lzw("ABABCABABCA")

```
1  List<Integer> lzw(String s) {
2    Map<String, Integer> sozluk =
3        ilkSozluk(s);  // A=0, B=1, C=2
4    String w = "";
5    List<Integer> kod = new ArrayList<>();
6    for (char c : s.toCharArray()) {
▶      String wc = w + c;
▶      if (sozluk.containsKey(wc)) {
▶        w = wc;
10     } else {
11       kod.add(sozluk.get(w));
12       sozluk.put(wc, sozluk.size());
13       w = "" + c;
14     }
15   }
16   if (!w.isEmpty())
17     kod.add(sozluk.get(w));
18   return kod;
19 }
```

A	B	A	B	C	A	B	A	B	C	A
0	1	2	3	4	5	6	7	8	9	10

c

0	A	5	ABC
1	B	6	CA
2	C	7	ABA
3	AB		
4	BA		

"AB" sözlükte (3) → w = "AB"; daha uzun bir eşleşme aranır.

Adım 10 / 13

DEĞİŞKENLER

`c = B` `w = "AB"` `wc = "AB"`

ÇIKTI

```
0 1 3 2 3
```

lzw("ABABCABABCA")

```
1  List<Integer> lzw(String s) {
2    Map<String, Integer> sozluk =
3        ilkSozluk(s);  // A=0, B=1, C=2
4    String w = "";
5    List<Integer> kod = new ArrayList<>();
6    for (char c : s.toCharArray()) {
▶      String wc = w + c;
▶      if (sozluk.containsKey(wc)) {
▶        w = wc;
10     } else {
11       kod.add(sozluk.get(w));
12       sozluk.put(wc, sozluk.size());
13       w = "" + c;
14     }
15   }
16   if (!w.isEmpty())
17     kod.add(sozluk.get(w));
18   return kod;
19 }
```

A	B	A	B	C	A	B	A	B	C	A
0	1	2	3	4	5	6	7	8	9	10
									c	

0	A	5	ABC
1	B	6	CA
2	C	7	ABA
3	AB		
4	BA		

"ABC" sözlükte (5) → w = "ABC"; daha uzun bir eşleşme aranır.

Adım 11 / 13

DEĞİŞKENLER

`c = C` `w = "ABC"` `wc = "ABC"`

ÇIKTI

```
0 1 3 2 3
```

lzw("ABABCABABCA")

```
1  List<Integer> lzw(String s) {
2    Map<String, Integer> sozluk =
3        ilkSozluk(s);  // A=0, B=1, C=2
4    String w = "";
5    List<Integer> kod = new ArrayList<>();
6    for (char c : s.toCharArray()) {
7      String wc = w + c;
8      if (sozluk.containsKey(wc)) {
9        w = wc;
10     } else {
▶        kod.add(sozluk.get(w));
▶        sozluk.put(wc, sozluk.size());
▶        w = "" + c;
14     }
15   }
16   if (!w.isEmpty())
17     kod.add(sozluk.get(w));
18   return kod;
19 }
```

A	B	A	B	C	A	B	A	B	C	A
0	1	2	3	4	5	6	7	8	9	10
										c

0	A	5	ABC
1	B	6	CA
2	C	7	ABA
3	AB	8	ABCA
4	BA		

"ABCA" sözlükte yok → w = "ABC" kodu **5** çıktıya; "ABCA" = 8 sözlüğe eklenir; w = "A".

Adım 12 / 13

DEĞİŞKENLER

`c = A` `w = "A"` `wc = "ABCA"`

ÇIKTI

```
0 1 3 2 3 5
```

lzw("ABABCABABCA")

```
1  List<Integer> lzw(String s) {
2    Map<String, Integer> sozluk =
3        ilkSozluk(s);  // A=0, B=1, C=2
4    String w = "";
5    List<Integer> kod = new ArrayList<>();
6    for (char c : s.toCharArray()) {
7      String wc = w + c;
8      if (sozluk.containsKey(wc)) {
9        w = wc;
10     } else {
11       kod.add(sozluk.get(w));
12       sozluk.put(wc, sozluk.size());
13       w = "" + c;
14     }
15   }
▶  if (!w.isEmpty())
▶    kod.add(sozluk.get(w));
18   return kod;
19 }
```

A	B	A	B	C	A	B	A	B	C	A
0	1	2	3	4	5	6	7	8	9	10

c

0	A	5	ABC
1	B	6	CA
2	C	7	ABA
3	AB	8	ABCA
4	BA		

Döngü bitti: kalan w = "A" kodu 0 eklenir. Çıktı: **0 1 3 2 3 5 0**.

Adım 13 / 13

DEĞİŞKENLER

`c = A` `w = "A"` `wc = –`

ÇIKTI

```
0 1 3 2 3 5 0
```

LZW Açma (Decompression)

- Açıcı yalnızca **başlangıç sözlüğünü** bilir; sözlüğü sıkıştırıcıyla aynı sırada kendisi kurar.
- Her yeni kod k için: e = sözlük[k] yazılır; sözlüğe **w + e[0]** eklenir; w = e.
- Açıcı sözlükte **bir adım geride** kalır. k henüz tanımlı değilse (k = sonraki kod), e = **w + w[0]** olmak zorundadır (cScSc durumu).
- Çıktı: A·B·AB·C·AB·ABC·A = **ABABCABABCA** ✓

Kod	Çıktı	Eklenen
0	A	—
1	B	3 = AB
3	AB	4 = BA
2	C	5 = ABC
3	AB	6 = CA
5	ABC	7 = ABA
0	A	8 = ABCA

LZW Örneği: DNA Dizisi

Metin: `ATGATCATGAG` (11 sembol)

Başlangıç: **A = 0, T = 1, G = 2, C = 3**

Çıktı: **0 1 2 4 3 4 6 2** (8 kod)

- Her satır: sözlüğe eklenen yeni dizgi ve o anda çıktıya yazılan kod.
- "AT" ikinci kez görüldüğünde tek kodla (4) yazılır; "GA" (6) da öyle.
- Son w = "G" döngüden sonra çıktıya eklenir (2).

Kod	Sözlüğe eklenen	Çıktı
4	`AT`	0
5	`TG`	1
6	`GA`	2
7	`ATC`	4
8	`CA`	3
9	`ATG`	4
10	`GAG`	6
—	(son w = G)	2

ASCII Sözlüklü LZW ve Özel Durum

- Pratikte başlangıç sözlüğü 256 ASCII karakteridir (a = 97, b = 98, c = 99); yeni dizgiler **256**'dan itibaren numaralanır.
- Metin: `aabaacababacb` (13 karakter) → 9 kod: **97 97 98 256 99 257 261 99 98**.
- "aba" = 261, "ab" (257) yazılırken eklenir ve **hemen bir sonraki kodda** kullanılır.
- Açıcı 261'i okuduğunda sözlüğünde henüz 260'a kadar kod vardır → **özel durum**: e = w + w[0] = "ab" + "a" = "aba".

Çıktı	Sözlüğe eklenen
97	256 = `aa`
97	257 = `ab`
98	258 = `ba`
256	259 = `aac`
99	260 = `ca`
257	261 = `aba`
261	262 = `abac`
99	263 = `cb`
98	(son w = b)

LZW · KODU ADIM ADIM ÇALIŞTIR

lzwAc([97 97 98 256 99 257 261 99 98])

```
1  String lzwAc(List<Integer> kod) {
▶    List<String> sozluk =
▶        ilkSozluk();      // 0..255: ASCII
▶    String w = sozluk.get(kod.get(0));
▶    StringBuilder s = new StringBuilder(w);
6    for (int i = 1; i < kod.size(); i++) {
7      int k = kod.get(i);
8      String e = k < sozluk.size()
9          ? sozluk.get(k)
10         : w + w.charAt(0); // özel durum
11     s.append(e);
12     sozluk.add(w + e.charAt(0));
13     w = e;
14   }
15   return s.toString();
16 }
```



sözlük 256'dan sonrası henüz boş

Açıcı yalnızca 256 ASCII karakterini bilir. İlk kod 97 her zaman tek karakterdir: w = "a", çıktı "a". Sözlükte henüz 256 ve sonrası yok.

Adım 1 / 10

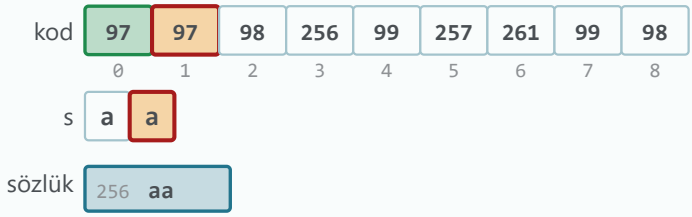
DEĞİŞKENLER

i = – k = – w = "a" e = –

LZW · KODU ADIM ADIM ÇALIŞTIR

lzwAc([97 97 98 256 99 257 261 99 98])

```
1  String lzwAc(List<Integer> kod) {
2    List<String> sozluk =
3        ilkSozluk();      // 0..255: ASCII
4    String w = sozluk.get(kod.get(0));
5    StringBuilder s = new StringBuilder(w);
6    for (int i = 1; i < kod.size(); i++) {
       int k = kod.get(i);
       String e = k < sozluk.size()
           ? sozluk.get(k)
10         : w + w.charAt(0); // özel durum
       s.append(e);
       sozluk.add(w + e.charAt(0));
       w = e;
14   }
15   return s.toString();
16 }
```



k = 97 (ASCII) → e = "a" yazılır; sözlüğe w + e[0] = "a" + "a" = **256 = "aa"** eklenir.

Adım 2 / 10

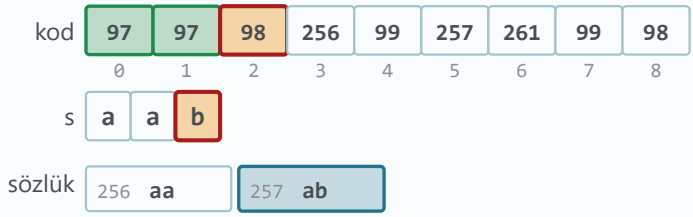
DEĞİŞKENLER

i = 1 | k = 97 | w = "a" | e = "a"

LZW · KODU ADIM ADIM ÇALIŞTIR

lzwAc([97 97 98 256 99 257 261 99 98])

```
1  String lzwAc(List<Integer> kod) {
2    List<String> sozluk =
3        ilkSozluk();      // 0..255: ASCII
4    String w = sozluk.get(kod.get(0));
5    StringBuilder s = new StringBuilder(w);
6    for (int i = 1; i < kod.size(); i++) {
       int k = kod.get(i);
       String e = k < sozluk.size()
           ? sozluk.get(k)
10         : w + w.charAt(0); // özel durum
       s.append(e);
       sozluk.add(w + e.charAt(0));
       w = e;
14   }
15   return s.toString();
16 }
```



k = 98 (ASCII) → e = "b" yazılır; sözlüğe w + e[0] = "a" + "b" = **257 = "ab"** eklenir.

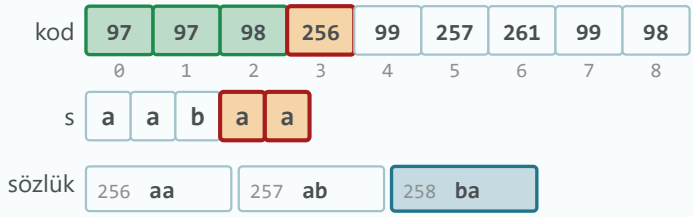
Adım 3 / 10

DEĞİŞKENLER

`i = 2` `k = 98` `w = "b"` `e = "b"`

lzwAc([97 97 98 256 99 257 261 99 98])

```
1  String lzwAc(List<Integer> kod) {
2    List<String> sozluk =
3        ilkSozluk();      // 0..255: ASCII
4    String w = sozluk.get(kod.get(0));
5    StringBuilder s = new StringBuilder(w);
6    for (int i = 1; i < kod.size(); i++) {
       int k = kod.get(i);
       String e = k < sozluk.size()
           ? sozluk.get(k)
10         : w + w.charAt(0); // özel durum
       s.append(e);
       sozluk.add(w + e.charAt(0));
       w = e;
14   }
15   return s.toString();
16 }
```



k = 256 (sözlükte 256 = "aa") → e = "aa" yazılır; sözlüğe w + e[0] = "b" + "a" = **258 = "ba"** eklenir.

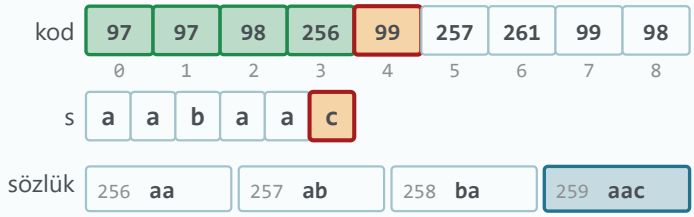
Adım 4 / 10

DEĞİŞKENLER

`i = 3` `k = 256` `w = "aa"` `e = "aa"`

lzwAc([97 97 98 256 99 257 261 99 98])

```
1  String lzwAc(List<Integer> kod) {
2    List<String> sozluk =
3        ilkSozluk();      // 0..255: ASCII
4    String w = sozluk.get(kod.get(0));
5    StringBuilder s = new StringBuilder(w);
6    for (int i = 1; i < kod.size(); i++) {
▶      int k = kod.get(i);
▶      String e = k < sozluk.size()
▶          ? sozluk.get(k)
10         : w + w.charAt(0); // özel durum
▶      s.append(e);
▶      sozluk.add(w + e.charAt(0));
▶      w = e;
14   }
15   return s.toString();
16 }
```



k = 99 (ASCII) → e = "c" yazılır; sözlüğe w + e[0] = "aa" + "c" = **259 = "aac"** eklenir.

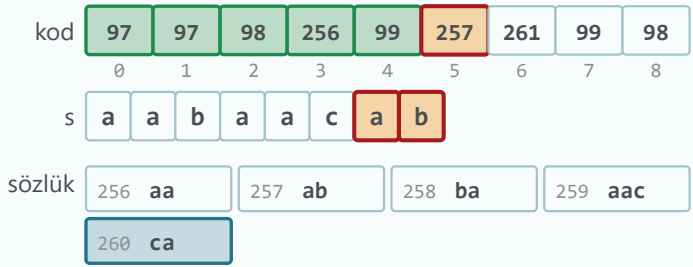
Adım 5 / 10

DEĞİŞKENLER

`i = 4` `k = 99` `w = "c"` `e = "c"`

lzwAc([97 97 98 256 99 257 261 99 98])

```
1  String lzwAc(List<Integer> kod) {
2    List<String> sozluk =
3        ilkSozluk();      // 0..255: ASCII
4    String w = sozluk.get(kod.get(0));
5    StringBuilder s = new StringBuilder(w);
6    for (int i = 1; i < kod.size(); i++) {
       int k = kod.get(i);
       String e = k < sozluk.size()
           ? sozluk.get(k)
10         : w + w.charAt(0); // özel durum
       s.append(e);
       sozluk.add(w + e.charAt(0));
       w = e;
14   }
15   return s.toString();
16 }
```



k = 257 (sözlükte 257 = "ab") → e = "ab" yazılır; sözlüğe w + e[0] = "c" + "a" = **260 = "ca"** eklenir.

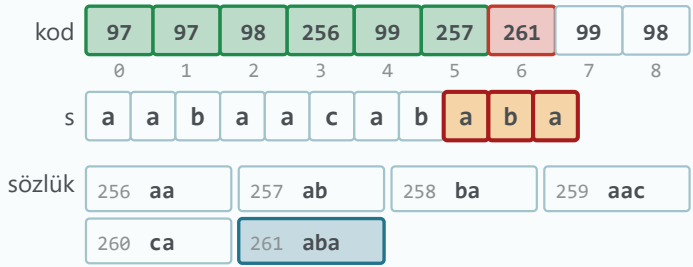
Adım 6 / 10

DEĞİŞKENLER

`i = 5` `k = 257` `w = "ab"` `e = "ab"`

lzwAc([97 97 98 256 99 257 261 99 98])

```
1  String lzwAc(List<Integer> kod) {
2    List<String> sozluk =
3        ilkSozluk();      // 0..255: ASCII
4    String w = sozluk.get(kod.get(0));
5    StringBuilder s = new StringBuilder(w);
6    for (int i = 1; i < kod.size(); i++) {
       int k = kod.get(i);
       String e = k < sozluk.size()
9          ? sozluk.get(k)
           : w + w.charAt(0); // özel durum
       s.append(e);
       sozluk.add(w + e.charAt(0));
       w = e;
14   }
15   return s.toString();
16 }
```



k = 261 ama sözlükte en büyük kod 260: **özel durum**. e = w + w[0] = "ab" + "a" = "aba"; sözlüğe de 261 = "aba" eklenir. Sıkıştırıcı bu kodu, eklediği anda kullanmıştı.

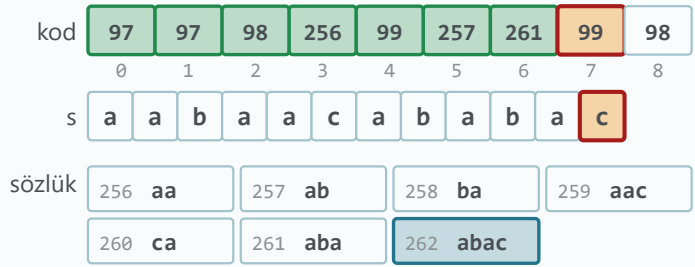
Adım 7 / 10

DEĞİŞKENLER

`i = 6` `k = 261` `w = "aba"` `e = "aba"`

lzwAc([97 97 98 256 99 257 261 99 98])

```
1  String lzwAc(List<Integer> kod) {
2    List<String> sozluk =
3        ilkSozluk();      // 0..255: ASCII
4    String w = sozluk.get(kod.get(0));
5    StringBuilder s = new StringBuilder(w);
6    for (int i = 1; i < kod.size(); i++) {
       int k = kod.get(i);
       String e = k < sozluk.size()
           ? sozluk.get(k)
10         : w + w.charAt(0); // özel durum
       s.append(e);
       sozluk.add(w + e.charAt(0));
       w = e;
14   }
15   return s.toString();
16 }
```



k = 99 (ASCII) → e = "c" yazılır; sözlüğe w + e[0] = "aba" + "c" = **262 = "abac"** eklenir.

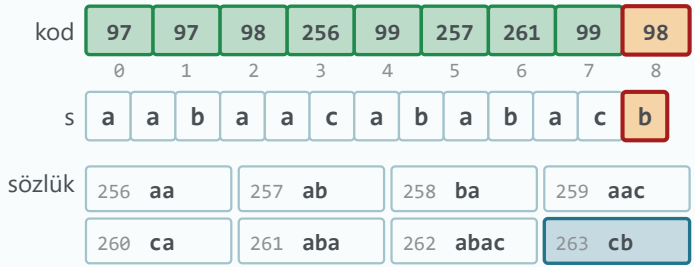
Adım 8 / 10

DEĞİŞKENLER

`i = 7` `k = 99` `w = "c"` `e = "c"`

lzwAc([97 97 98 256 99 257 261 99 98])

```
1  String lzwAc(List<Integer> kod) {
2    List<String> sozluk =
3        ilkSozluk();      // 0..255: ASCII
4    String w = sozluk.get(kod.get(0));
5    StringBuilder s = new StringBuilder(w);
6    for (int i = 1; i < kod.size(); i++) {
       int k = kod.get(i);
       String e = k < sozluk.size()
           ? sozluk.get(k)
10         : w + w.charAt(0); // özel durum
       s.append(e);
       sozluk.add(w + e.charAt(0));
       w = e;
14   }
15   return s.toString();
16 }
```



k = 98 (ASCII) → e = "b" yazılır; sözlüğe w + e[0] = "c" + "b" = **263 = "cb"** eklenir.

Adım 9 / 10

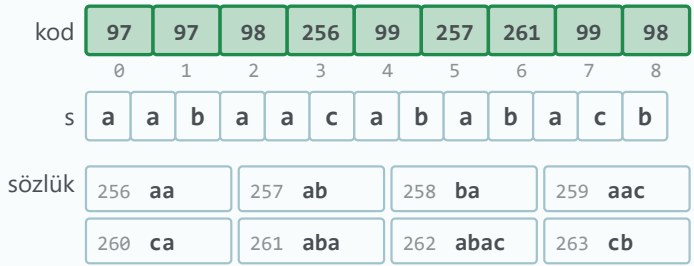
DEĞİŞKENLER

`i = 8` `k = 98` `w = "b"` `e = "b"`

LZW · KODU ADIM ADIM ÇALIŞTIR

lzwAc([97 97 98 256 99 257 261 99 98])

```
1  String lzwAc(List<Integer> kod) {
2    List<String> sozluk =
3        ilkSozluk();      // 0..255: ASCII
4    String w = sozluk.get(kod.get(0));
5    StringBuilder s = new StringBuilder(w);
6    for (int i = 1; i < kod.size(); i++) {
7      int k = kod.get(i);
8      String e = k < sozluk.size()
9          ? sozluk.get(k)
10         : w + w.charAt(0); // özel durum
11     s.append(e);
12     sozluk.add(w + e.charAt(0));
13     w = e;
14   }
▶  return s.toString();
16 }
```



9 kod, 13 karakterlik **aabaacababacb** metnine geri döndü; açıcının kurduğu 256–263 sözlüğü sıkıştırıcınınkiyle aynıdır.

Adım 10 / 10

DEĞİŞKENLER

`i = 9` `k = 98` `w = "b"` `e = "b"`

LZ78: LZW'nin Atası

- LZ78 sözlüğü **boş** başlar; çıktı **(indis, karakter)** çiftleridir: sözlükteki en uzun eşleşmenin indisi (yoksa 0) + onu izleyen karakter.
- Metin: `aabaacabcabcb` → (0,a) (1,b) (1,a) (0,c) (2,c) (5,b).
- Welch'in değişikliği (LZW): sözlüğü tüm tek karakterlerle başlatmak → çıktı yalnızca **kodlardan** oluşur, karakter yazmaya gerek kalmaz.
- LZ77 (kayan pencere) ailesi ise gzip/ZIP (DEFLATE) ve PNG'nin temelidir.

Çıktı	Sözlük
(0, a)	1 = `a`
(1, b)	2 = `ab`
(1, a)	3 = `aa`
(0, c)	4 = `c`
(2, c)	5 = `abc`
(5, b)	6 = `abcb`

LZW · KODU ADIM ADIM ÇALIŞTIR

lz78("aabaacabcabcb")

```
1  List<String> lz78(String s) {
▶    Map<String, Integer> sozluk =
▶        new HashMap<>();
▶    List<String> cikti = new ArrayList<>();
▶    String w = "";
6    for (char c : s.toCharArray()) {
7      if (sozluk.containsKey(w + c)) {
8        w = w + c;               // uzat
9      } else {
10       int k = w.isEmpty() ? 0
11                           : sozluk.get(w);
12       cikti.add("(" + k + ", " + c + ")");
13       sozluk.put(w + c, sozluk.size() + 1);
14       w = "";
15     }
16   }
17   if (!w.isEmpty())        // son parça
18     cikti.add("(" + sozluk.get(w) + ")");
19   return cikti;
20 }
```

s	a	a	b	a	a	c	a	b	c	a	b	c	b
	0	1	2	3	4	5	6	7	8	9	10	11	12

sözlük boş (indis 0 = boş dizgi)

çıktı

LZ78 sözlüğü **boş** başlar (LZW'den farkı budur). w, sözlükte bulunan en uzun ön parçayı tutar; şimdilik boş.

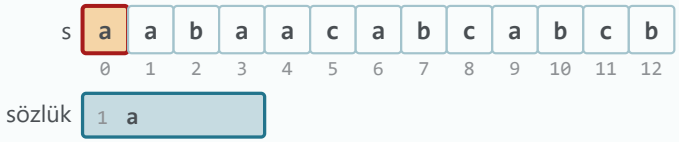
Adım 1 / 14

DEĞİŞKENLER

c = – w = "" k = –

lz78("aabaacabcabcb")

```
1  List<String> lz78(String s) {
2    Map<String, Integer> sozluk =
3        new HashMap<>();
4    List<String> cikti = new ArrayList<>();
5    String w = "";
6    for (char c : s.toCharArray()) {
▶      if (sozluk.containsKey(w + c)) {
8        w = w + c;                 // uzat
9      } else {
▶        int k = w.isEmpty() ? 0
▶                            : sozluk.get(w);
▶        cikti.add("(" + k + ", " + c + ")");
▶        sozluk.put(w + c, sozluk.size() + 1);
▶        w = "";
15     }
16   }
17   if (!w.isEmpty())          // son parça
18     cikti.add("(" + sozluk.get(w) + ")");
19   return cikti;
20 }
```



w boş ve 'a' sözlükte yok → **(0, a)** (0 = boş önek); sözlüğe 1 = "a" eklenir.

Adım 2 / 14

DEĞİŞKENLER

`c = 'a'` `w = ""` `k = 0`

lz78("aabaacabcabcb")

```
1  List<String> lz78(String s) {
2    Map<String, Integer> sozluk =
3        new HashMap<>();
4    List<String> cikti = new ArrayList<>();
5    String w = "";
6    for (char c : s.toCharArray()) {
▶      if (sozluk.containsKey(w + c)) {
▶        w = w + c;                // uzat
9      } else {
10       int k = w.isEmpty() ? 0
11                           : sozluk.get(w);
12       cikti.add("(" + k + ", " + c + ")");
13       sozluk.put(w + c, sozluk.size() + 1);
14       w = "";
15     }
16   }
17   if (!w.isEmpty())         // son parça
18     cikti.add("(" + sozluk.get(w) + ")");
19   return cikti;
20 }
```

s

a	a	b	a	a	c	a	b	c	a	b	c	b
0	1	2	3	4	5	6	7	8	9	10	11	12

sözlük

1	a

çıktı `(0,a)`

w + c = "a" sözlükte var → w = "a" olur; çıktı yazılmaz, bir sonraki karakterle daha uzun bir parça aranır.

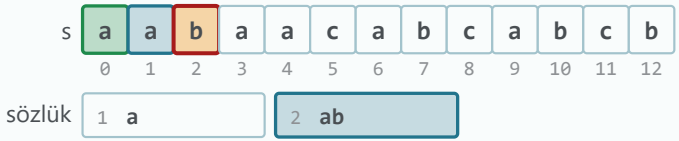
Adım 3 / 14

DEĞİŞKENLER

`c = 'a'` `w = "a"` `k = –`

lz78("aabaacabcabcb")

```
1  List<String> lz78(String s) {
2    Map<String, Integer> sozluk =
3        new HashMap<>();
4    List<String> cikti = new ArrayList<>();
5    String w = "";
6    for (char c : s.toCharArray()) {
▶      if (sozluk.containsKey(w + c)) {
8        w = w + c;                // uzat
9      } else {
▶        int k = w.isEmpty() ? 0
▶                            : sozluk.get(w);
▶        cikti.add("(" + k + ", " + c + ")");
▶        sozluk.put(w + c, sozluk.size() + 1);
▶        w = "";
15     }
16   }
17   if (!w.isEmpty())          // son parça
18     cikti.add("(" + sozluk.get(w) + ")");
19   return cikti;
20 }
```



çıktı `(0,a)` `(1,b)`

"ab" sözlükte yok → w = "a" (indis 1) ve c = 'b' yazılır: **(1, b)**; sözlüğe 2 = "ab" eklenir, w sıfırlanır.

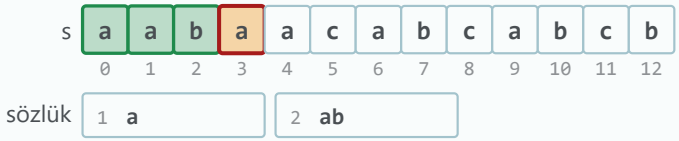
Adım 4 / 14

DEĞİŞKENLER

`c = 'b'` `w = ""` `k = 1`

lz78("aabaacabcabcb")

```
1  List<String> lz78(String s) {
2    Map<String, Integer> sozluk =
3        new HashMap<>();
4    List<String> cikti = new ArrayList<>();
5    String w = "";
6    for (char c : s.toCharArray()) {
▶      if (sozluk.containsKey(w + c)) {
▶        w = w + c;                // uzat
9      } else {
10       int k = w.isEmpty() ? 0
11                           : sozluk.get(w);
12       cikti.add("(" + k + ", " + c + ")");
13       sozluk.put(w + c, sozluk.size() + 1);
14       w = "";
15     }
16   }
17   if (!w.isEmpty())         // son parça
18     cikti.add("(" + sozluk.get(w) + ")");
19   return cikti;
20 }
```



w + c = "a" sözlükte var → w = "a" olur; çıktı yazılmaz, bir sonraki karakterle daha uzun bir parça aranır.

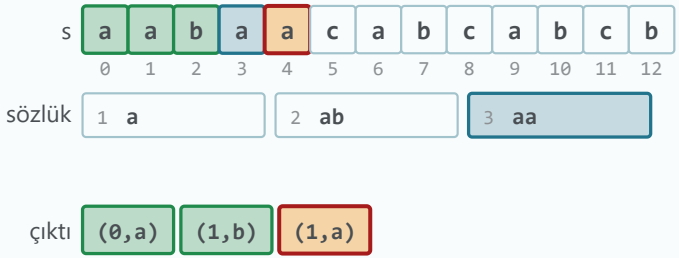
Adım 5 / 14

DEĞİŞKENLER

`c = 'a'` `w = "a"` `k = –`

lz78("aabaacabcabcb")

```
1  List<String> lz78(String s) {
2    Map<String, Integer> sozluk =
3        new HashMap<>();
4    List<String> cikti = new ArrayList<>();
5    String w = "";
6    for (char c : s.toCharArray()) {
▶      if (sozluk.containsKey(w + c)) {
8        w = w + c;                // uzat
9      } else {
▶        int k = w.isEmpty() ? 0
▶                            : sozluk.get(w);
▶        cikti.add("(" + k + ", " + c + ")");
▶        sozluk.put(w + c, sozluk.size() + 1);
▶        w = "";
15     }
16   }
17   if (!w.isEmpty())          // son parça
18     cikti.add("(" + sozluk.get(w) + ")");
19   return cikti;
20 }
```



"aa" sözlükte yok → w = "a" (indis 1) ve c = 'a' yazılır: **(1, a)**; sözlüğe 3 = "aa" eklenir, w sıfırlanır.

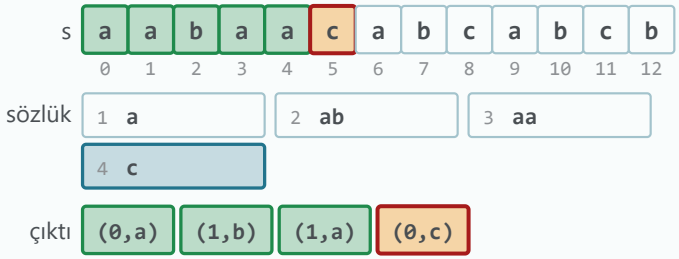
Adım 6 / 14

DEĞİŞKENLER

`c = 'a'` `w = ""` `k = 1`

lz78("aabaacabcabcb")

```
1  List<String> lz78(String s) {
2    Map<String, Integer> sozluk =
3        new HashMap<>();
4    List<String> cikti = new ArrayList<>();
5    String w = "";
6    for (char c : s.toCharArray()) {
▶      if (sozluk.containsKey(w + c)) {
8        w = w + c;                 // uzat
9      } else {
▶        int k = w.isEmpty() ? 0
▶                            : sozluk.get(w);
▶        cikti.add("(" + k + ", " + c + ")");
▶        sozluk.put(w + c, sozluk.size() + 1);
▶        w = "";
15     }
16   }
17   if (!w.isEmpty())          // son parça
18     cikti.add("(" + sozluk.get(w) + ")");
19   return cikti;
20 }
```



w boş ve 'c' sözlükte yok → **(0, c)** (0 = boş önek); sözlüğe 4 = "c" eklenir.

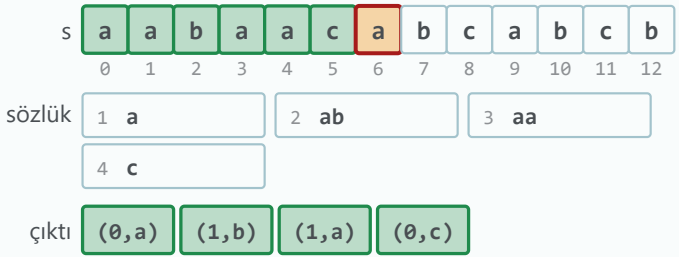
Adım 7 / 14

DEĞİŞKENLER

`c = 'c'` `w = ""` `k = 0`

lz78("aabaacabcabcb")

```
1  List<String> lz78(String s) {
2    Map<String, Integer> sozluk =
3        new HashMap<>();
4    List<String> cikti = new ArrayList<>();
5    String w = "";
6    for (char c : s.toCharArray()) {
▶      if (sozluk.containsKey(w + c)) {
▶        w = w + c;                // uzat
9      } else {
10       int k = w.isEmpty() ? 0
11                           : sozluk.get(w);
12       cikti.add("(" + k + ", " + c + ")");
13       sozluk.put(w + c, sozluk.size() + 1);
14       w = "";
15     }
16   }
17   if (!w.isEmpty())         // son parça
18     cikti.add("(" + sozluk.get(w) + ")");
19   return cikti;
20 }
```



w + c = "a" sözlükte var → w = "a" olur; çıktı yazılmaz, bir sonraki karakterle daha uzun bir parça aranır.

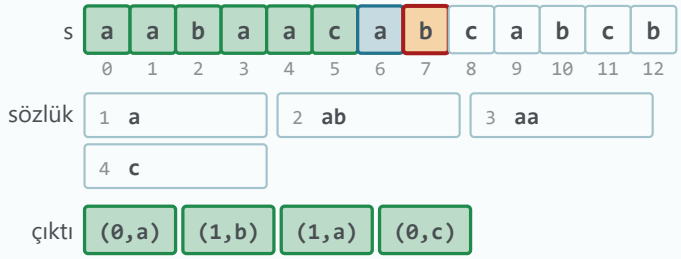
Adım 8 / 14

DEĞİŞKENLER

`c = 'a'` `w = "a"` `k = –`

lz78("aabaacabcabcb")

```
1  List<String> lz78(String s) {
2    Map<String, Integer> sozluk =
3        new HashMap<>();
4    List<String> cikti = new ArrayList<>();
5    String w = "";
6    for (char c : s.toCharArray()) {
▶      if (sozluk.containsKey(w + c)) {
▶        w = w + c;                // uzat
9      } else {
10       int k = w.isEmpty() ? 0
11                           : sozluk.get(w);
12       cikti.add("(" + k + ", " + c + ")");
13       sozluk.put(w + c, sozluk.size() + 1);
14       w = "";
15     }
16   }
17   if (!w.isEmpty())         // son parça
18     cikti.add("(" + sozluk.get(w) + ")");
19   return cikti;
20 }
```



w + c = "ab" sözlükte var → w = "ab" olur; çıktı yazılmaz, bir sonraki karakterle daha uzun bir parça aranır.

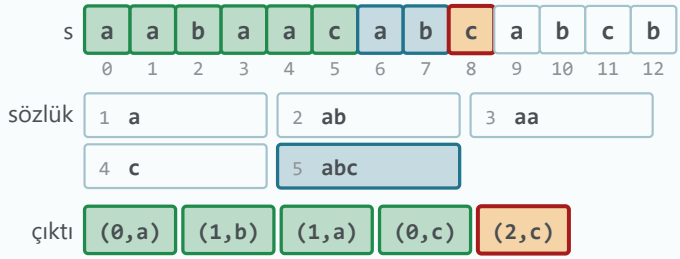
Adım 9 / 14

DEĞİŞKENLER

`c = 'b'` `w = "ab"` `k = –`

lz78("aabaacabcabcb")

```
1  List<String> lz78(String s) {
2    Map<String, Integer> sozluk =
3        new HashMap<>();
4    List<String> cikti = new ArrayList<>();
5    String w = "";
6    for (char c : s.toCharArray()) {
▶      if (sozluk.containsKey(w + c)) {
8        w = w + c;                // uzat
9      } else {
▶        int k = w.isEmpty() ? 0
▶                            : sozluk.get(w);
▶        cikti.add("(" + k + ", " + c + ")");
▶        sozluk.put(w + c, sozluk.size() + 1);
▶        w = "";
15     }
16   }
17   if (!w.isEmpty())          // son parça
18     cikti.add("(" + sozluk.get(w) + ")");
19   return cikti;
20 }
```



"abc" sözlükte yok → w = "ab" (indis 2) ve c = 'c' yazılır: **(2, c)**; sözlüğe 5 = "abc" eklenir, w sıfırlanır.

Adım 10 / 14

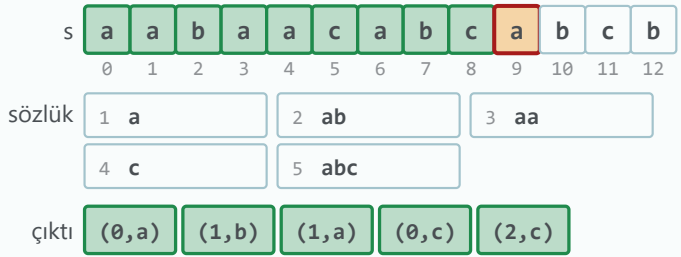
DEĞİŞKENLER

`c = 'c'` `w = ""` `k = 2`

LZW · KODU ADIM ADIM ÇALIŞTIR

lz78("aabaacabcabcb")

```
1  List<String> lz78(String s) {
2    Map<String, Integer> sozluk =
3        new HashMap<>();
4    List<String> cikti = new ArrayList<>();
5    String w = "";
6    for (char c : s.toCharArray()) {
▶      if (sozluk.containsKey(w + c)) {
▶        w = w + c;                // uzat
9      } else {
10       int k = w.isEmpty() ? 0
11                           : sozluk.get(w);
12       cikti.add("(" + k + ", " + c + ")");
13       sozluk.put(w + c, sozluk.size() + 1);
14       w = "";
15     }
16   }
17   if (!w.isEmpty())         // son parça
18     cikti.add("(" + sozluk.get(w) + ")");
19   return cikti;
20 }
```



w + c = "a" sözlükte var → w = "a" olur; çıktı yazılmaz, bir sonraki karakterle daha uzun bir parça aranır.

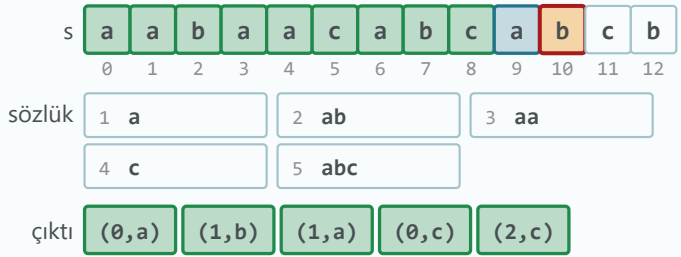
Adım 11 / 14

DEĞİŞKENLER

`c = 'a'` `w = "a"` `k = –`

lz78("aabaacabcabcb")

```
1  List<String> lz78(String s) {
2    Map<String, Integer> sozluk =
3        new HashMap<>();
4    List<String> cikti = new ArrayList<>();
5    String w = "";
6    for (char c : s.toCharArray()) {
▶      if (sozluk.containsKey(w + c)) {
▶        w = w + c;                // uzat
9      } else {
10       int k = w.isEmpty() ? 0
11                           : sozluk.get(w);
12       cikti.add("(" + k + ", " + c + ")");
13       sozluk.put(w + c, sozluk.size() + 1);
14       w = "";
15     }
16   }
17   if (!w.isEmpty())         // son parça
18     cikti.add("(" + sozluk.get(w) + ")");
19   return cikti;
20 }
```



w + c = "ab" sözlükte var → w = "ab" olur; çıktı yazılmaz, bir sonraki karakterle daha uzun bir parça aranır.

Adım 12 / 14

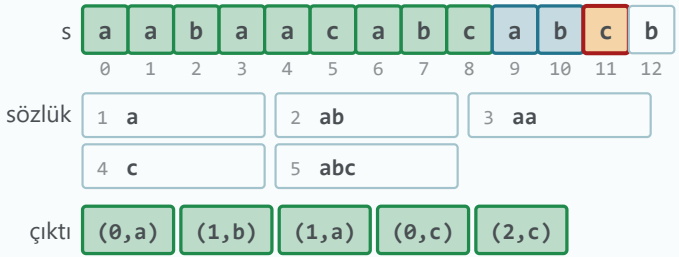
DEĞİŞKENLER

`c = 'b'` `w = "ab"` `k = –`

LZW · KODU ADIM ADIM ÇALIŞTIR

lz78("aabaacabcabcb")

```
1  List<String> lz78(String s) {
2    Map<String, Integer> sozluk =
3        new HashMap<>();
4    List<String> cikti = new ArrayList<>();
5    String w = "";
6    for (char c : s.toCharArray()) {
▶      if (sozluk.containsKey(w + c)) {
▶        w = w + c;                // uzat
9      } else {
10       int k = w.isEmpty() ? 0
11                           : sozluk.get(w);
12       cikti.add("(" + k + ", " + c + ")");
13       sozluk.put(w + c, sozluk.size() + 1);
14       w = "";
15     }
16   }
17   if (!w.isEmpty())         // son parça
18     cikti.add("(" + sozluk.get(w) + ")");
19   return cikti;
20 }
```



w + c = "abc" sözlükte var → w = "abc" olur; çıktı yazılmaz, bir sonraki karakterle daha uzun bir parça aranır.

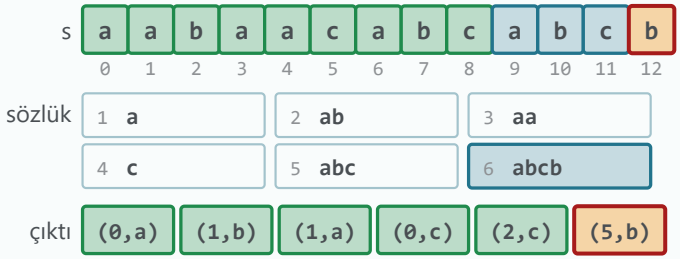
Adım 13 / 14

DEĞİŞKENLER

`c = 'c'` `w = "abc"` `k = –`

lz78("aabaacabcabcb")

```
1  List<String> lz78(String s) {
2    Map<String, Integer> sozluk =
3        new HashMap<>();
4    List<String> cikti = new ArrayList<>();
5    String w = "";
6    for (char c : s.toCharArray()) {
▶      if (sozluk.containsKey(w + c)) {
8        w = w + c;                // uzat
9      } else {
▶        int k = w.isEmpty() ? 0
▶                            : sozluk.get(w);
▶        cikti.add("(" + k + ", " + c + ")");
▶        sozluk.put(w + c, sozluk.size() + 1);
▶        w = "";
15     }
16   }
17   if (!w.isEmpty())          // son parça
18     cikti.add("(" + sozluk.get(w) + ")");
19   return cikti;
20 }
```



"abcb" sözlükte yok → w = "abc" (indis 5) ve c = 'b' yazılır: **(5, b)**; sözlüğe 6 = "abcb" eklenir, w sıfırlanır. Metin bitti: 6 çift.

Adım 14 / 14

DEĞİŞKENLER

`c = 'b'` `w = ""` `k = 5`

RLE ve LZW Karşılaştırması

	RLE	LZW
Yakaladığı fazlalık	aynı sembolün ardışık tekrarı	tekrar eden **dizgiler** (ardışık olmasa da)
Ek yapı	yok	sözlük (hash tablosu / trie)
Zaman	O(n)	O(n) (sözlük erişimi O(1) ortalama)
En kötü durum	tekrar yoksa çıktı 2 katına çıkar	kısa/rastgele veride kazanç az, kod genişliği büyür
Kullanım	faks, BMP/PCX, basit grafikler	GIF, TIFF, PDF, Unix `compress`

Her ikisi de **kayıpsızdır**: sıkıştırılmış veri açıldığında orijinal veri aynen elde edilir. Hiçbir kayıpsız yöntem **her** girdiyi küçültemez (güvercin yuvası ilkesi).

ÖZET

Özet

Konu	Hatırlanacaklar
Kaba kuvvet	her kaymayı soldan sağa dene; ön işleme yok — en kötü O(n · m)
KMP	lps[q] = en uzun uygun önek = sonek; uyuşmazlıkta q = lps[q − 1], i geri gitmez — O(n + m)
Boyer-Moore	sağdan sola; s += max(1, j − son[c]) ve iyi sonek; en iyi O(n/m), pratikte en hızlı
Rabin-Karp	polinom kayan hash, O(1) güncelleme; hash eşitse doğrula; beklenen O(n + m)
RLE	koşu → (sayı, sembol); O(n); tekrar yoksa büyütür; bayrak baytı ile iyileştirilir
LZW	w + c sözlükte varsa uzat, yoksa kod(w) yaz ve w + c'yi ekle; açıcı sözlüğü kendisi kurar (özel durum: w + w[0])

ÖZET

Etkileşimli Simülatörler

Dizgi Eşleştirme

Herhangi bir metin ve örüntü için Naive, KMP, Boyer-Moore (kötü karakter tablosu) ve Rabin-Karp (taban 256, mod 101 hash) adım adım ya da otomatik oynatmayla; bulunan eşleşmeler işaretlenir.

Dizgi Sıkıştırma

RLE ve LZW bir girdi üzerinde adım adım: LZW sözlük tablosunun büyümesi, sıkıştırılmış çıktı ve sıkıştırma oranı.

Önek Fonksiyonu ve Z Algoritması

Bir dizgi için KMP önek (π, LPS) dizisi ve Z dizisi yan yana.

ÖZET

Sık Yapılan Hatalar

Döngü sınırı

Kaba kuvvette `i < n − m` yazmak son kaymayı (s = n − m) atlar; doğrusu `i <= n − m`.

LPS'yi yanlış tanımlamak

Örüntünün kendisi sayılmaz: lps("aaaa") = [0, 1, 2, 3], [1, 2, 3, 4] değil. KMP'de uyuşmazlıkta **lps[q − 1]** kullanılır, lps[q] değil.

Negatif kaydırma

Boyer-Moore'da j − son[c] negatif olabilir (c, j'nin sağında). `max(1, …)` unutulursa örüntü geri gider ya da sonsuz döngü olur.

Hash eşitliğine güvenmek

Rabin-Karp'ta hash eşitliği eşleşme demek değildir: doğrulama yapılmalı. Java'da `%` negatif sonuç verebilir: `if (th < 0) th += Q`.

RLE'de çok basamaklı sayılar

"12A" çözülürken sayıyı tek rakam okumak hatalıdır (1 tane '2'?). Sayı sınırlanmalı ya da bayt olarak yazılmalı.

LZW'de son w ve özel durum

Döngüden sonra kalan w'nin kodunu yazmayı unutmak; açarken henüz tanımlanmamış kodu (w + w[0]) ele almamak.

ÖZET

Örnek Problemler

1. "aabaaab" örüntüsünün LPS tablosunu bulun.
2. T = "aaaaaaaa" (n = 8), P = "aab" için kaba kuvvet kaç karşılaştırma yapar?
3. Boyer-Moore (yalnız kötü karakter): T = "GCAATGCCTATGTGACC", P = "TATGTG". İlk kaydırma kaç olur?
4. Taban 10, Q = 13, karakterler rakam: T = "31415", m = 3. h("314") verildiğinde h("141")'i kaydırma formülüyle bulun.
5. "WWWWBBBWWWWWWB" dizgisini RLE ile kodlayın; oranı hesaplayın.
6. A = 0, B = 1 sözlüğüyle "ABABABA" dizgisini LZW ile sıkıştırın.

▶ **Cevapları göster**

ÖZET

Alıştırmalar

1. KMP aramasını, **tüm** geçişleri bir listede döndürecek şekilde değiştirin.
2. "abcbcabbabcabc" örüntüsünün son görülme tablosunu ve "abccbaabccbaabcbcabbabcabc" metnindeki ilk kaydırmayı bulun.
3. Rabin-Karp ile aynı uzunlukta k örüntüyü tek geçişte arayan bir yöntem tasarlayın.
4. `rleAc(String s)` : çok basamaklı sayıları da doğru çözen RLE açıcıyı yazın.
5. 0 1 3 2 3 5 0 kodlarını (A = 0, B = 1, C = 2) LZW ile açın.
6. Kaba kuvvetin en kötü durumunu üreten bir T, P çifti (n = 12, m = 4) verin; karşılaştırma sayısını yazın.

▶ **Cevaplar / ipuçları**

Kaynaklar

- Cormen, Leiserson, Rivest ve Stein, *Introduction to Algorithms*, 4. baskı, 2022 (Bölüm 32: String Matching).
- Sedgewick ve Wayne, *Algorithms*, 4. baskı, 2011 (Bölüm 5.3 Substring Search, 5.5 Data Compression).
- Goodrich, Tamassia ve Goldwasser, *Data Structures and Algorithms in Java*, 6. baskı, 2014 (Bölüm 13).
- D. E. Knuth, J. H. Morris, V. R. Pratt, "Fast Pattern Matching in Strings", *SIAM J. Computing* 6(2), 1977.
- R. S. Boyer, J S. Moore, "A Fast String Searching Algorithm", *Communications of the ACM* 20(10), 1977.
- R. M. Karp, M. O. Rabin, "Efficient Randomized Pattern-Matching Algorithms", *IBM J. Res. Dev.* 31(2), 1987.
- T. A. Welch, "A Technique for High-Performance Data Compression", *IEEE Computer* 17(6), 1984.

Son

Bölüm 9: Dizgi Eşleştirme ve Sıkıştırma

Sonraki bölüm: Bölüm 10: İleri Dizgi Algoritmaları · Terim sözlüğü