Bölüm 12: Eşlemeler ve Hash Tabloları

Veri Yapıları

Eşleme (Map) ADT | Doğrudan adresleme | Hash fonksiyonu | Çakışma | Ayrı zincirleme | Açık adresleme | Yük faktörü | Yeniden boyutlandırma

Sercan KÜLCÜ · Giresun Üniversitesi · Bilgisayar Mühendisliği

GİRİŞ

Öğrenme Çıktıları

Eşleme modeli

Anahtar-değer ilişkisini ve hash tablosunun temel işleyişini açıklamak.

Çakışmalar

Zincirleme ve açık adresleme yöntemlerini uygulamak.

Yük faktörü

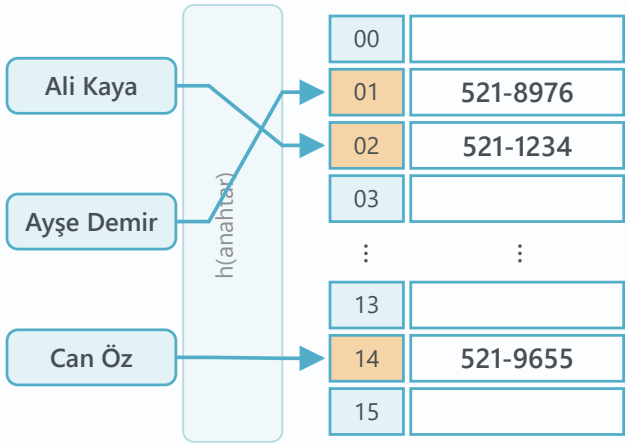
Yük faktörü, yeniden boyutlandırma ve dağılım ilişkisini yorumlamak.

Doğruluk ve maliyet

`equals` / `hashCode` sözleşmesini uygulayıp ortalama ve en kötü durum maliyetlerini karşılaştırmak.

Eşleme (Map)

- Veri öğelerini **anahtar-değer** (key-value) çiftleri olarak saklar.
- Anahtarlar her öğeyi **benzersiz** (tekil) olarak tanımlar; değerler bu anahtarlarla ilişkilendirilir.
- Değere anahtar üzerinden erişilir: "Ayşe Demir'in telefonu nedir?"
- Hash tablosuyla gerçeklendiğinde işlemler **ortalama O(1)** sürer.



Eşleme bir **soyut veri tipidir** (ne yapılacağını söyler); hash tablosu ise onu gerçekleyen en yaygın **veri yapısıdır** (nasıl yapılacağını söyler).

Eşleme ADT'sinin Temel İşlevleri

İşlev	java.util.Map	Açıklama
Ekleme (insertion)	`put(k, v)`	Çifti tabloya ekler
Arama (search)	`get(k)`	k anahtarının değerini döndürür; yoksa `null`
Silme (deletion)	`remove(k)`	k anahtarlı öğeyi çıkarır, değerini döndürür
Güncelleme (update)	`put(k, v')`	k zaten varsa değeri v' ile değiştirir (eskisini döndürür)
Diğer	`size()`, `isEmpty()`, `containsKey(k)`, `keySet()`, `values()`	

```
Map<Integer, String> m = new HashMap<>();
m.put(105, "Murat");   m.put(21, "Leyla");
m.put(21, "Ali");      // güncelleme: "Leyla" → "Ali"
m.get(21);             // "Ali"
m.remove(105);         // "Murat"
m.get(88);             // null: böyle bir anahtar yok
```

Dizi Üzerinde Arama İşlemi

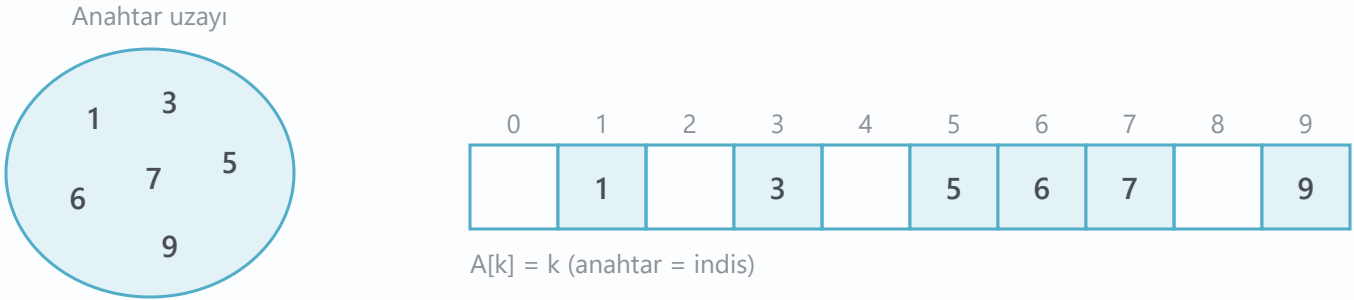
- Dizide arama işleminin zaman karmaşıklığı O(1) yapılabilir mi?
- Öğeleri birbiriyle karşılaştırarak zor: doğrusal arama O(n), sıralı dizide ikili arama O(log n).
- Ama aranan öğenin **konumunu (indisini) biliyorsak** erişim tek adımdır:

```
int[] siraliDizi = {1, 3, 5, 7, 9};
int aranacakIndis = 2;                    // erişilmek istenen öğenin indisi
int oge = siraliDizi[aranacakIndis];      // O(1): oge = 5
```

Fikir: Anahtarı doğrudan bir dizi indisine çevirebilirsek, karşılaştırma yapmadan tek adımda doğru hücreye gideriz.

İndisleme: Anahtar = İndis

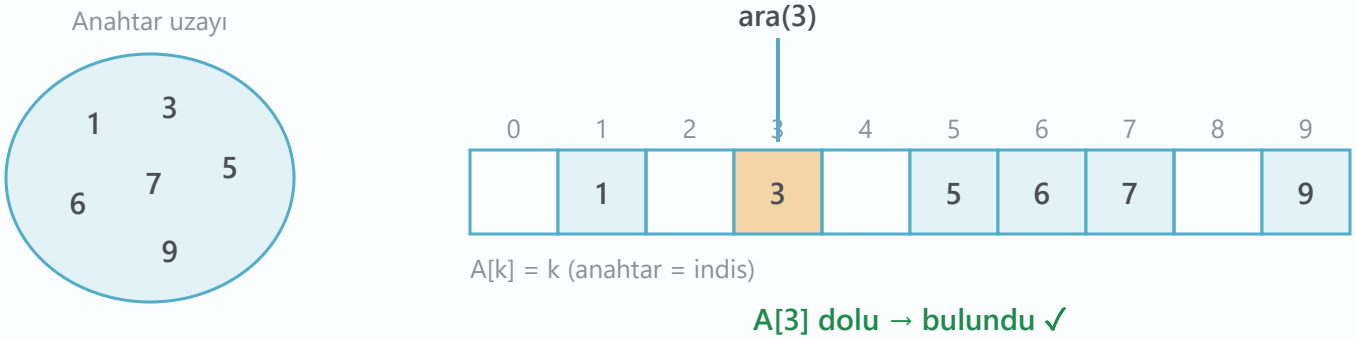
Anahtarlar 0–9 arasındaki tam sayılarsa, k anahtarını dizinin k numaralı hücresine koyarız. Arama tek adımdır.



Anahtar uzayındaki her k değeri A[k] hücresine yerleştirilir; kullanılmayan hücreler boş kalır.

İndisleme: Anahtar = İndis

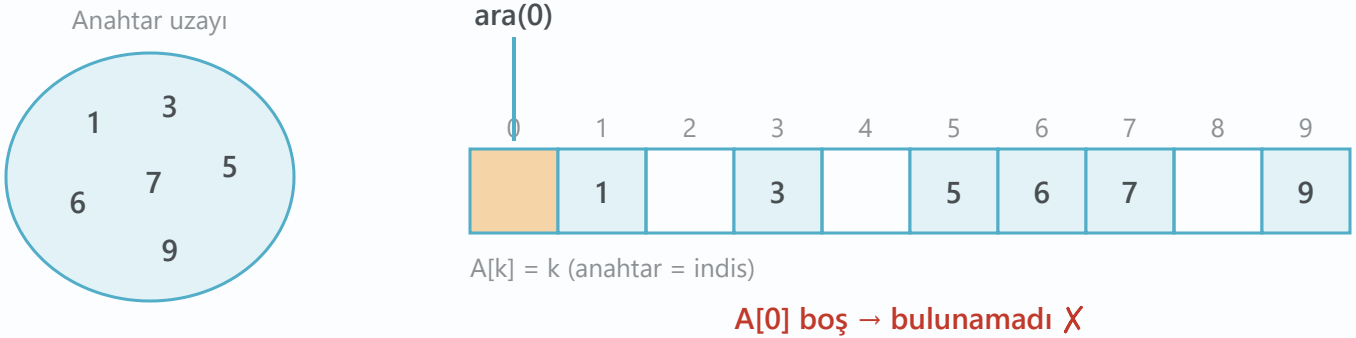
Anahtarlar 0–9 arasındaki tam sayılarsa, k anahtarını dizinin k numaralı hücresine koyarız. Arama tek adımdır.



ara(3): doğrudan A[3]'e bakılır → dolu → **bulundu**, O(1).

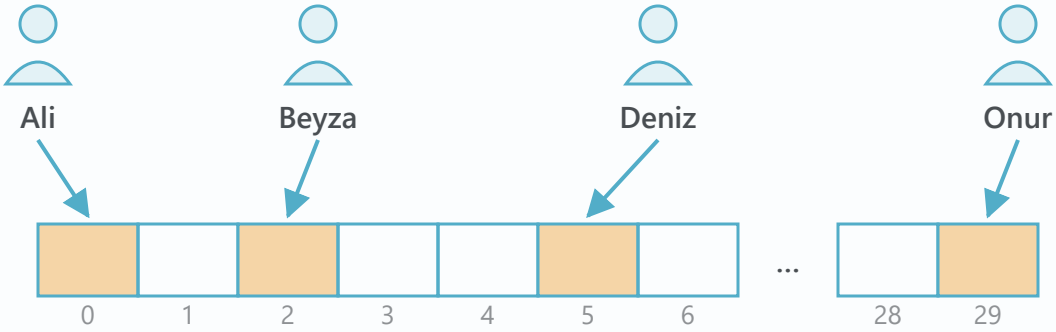
İndisleme: Anahtar = İndis

Anahtarlar 0–9 arasındaki tam sayılarsa, k anahtarını dizinin k numaralı hücresine koyarız. Arama tek adımdır.



ara(0): A[0]'a bakılır → boş → **bulunamadı**, yine O(1).

Doğrudan Adresleme ve Sınırı

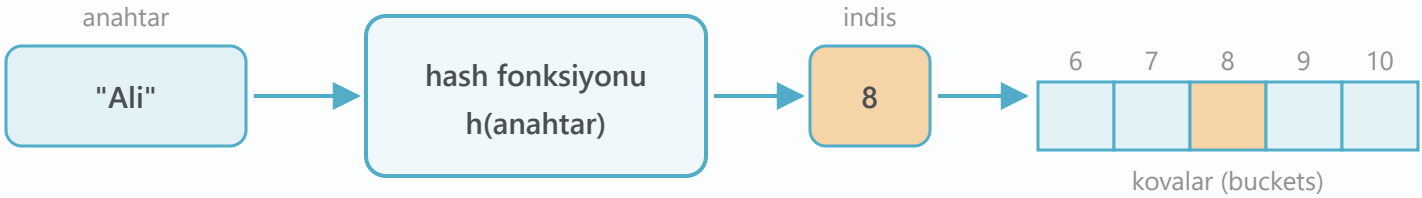


- Her olası anahtar için bir hücre ayrılır: arama, ekleme, silme O(1).
- Anahtar uzayı büyükse bellek israfı çok büyüktür: 9 haneli okul numaraları için 10^9 hücre (int dizisiyle ≈ 4 GB) gerekir, oysa bölümde birkaç yüz öğrenci vardır.
- Anahtar tam sayı değilse ("Ali", "Beyza") doğrudan indis olarak kullanılamaz.

Çözüm: Büyük anahtar uzayını küçük bir diziye sıkıştıran bir fonksiyon kullanmak → **hash fonksiyonu**.

Hash Tekniği

- Öğe saklama ve erişim amacıyla kullanılan bir tekniktir.
- Öğe için hızlı bir şekilde konum (**kova**, bucket) bulmak için kullanılır.
- Konum belli ise dizide öğe ekleme, çıkarma ve güncelleme kolaydır.
- Hash tekniği bir **hash fonksiyonu** kullanır: aldığı anahtarı `0 … N−1` aralığında bir tam sayıya (indise) dönüştürür.



"Ali".hashCode() = 65918 → 65918 mod 13 = 8 (N = 13 kovalı tablo)

Hash Fonksiyonları

- Hash tablosunun temelinde hash fonksiyonu yer alır: anahtarı alır, öğenin tablodaki konumunu (indisini) üretir.

Belirleyici (deterministic)
Aynı anahtar her zaman aynı indisi vermeli; yoksa öğe bir daha bulunamaz.

Hızlı
Hesaplaması O(1) olmalı (anahtar uzunluğu sabit kabul edilir).

Düzgün dağılım
Anahtarları kovalara eşit yaymalı; çakışmaları azaltır.

Tüm anahtarı kullanmalı
Yalnızca ilk harfe bakan fonksiyon "Ali", "Ayşe", "Ahmet"i aynı kovaya atar.

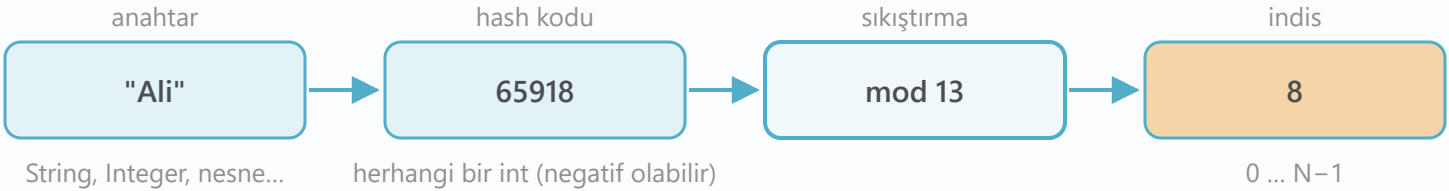
Kötü fonksiyon

– Her öğe aynı yuvaya hash'lenirse tablo bir bağlı listeye döner: arama O(n).

Kapasite (N) seçimi

+ Büyük kapasite belleği israf eder; küçük kapasite çakışmaları artırır.

Hash Kodu + Sıkıştırma



Bölme yöntemi

`h(k) = k mod N`

N'yi **asal** seçmek, anahtarlardaki düzenli örüntülerin (10'un katları gibi) aynı kovalara yığılmasını önler.

MAD (çarp-topla-böl)

`h(k) = ((a·k + b) mod p) mod N`

p > N asal; a, b rastgele (a ≠ 0).

k	k mod 10	k mod 11
10	0	10
20	0	9
30	0	8
40	0	7
50	0	6

N = 10: hepsi aynı kovada. N = 11 (asal): hepsi farklı kovalarda.

HASH FONKSİYONLARI

`equals` ve `hashCode` Sözleşmesi

Eşit nesneler

`a.equals(b)` doğruysa `a.hashCode() == b.hashCode()` olmak zorundadır; yoksa eşit anahtarlar farklı kovalara düşer.

Aynı hash kodu

İki nesnenin hash kodunun aynı olması, nesnelerin eşit olduğu anlamına gelmez. Hash tablosu kovada ayrıca `equals` ile karşılaştırır.

Birlikte ezme

`equals` yeniden tanımlanıyorsa `hashCode` da aynı alanlara göre yeniden tanımlanmalıdır.

Değiştirilebilir anahtar

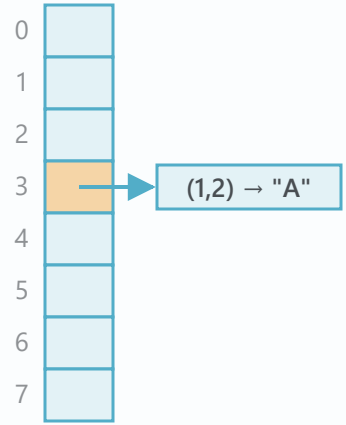
Tabloya eklendikten sonra equals/hashCode alanları değişirse öğe doğru kovada aranamaz.

```
record Ogrenci(int no, String ad) {}
Map<Ogrenci, Double> notlar = new HashMap<>();
notlar.put(new Ogrenci(42, "Ada"), 87.5); // record: equals/hashCode hazır (Java 16+)
```

Hash tablosunda kova seçimini `hashCode`, anahtarın gerçekten aynı olup olmadığını `equals` belirler.

Sık Hata: `equals` Var, `hashCode` Yok

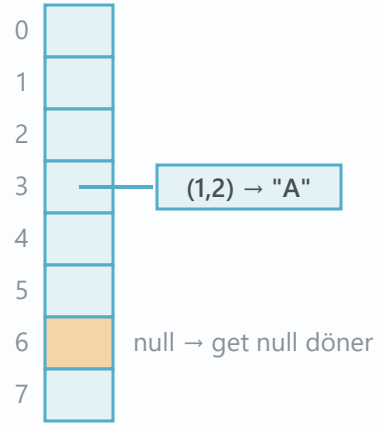
```
class Nokta {
  int x, y;
  Nokta(int x, int y) { this.x = x; this.y = y; }
  public boolean equals(Object o) {
    return o instanceof Nokta p
        && x == p.x && y == p.y;
  }
  // hashCode() yok -> Object'inki
}
Map<Nokta, String> m = new HashMap<>();
m.put(new Nokta(1, 2), "A");
m.get(new Nokta(1, 2));  // büyük olasılıkla null (!)
```



`put(p1, "A")` : kova, `p1.hashCode()` ile seçilir. Nokta ezmediği için **Object.hashCode()** (nesne kimliği) kullanılır; örneğin kova 3.

Sık Hata: `equals` Var, `hashCode` Yok

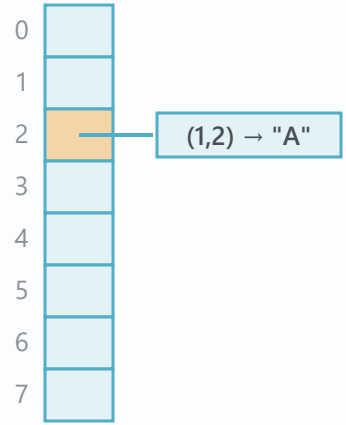
```
class Nokta {
  int x, y;
  Nokta(int x, int y) { this.x = x; this.y = y; }
  public boolean equals(Object o) {
    return o instanceof Nokta p
        && x == p.x && y == p.y;
  }
  // hashCode() yok -> Object'inki
}
Map<Nokta, String> m = new HashMap<>();
m.put(new Nokta(1, 2), "A");
m.get(new Nokta(1, 2));  // büyük olasılıkla null (!)
```



`get(p2)` : p2, p1 ile **eşit** (equals true) ama ayrı bir nesne; kimlik hash'i (neredeyse her zaman) farklı → örneğin kova 6'ya bakılır, orası boş → **null**. equals hiç çağrılmaz!

Sık Hata: `equals` Var, `hashCode` Yok

```
class Nokta {
  int x, y;
  Nokta(int x, int y) { this.x = x; this.y = y; }
  public boolean equals(Object o) {
    return o instanceof Nokta p
        && x == p.x && y == p.y;
  }
  // hashCode() yok -> Object'inki
}
Map<Nokta, String> m = new HashMap<>();
m.put(new Nokta(1, 2), "A");
m.get(new Nokta(1, 2));  // büyük olasılıkla null (!)
```



Çözüm: `hashCode()` da ezilir:

`return Objects.hash(x, y);`

Eşit noktalar aynı kovaya düşer, kovada equals eşleşmeyi bulur → **"A"**.

Örnek Hash Fonksiyonları

Mod alma (bölme kalanı)

```
int modulusHash(int anahtar, int diziBoyu) {
  return Math.floorMod(anahtar, diziBoyu);
}
```

`floorMod`, negatif anahtarlarda da 0 … diziBoyu−1 aralığında güvenli bir indis üretir.

CRC32

```
long crc32Hash(String anahtar) {
  CRC32 crc = new CRC32();  // java.util.zip
  crc.update(anahtar.getBytes());
  return crc.getValue();
}
```

DJB2 (dizgi)

```
int djb2Hash(String s) {
  int h = 5381;
  for (int i = 0; i < s.length(); i++)
    h = (h * 33) ^ s.charAt(i);
  return h;
}
```

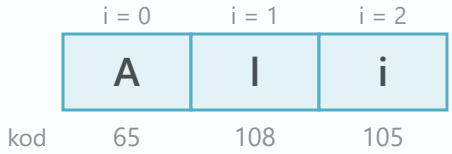
Klasik DJB2 `h*33 + c` kullanır; burada XOR'lu biçimi var. Sonuç negatif olabilir, indis için sıkıştırma gerekir.

Java: String.hashCode()

```
// Σ s[i] · 31^(n−1−i)
"abc".hashCode()  // 96354
```

$97 \cdot 31^2 + 98 \cdot 31 + 99 = 96354$

Kod İzleme: `"Ali".hashCode()`



`hash("Ali")` çağrılır; her karakter Unicode koduyla (A = 65, l = 108, i = 105) işlenir.

Adım 1 / 12

```
1  // String.hashCode() ile aynı hesap:
2  int hash(String s) {
3    int h = 0;
4    for (int i = 0; i < s.length(); i++)
5      h = 31 * h + s.charAt(i);
6    return h;
7  }
8
9  // main:
▶ int kod = hash("Ali");                  // 65918
11 int indis = Math.floorMod(kod, 13);    // 8
```

DEĞİŞKENLER

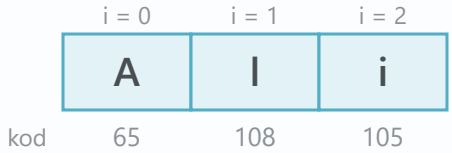
`s = "Ali"`

ÇAĞRI YIĞINI

`hash("Ali")`

`main()`

Kod İzleme: `"Ali".hashCode()`



h = 0

h = 0 ile başlanır.

Adım 2 / 12

```
1  // String.hashCode() ile aynı hesap:
2  int hash(String s) {
▶    int h = 0;
4    for (int i = 0; i < s.length(); i++)
5      h = 31 * h + s.charAt(i);
6    return h;
7  }
8
9  // main:
10 int kod = hash("Ali");                // 65918
11 int indis = Math.floorMod(kod, 13);   // 8
```

DEĞİŞKENLER

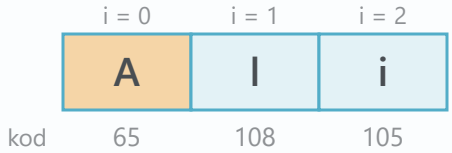
`s = "Ali"` `h = 0`

ÇAĞRI YIĞINI

`hash("Ali")`

`main()`

Kod İzleme: `"Ali".hashCode()`



h = 0

0 < 3 → döngü sürer; karakter **s.charAt(0) = 'A'** (65).

Adım 3 / 12

```
1  // String.hashCode() ile aynı hesap:
2  int hash(String s) {
3    int h = 0;
▶    for (int i = 0; i < s.length(); i++)
5      h = 31 * h + s.charAt(i);
6    return h;
7  }
8
9  // main:
10 int kod = hash("Ali");                  // 65918
11 int indis = Math.floorMod(kod, 13);     // 8
```

DEĞİŞKENLER

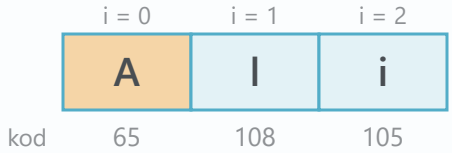
`s = "Ali"` `h = 0` `i = 0`

ÇAĞRI YIĞINI

`hash("Ali")`

`main()`

Kod İzleme: `"Ali".hashCode()`



$h = 31 \cdot 0 + 65$

$= 65$

h = 31 · 0 + 65 = 65. Önceki toplam 31 ile çarpılır: karakterin **konumu** da sonucu etkiler.

Adım 4 / 12

```
1  // String.hashCode() ile aynı hesap:
2  int hash(String s) {
3    int h = 0;
4    for (int i = 0; i < s.length(); i++)
▶      h = 31 * h + s.charAt(i);
6    return h;
7  }
8
9  // main:
10 int kod = hash("Ali");                  // 65918
11 int indis = Math.floorMod(kod, 13);     // 8
```

DEĞİŞKENLER

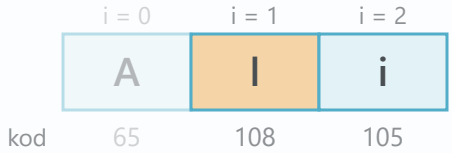
`s = "Ali"` `h = 65` `i = 0`

ÇAĞRI YIĞINI

`hash("Ali")`

`main()`

Kod İzleme: `"Ali".hashCode()`



h = 65

1 < 3 → döngü sürer; karakter **s.charAt(1) = 'l'** (108).

Adım 5 / 12

```
1  // String.hashCode() ile aynı hesap:
2  int hash(String s) {
3    int h = 0;
▶    for (int i = 0; i < s.length(); i++)
5      h = 31 * h + s.charAt(i);
6    return h;
7  }
8
9  // main:
10 int kod = hash("Ali");                  // 65918
11 int indis = Math.floorMod(kod, 13);     // 8
```

DEĞİŞKENLER

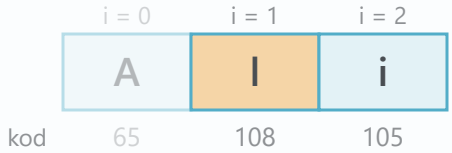
`s = "Ali"` `h = 65` `i = 1`

ÇAĞRI YIĞINI

`hash("Ali")`

`main()`

Kod İzleme: `"Ali".hashCode()`



$h = 31 \cdot 65 + 108$

$= 2123$

h = 31 · 65 + 108 = 2123. Önceki toplam 31 ile çarpılır: karakterin **konumu** da sonucu etkiler.

Adım 6 / 12

```
1  // String.hashCode() ile aynı hesap:
2  int hash(String s) {
3    int h = 0;
4    for (int i = 0; i < s.length(); i++)
▶      h = 31 * h + s.charAt(i);
6    return h;
7  }
8
9  // main:
10 int kod = hash("Ali");                  // 65918
11 int indis = Math.floorMod(kod, 13);     // 8
```

DEĞİŞKENLER

`s = "Ali"` `h = 2123` `i = 1`

ÇAĞRI YIĞINI

`hash("Ali")`

`main()`

Kod İzleme: `"Ali".hashCode()`



h = 2123

2 < 3 → döngü sürer; karakter **s.charAt(2) = 'i'** (105).

Adım 7 / 12

```
1  // String.hashCode() ile aynı hesap:
2  int hash(String s) {
3    int h = 0;
▶    for (int i = 0; i < s.length(); i++)
5      h = 31 * h + s.charAt(i);
6    return h;
7  }
8
9  // main:
10 int kod = hash("Ali");                 // 65918
11 int indis = Math.floorMod(kod, 13);    // 8
```

DEĞİŞKENLER

`s = "Ali"` `h = 2123` `i = 2`

ÇAĞRI YIĞINI

`hash("Ali")`

`main()`

Kod İzleme: `"Ali".hashCode()`



h = 31 · 2123 + 105

= 65918

h = 31 · 2123 + 105 = 65918. Önceki toplam 31 ile çarpılır: karakterin **konumu** da sonucu etkiler.

Adım 8 / 12

```
1  // String.hashCode() ile aynı hesap:
2  int hash(String s) {
3    int h = 0;
4    for (int i = 0; i < s.length(); i++)
▶      h = 31 * h + s.charAt(i);
6    return h;
7  }
8
9  // main:
10 int kod = hash("Ali");                // 65918
11 int indis = Math.floorMod(kod, 13);   // 8
```

DEĞİŞKENLER

`s = "Ali"` `h = 65918` `i = 2`

ÇAĞRI YIĞINI

`hash("Ali")`

`main()`

Kod İzleme: `"Ali".hashCode()`



h = 65918

3 < 3 yanlış → döngü biter.


```
1  // String.hashCode() ile aynı hesap:
2  int hash(String s) {
3    int h = 0;
▶    for (int i = 0; i < s.length(); i++)
5      h = 31 * h + s.charAt(i);
6    return h;
7  }
8
9  // main:
10 int kod = hash("Ali");                // 65918
11 int indis = Math.floorMod(kod, 13);   // 8
```

DEĞİŞKENLER

`s = "Ali"` `h = 65918` `i = 3`

ÇAĞRI YIĞINI

`hash("Ali")`

`main()`

Kod İzleme: `"Ali".hashCode()`



$65 \cdot 31^2 + 108 \cdot 31 + 105$

$= 65918$

`return 65918` = $65 \cdot 31^2 + 108 \cdot 31 + 105$.

Adım 10 / 12

```
1  // String.hashCode() ile aynı hesap:
2  int hash(String s) {
3    int h = 0;
4    for (int i = 0; i < s.length(); i++)
5      h = 31 * h + s.charAt(i);
▶    return h;
7  }
8
9  // main:
10 int kod = hash("Ali");                  // 65918
11 int indis = Math.floorMod(kod, 13);     // 8
```

DEĞİŞKENLER

`s = "Ali"` `h = 65918`

ÇAĞRI YIĞINI

`hash("Ali")`

`main()`

Kod İzleme: `"Ali".hashCode()`



kod = 65918

kod = 65918. Uzun dizgilerde çarpım int sınırını aşar ve sonuç **negatif** olabilir.

Adım 11 / 12

```
1  // String.hashCode() ile aynı hesap:
2  int hash(String s) {
3    int h = 0;
4    for (int i = 0; i < s.length(); i++)
5      h = 31 * h + s.charAt(i);
6    return h;
7  }
8
9  // main:
▶ int kod = hash("Ali");                  // 65918
11 int indis = Math.floorMod(kod, 13);    // 8
```

DEĞİŞKENLER

`kod = 65918`

ÇAĞRI YIĞINI

`main()`

Kod İzleme: `"Ali".hashCode()`



65918 mod 13

= 8

Sıkıştırma: floorMod(65918, 13) = **8** → 13 kovalı tabloda kova 8.

Adım 12 / 12

```
1  // String.hashCode() ile aynı hesap:
2  int hash(String s) {
3    int h = 0;
4    for (int i = 0; i < s.length(); i++)
5      h = 31 * h + s.charAt(i);
6    return h;
7  }
8
9  // main:
10 int kod = hash("Ali");                  // 65918
▶  int indis = Math.floorMod(kod, 13);     // 8
```

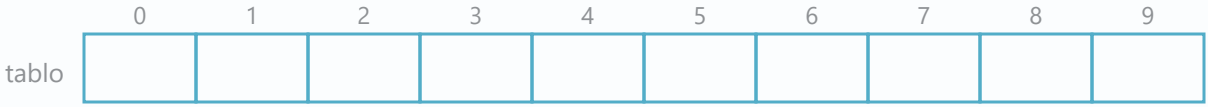
DEĞİŞKENLER

`kod = 65918` `indis = 8`

ÇAĞRI YIĞINI

`main()`

Kod İzleme: `modulusHash()`



10 hücrelik boş tablo; her hücre `null`.

Adım 1 / 16

```
1  int modulusHash(int anahtar, int diziBoyu) {
2    return Math.floorMod(anahtar, diziBoyu);
3  }
4
5  // main:
▶  Integer[] tablo = new Integer[10];
7  for (int k : new int[]{5, 1, 10, 26, 99}) {
8    int j = modulusHash(k, tablo.length);
9    tablo[j] = k;
10 }
```

DEĞİŞKENLER

`tablo.length = 10`

ÇAĞRI YIĞINI

`main()`

Kod İzleme: `modulusHash()`



Döngü sıradaki anahtarı alır: **k = 5**.

Adım 2 / 16

```
1  int modulusHash(int anahtar, int diziBoyu) {
2    return Math.floorMod(anahtar, diziBoyu);
3  }
4
5  // main:
6  Integer[] tablo = new Integer[10];
▶ for (int k : new int[]{5, 1, 10, 26, 99}) {
8    int j = modulusHash(k, tablo.length);
9    tablo[j] = k;
10 }
```

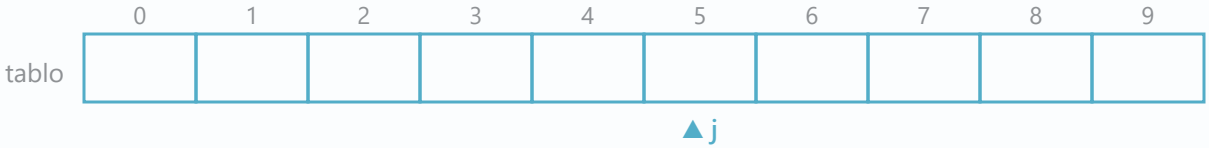
DEĞİŞKENLER

`k = 5`

ÇAĞRI YIĞINI

`main()`

Kod İzleme: `modulusHash()`



`modulusHash(5, 10)` : 5 mod 10 = **5** (bölümden kalan).

Adım 3 / 16

```
1  int modulusHash(int anahtar, int diziBoyu) {
▶    return Math.floorMod(anahtar, diziBoyu);
3  }
4
5  // main:
6  Integer[] tablo = new Integer[10];
7  for (int k : new int[]{5, 1, 10, 26, 99}) {
8    int j = modulusHash(k, tablo.length);
9    tablo[j] = k;
10 }
```

DEĞİŞKENLER

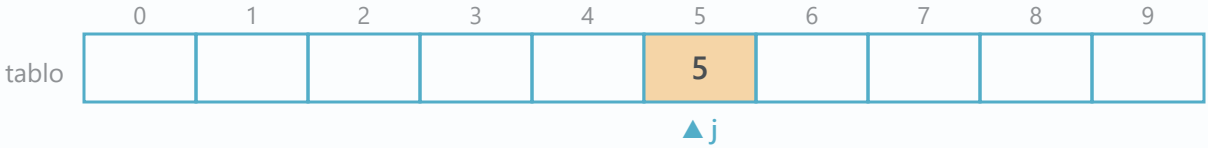
`anahtar = 5` `diziBoyu = 10`

ÇAĞRI YIĞINI

`modulusHash(5, 10)`

`main()`

Kod İzleme: `modulusHash()`



`tablo[5] = 5` .

Adım 4 / 16

```
1  int modulusHash(int anahtar, int diziBoyu) {
2    return Math.floorMod(anahtar, diziBoyu);
3  }
4
5  // main:
6  Integer[] tablo = new Integer[10];
7  for (int k : new int[]{5, 1, 10, 26, 99}) {
8    int j = modulusHash(k, tablo.length);
▶    tablo[j] = k;
10 }
```

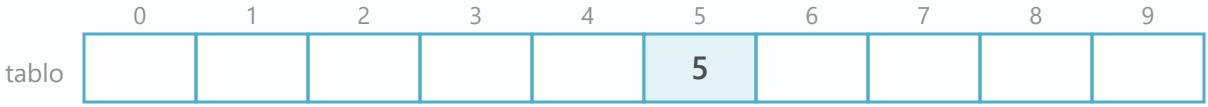
DEĞİŞKENLER

`k = 5` `j = 5`

ÇAĞRI YIĞINI

`main()`

Kod İzleme: `modulusHash()`



Döngü sıradaki anahtarı alır: **k = 1**.

Adım 5 / 16

```
1  int modulusHash(int anahtar, int diziBoyu) {
2    return Math.floorMod(anahtar, diziBoyu);
3  }
4
5  // main:
6  Integer[] tablo = new Integer[10];
▶ for (int k : new int[]{5, 1, 10, 26, 99}) {
8    int j = modulusHash(k, tablo.length);
9    tablo[j] = k;
10 }
```

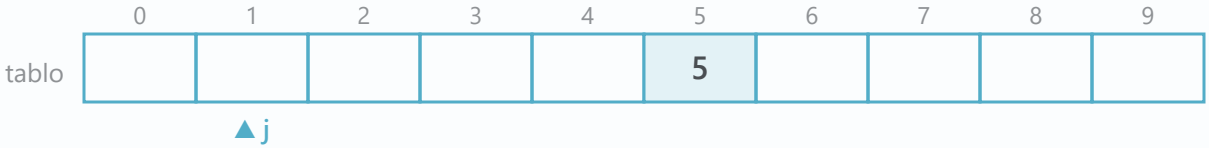
DEĞİŞKENLER

`k = 1`

ÇAĞRI YIĞINI

`main()`

Kod İzleme: `modulusHash()`



`modulusHash(1, 10)` : 1 mod 10 = **1** (bölümden kalan).

Adım 6 / 16

```
1  int modulusHash(int anahtar, int diziBoyu) {
▶    return Math.floorMod(anahtar, diziBoyu);
3  }
4
5  // main:
6  Integer[] tablo = new Integer[10];
7  for (int k : new int[]{5, 1, 10, 26, 99}) {
8    int j = modulusHash(k, tablo.length);
9    tablo[j] = k;
10 }
```

DEĞİŞKENLER

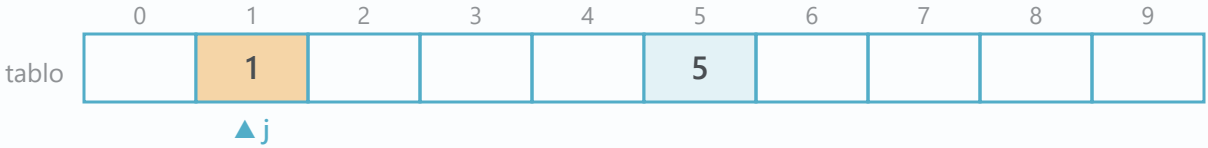
`anahtar = 1` `diziBoyu = 10`

ÇAĞRI YIĞINI

`modulusHash(1, 10)`

`main()`

Kod İzleme: `modulusHash()`



`tablo[1] = 1` .

Adım 7 / 16

```
1  int modulusHash(int anahtar, int diziBoyu) {
2    return Math.floorMod(anahtar, diziBoyu);
3  }
4
5  // main:
6  Integer[] tablo = new Integer[10];
7  for (int k : new int[]{5, 1, 10, 26, 99}) {
8    int j = modulusHash(k, tablo.length);
▶    tablo[j] = k;
10 }
```

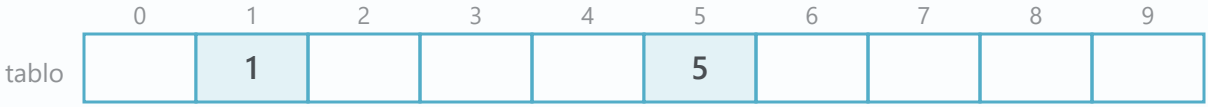
DEĞİŞKENLER

`k = 1` `j = 1`

ÇAĞRI YIĞINI

`main()`

Kod İzleme: `modulusHash()`



Döngü sıradaki anahtarı alır: **k = 10**.

Adım 8 / 16

```
1  int modulusHash(int anahtar, int diziBoyu) {
2    return Math.floorMod(anahtar, diziBoyu);
3  }
4
5  // main:
6  Integer[] tablo = new Integer[10];
▶ for (int k : new int[]{5, 1, 10, 26, 99}) {
8    int j = modulusHash(k, tablo.length);
9    tablo[j] = k;
10 }
```

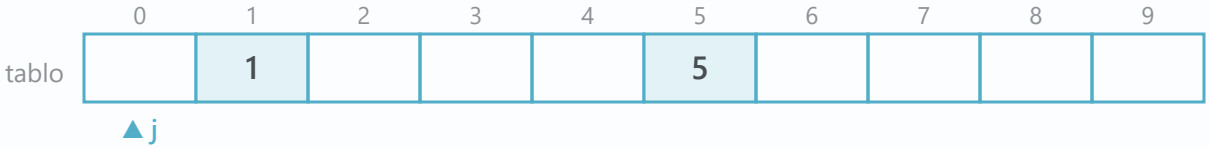
DEĞİŞKENLER

`k = 10`

ÇAĞRI YIĞINI

`main()`

Kod İzleme: `modulusHash()`



`modulusHash(10, 10)` : 10 mod 10 = **0** (bölümden kalan).

Adım 9 / 16

```
1  int modulusHash(int anahtar, int diziBoyu) {
▶    return Math.floorMod(anahtar, diziBoyu);
3  }
4
5  // main:
6  Integer[] tablo = new Integer[10];
7  for (int k : new int[]{5, 1, 10, 26, 99}) {
8    int j = modulusHash(k, tablo.length);
9    tablo[j] = k;
10 }
```

DEĞİŞKENLER

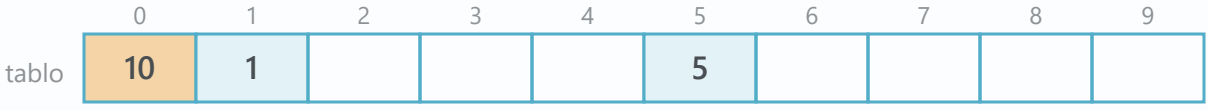
`anahtar = 10` `diziBoyu = 10`

ÇAĞRI YIĞINI

`modulusHash(10, 10)`

`main()`

Kod İzleme: `modulusHash()`



`tablo[0] = 10` .

Adım 10 / 16

```
1  int modulusHash(int anahtar, int diziBoyu) {
2    return Math.floorMod(anahtar, diziBoyu);
3  }
4
5  // main:
6  Integer[] tablo = new Integer[10];
7  for (int k : new int[]{5, 1, 10, 26, 99}) {
8    int j = modulusHash(k, tablo.length);
▶    tablo[j] = k;
10 }
```

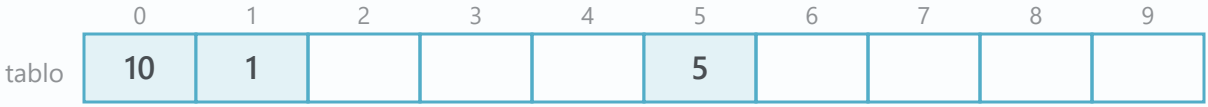
DEĞİŞKENLER

`k = 10` `j = 0`

ÇAĞRI YIĞINI

`main()`

Kod İzleme: `modulusHash()`



Döngü sıradaki anahtarı alır: **k = 26**.

Adım 11 / 16

```
1  int modulusHash(int anahtar, int diziBoyu) {
2    return Math.floorMod(anahtar, diziBoyu);
3  }
4
5  // main:
6  Integer[] tablo = new Integer[10];
▶ for (int k : new int[]{5, 1, 10, 26, 99}) {
8    int j = modulusHash(k, tablo.length);
9    tablo[j] = k;
10 }
```

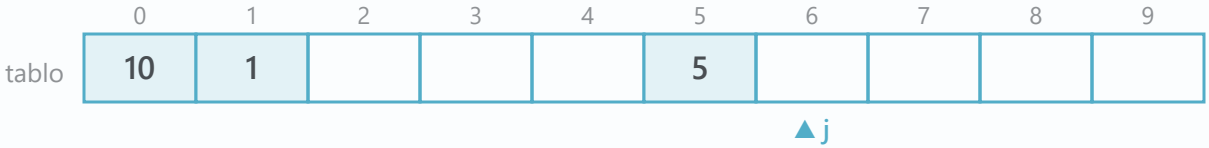
DEĞİŞKENLER

`k = 26`

ÇAĞRI YIĞINI

`main()`

Kod İzleme: `modulusHash()`



`modulusHash(26, 10)` : 26 mod 10 = **6** (bölümden kalan).

Adım 12 / 16

```
1  int modulusHash(int anahtar, int diziBoyu) {
▶    return Math.floorMod(anahtar, diziBoyu);
3  }
4
5  // main:
6  Integer[] tablo = new Integer[10];
7  for (int k : new int[]{5, 1, 10, 26, 99}) {
8    int j = modulusHash(k, tablo.length);
9    tablo[j] = k;
10 }
```

DEĞİŞKENLER

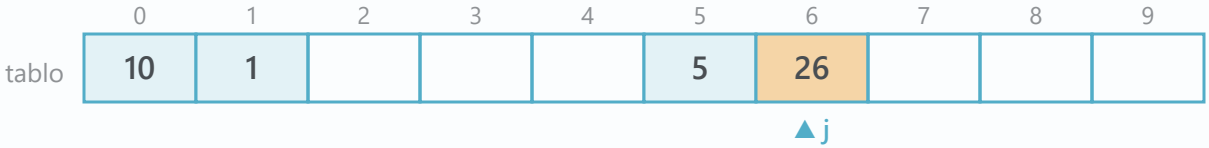
`anahtar = 26` `diziBoyu = 10`

ÇAĞRI YIĞINI

`modulusHash(26, 10)`

`main()`

Kod İzleme: `modulusHash()`



`tablo[6] = 26`.

Adım 13 / 16

```
1  int modulusHash(int anahtar, int diziBoyu) {
2    return Math.floorMod(anahtar, diziBoyu);
3  }
4
5  // main:
6  Integer[] tablo = new Integer[10];
7  for (int k : new int[]{5, 1, 10, 26, 99}) {
8    int j = modulusHash(k, tablo.length);
▶    tablo[j] = k;
10 }
```

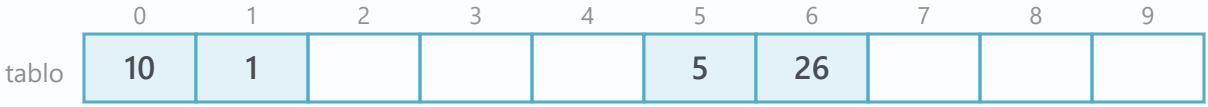
DEĞİŞKENLER

`k = 26` `j = 6`

ÇAĞRI YIĞINI

`main()`

Kod İzleme: `modulusHash()`



Döngü sıradaki anahtarı alır: **k = 99**.

Adım 14 / 16

```
1  int modulusHash(int anahtar, int diziBoyu) {
2    return Math.floorMod(anahtar, diziBoyu);
3  }
4
5  // main:
6  Integer[] tablo = new Integer[10];
▶ for (int k : new int[]{5, 1, 10, 26, 99}) {
8    int j = modulusHash(k, tablo.length);
9    tablo[j] = k;
10 }
```

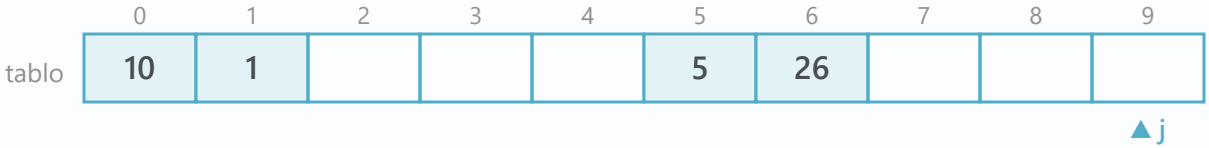
DEĞİŞKENLER

`k = 99`

ÇAĞRI YIĞINI

`main()`

Kod İzleme: `modulusHash()`



`modulusHash(99, 10)` : 99 mod 10 = **9** (bölümden kalan).

Adım 15 / 16

```
1  int modulusHash(int anahtar, int diziBoyu) {
▶    return Math.floorMod(anahtar, diziBoyu);
3  }
4
5  // main:
6  Integer[] tablo = new Integer[10];
7  for (int k : new int[]{5, 1, 10, 26, 99}) {
8    int j = modulusHash(k, tablo.length);
9    tablo[j] = k;
10 }
```

DEĞİŞKENLER

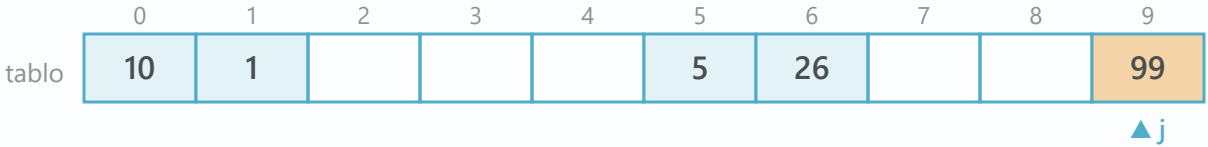
`anahtar = 99` `diziBoyu = 10`

ÇAĞRI YIĞINI

`modulusHash(99, 10)`

`main()`

Kod İzleme: `modulusHash()`



`tablo[9] = 99`. Beş anahtar, beş farklı hücre: çakışma yok. Ama 15 gelseydi 5 ile çakışırdı.

Adım 16 / 16

```
1  int modulusHash(int anahtar, int diziBoyu) {
2    return Math.floorMod(anahtar, diziBoyu);
3  }
4
5  // main:
6  Integer[] tablo = new Integer[10];
7  for (int k : new int[]{5, 1, 10, 26, 99}) {
8    int j = modulusHash(k, tablo.length);
▶    tablo[j] = k;
10 }
```

DEĞİŞKENLER

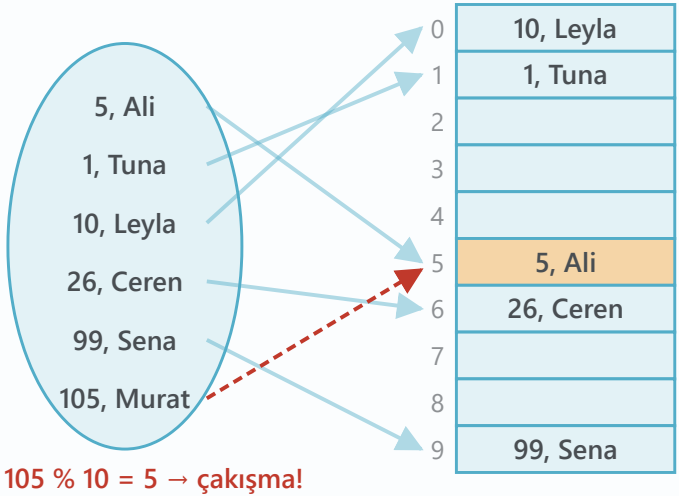
`k = 99` `j = 9`

ÇAĞRI YIĞINI

`main()`

Çakışmalar (Collisions)

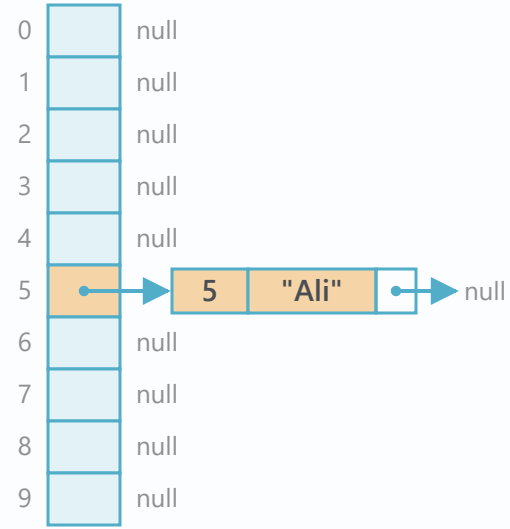
- Farklı anahtarlar aynı indisi ürettiğinde ortaya çıkar: h(5) = h(105) = 5.
- İki öğe aynı hücrede doğrudan saklanamaz.
- Anahtar uzayı tablodan büyük olduğu için çakışmalar **kaçınılmazdır**; bir çözüm stratejisi şarttır:



Ayrı zincirleme (separate chaining)
Her kova, o kovaya düşen öğelerin bağlı listesini tutar.

Açık adresleme (open addressing)
Dolu yuvaya gelen öğe için dizide başka boş yuva aranır.

Çakışma Çözümü: Ayrı Zincirleme

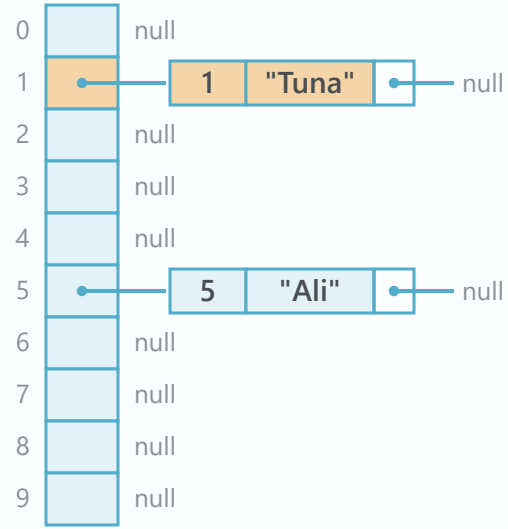


h(k) = k % 10

5 % 10 = 5

h(5) = 5 % 10 = **5** → kova 5 boş; (5, "Ali") düğümü kovaya bağlanır.

Çakışma Çözümü: Ayrı Zincirleme



HashDugumu[] kovalar

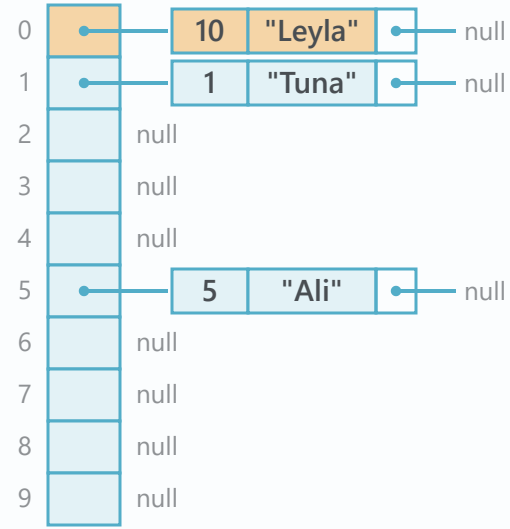
h(k) = k % 10

5 % 10 = 5

1 % 10 = 1

h(1) = 1 % 10 = **1** → kova 1 boş; (1, "Tuna") düğümü kovaya bağlanır.

Çakışma Çözümü: Ayrı Zincirleme



h(k) = k % 10

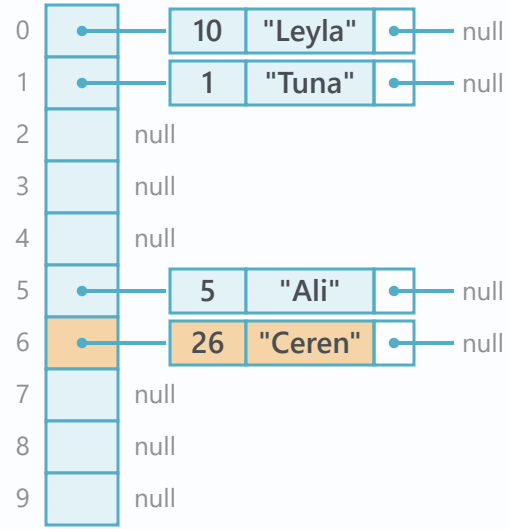
5 % 10 = 5

1 % 10 = 1

10 % 10 = 0

h(10) = 10 % 10 = **0** → kova 0 boş; (10, "Leyla") düğümü kovaya bağlanır.

Çakışma Çözümü: Ayrı Zincirleme



HashDugumu[] kovalar

h(k) = k % 10

5 % 10 = 5

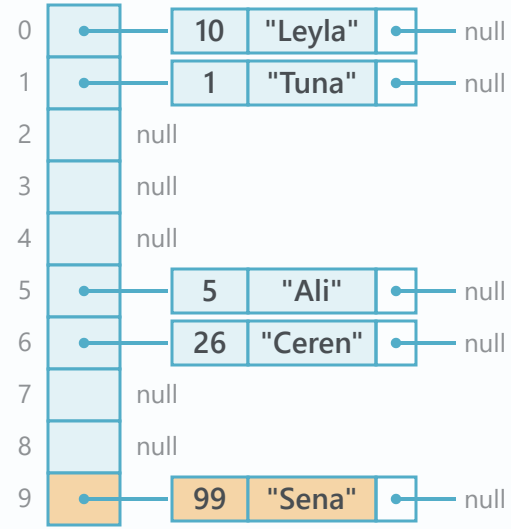
1 % 10 = 1

10 % 10 = 0

26 % 10 = 6

h(26) = 26 % 10 = **6** → kova 6 boş; (26, "Ceren") düğümü kovaya bağlanır.

Çakışma Çözümü: Ayrı Zincirleme



h(k) = k % 10

5 % 10 = 5

1 % 10 = 1

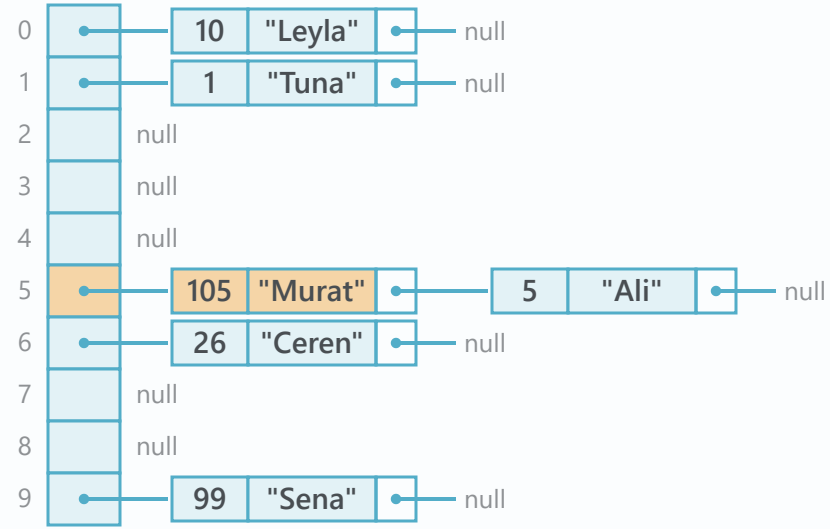
10 % 10 = 0

26 % 10 = 6

99 % 10 = 9

h(99) = 99 % 10 = **9** → kova 9 boş; (99, "Sena") düğümü kovaya bağlanır.

Çakışma Çözümü: Ayrı Zincirleme



h(k) = k % 10

5 % 10 = 5

1 % 10 = 1

10 % 10 = 0

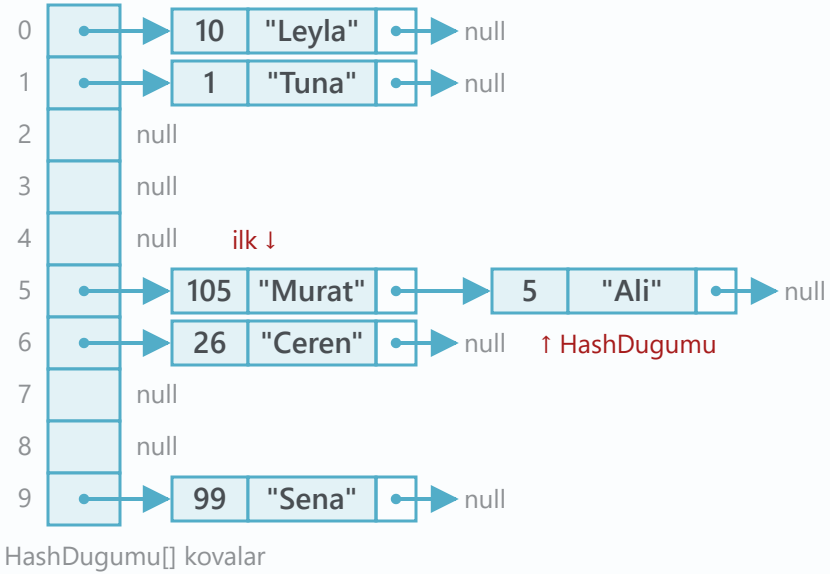
26 % 10 = 6

99 % 10 = 9

105 % 10 = 5

h(105) = 105 % 10 = **5** → kova 5 dolu (5, "Ali"): **çakışma**. Yeni düğüm zincirin **başına** eklenir.

Hash Tablosu Terminolojisi



kovalar

HashDugumu[] dizisi

kova (bucket)

dizinin bir hücresi

kovaSayisi = 10

dizi uzunluğu = kapasite

buyukluk = 6

saklanan çift sayısı

zincir

aynı kovadaki düğümler

ilk

zincirin ilk düğümü

Hash Tablosu Gerçeklemesi: Sınıf

```java
public class HashTablosu {
  private HashDugumu[] kovalar;   // zincirlerin ilk düğümleri
  private int kovaSayisi, buyukluk;

  public HashTablosu(int kapasite) {
    this.kovaSayisi = kapasite;
    this.kovalar = new HashDugumu[kapasite];
    this.buyukluk = 0;
  }
  private class HashDugumu {
    private Integer anahtar;
    private String deger;
    private HashDugumu sonraki;
    public HashDugumu(Integer anahtar, String deger) {
      this.anahtar = anahtar; this.deger = deger;
    }
  }
}
```

kovalar

Her hücre bir zincirin ilk düğümünü tutar.

HashDugumu

anahtar + değer + sonraki: tek yönlü bağlı liste düğümü.

new HashTablosu(10)

10 kova oluşur, hepsi `null`; buyukluk = 0.

AYRI ZİNCİRLEME · JAVA

Kova İndisi: getKovaIndisi()

```java
// Basit yol: anahtar % kovalar.length → negatif olabilir!
private int getKovaIndisi(Integer anahtar) {
  int h = anahtar.hashCode();  // = anahtar
  return Math.floorMod(h, kovalar.length);
}
```

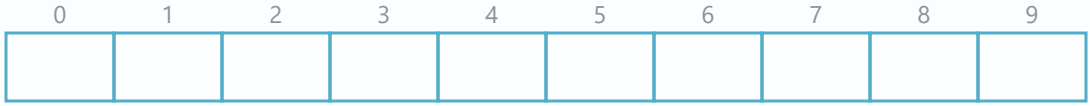
anahtar	anahtar % 10	Math.floorMod(anahtar, 10)	Sonuç
105	5	5	kova 5
21	1	1	kova 1
41	1	1	kova 1 (21 ile çakışır)
−7	−7	3	`%` ile `kovalar[-7]` → ArrayIndexOutOfBoundsException

Java'da `%` işleminin sonucu bölünenin işaretini alır. `Integer.hashCode()` sayının kendisidir; String gibi anahtarlarda `hashCode()` sıkça negatif olur. `Math.abs(h) % N` de güvenli değildir: `Math.abs(Integer.MIN_VALUE)` negatiftir.

Kod İzleme: `getKovaIndisi(41)`, `(-7)`

anahtar

41



`getKovaIndisi(41)` çağrılır; tabloda 10 kova var.

Adım 1 / 14

```
▶ private int getKovaIndisi(Integer anahtar) {
2   int h = anahtar.hashCode();  // = anahtar
3   return Math.floorMod(h, kovalar.length);
4 }
5
6 // floorMod'un n > 0 için eşdeğeri:
7 static int floorMod(int x, int n) {
8   int r = x % n;       // işaret x'ten gelir
9   if (r < 0) r += n;   // negatifse n ekle
10  return r;
11 }
```

DEĞİŞKENLER

`anahtar = 41` `kovalar.length = 10`

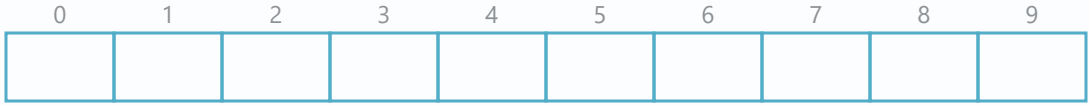
ÇAĞRI YIĞINI

`getKovaIndisi(41)`

Kod İzleme: `getKovaIndisi(41)`, `(-7)`

anahtar

41



`Integer.hashCode()` sayının kendisidir: **h = 41**.

Adım 2 / 14

```
1  private int getKovaIndisi(Integer anahtar) {
▶    int h = anahtar.hashCode();  // = anahtar
3    return Math.floorMod(h, kovalar.length);
4  }
5
6  // floorMod'un n > 0 için eşdeğeri:
7  static int floorMod(int x, int n) {
8    int r = x % n;       // işaret x'ten gelir
9    if (r < 0) r += n;   // negatifse n ekle
10   return r;
11 }
```

DEĞİŞKENLER

`anahtar = 41` `h = 41`

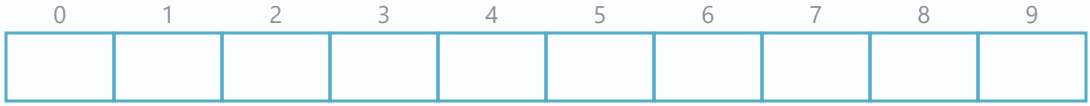
ÇAĞRI YIĞINI

`getKovaIndisi(41)`

Kod İzleme: `getKovaIndisi(41)`, `(-7)`

anahtar

41



`floorMod(41, 10)` çağrılır.

Adım 3 / 14

```
1  private int getKovaIndisi(Integer anahtar) {
2    int h = anahtar.hashCode();  // = anahtar
▶    return Math.floorMod(h, kovalar.length);
4  }
5
6  // floorMod'un n > 0 için eşdeğeri:
7  static int floorMod(int x, int n) {
8    int r = x % n;       // işaret x'ten gelir
9    if (r < 0) r += n;   // negatifse n ekle
10   return r;
11 }
```

DEĞİŞKENLER

`anahtar = 41` `h = 41`

ÇAĞRI YIĞINI

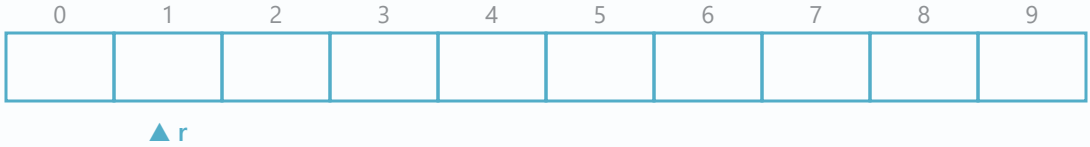
`floorMod(41, 10)`

`getKovaIndisi(41)`

Kod İzleme: `getKovaIndisi(41)`, `(-7)`

anahtar

41



Java'da **41 % 10 = 1**.

Adım 4 / 14

```
1  private int getKovaIndisi(Integer anahtar) {
2    int h = anahtar.hashCode();  // = anahtar
3    return Math.floorMod(h, kovalar.length);
4  }
5
6  // floorMod'un n > 0 için eşdeğeri:
7  static int floorMod(int x, int n) {
▶    int r = x % n;        // işaret x'ten gelir
9    if (r < 0) r += n;    // negatifse n ekle
10   return r;
11 }
```

DEĞİŞKENLER

`x = 41` `n = 10` `r = 1`

ÇAĞRI YIĞINI

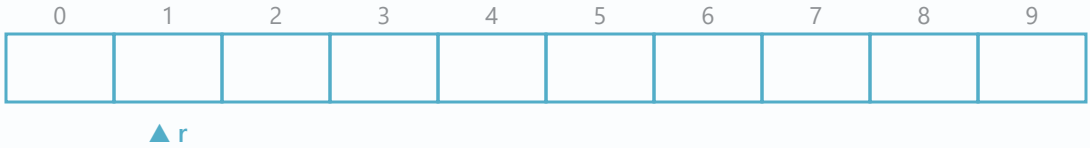
`floorMod(41, 10)`

`getKovaIndisi(41)`

Kod İzleme: `getKovaIndisi(41)`, `(-7)`

anahtar

41



r = 1 ≥ 0 → düzeltme gerekmez.

Adım 5 / 14

```
1  private int getKovaIndisi(Integer anahtar) {
2    int h = anahtar.hashCode();  // = anahtar
3    return Math.floorMod(h, kovalar.length);
4  }
5
6  // floorMod'un n > 0 için eşdeğeri:
7  static int floorMod(int x, int n) {
8    int r = x % n;       // işaret x'ten gelir
▶    if (r < 0) r += n;   // negatifse n ekle
10   return r;
11 }
```

DEĞİŞKENLER

`x = 41` `n = 10` `r = 1`

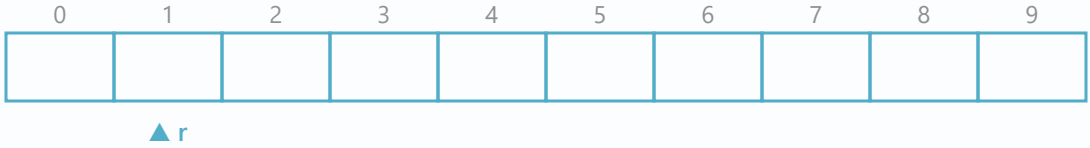
ÇAĞRI YIĞINI

- `floorMod(41, 10)`
- `getKovaIndisi(41)`

Kod İzleme: `getKovaIndisi(41)`, `(-7)`

anahtar

41



`return 1` .

Adım 6 / 14

```
1  private int getKovaIndisi(Integer anahtar) {
2    int h = anahtar.hashCode();  // = anahtar
3    return Math.floorMod(h, kovalar.length);
4  }
5
6  // floorMod'un n > 0 için eşdeğeri:
7  static int floorMod(int x, int n) {
8    int r = x % n;       // işaret x'ten gelir
9    if (r < 0) r += n;   // negatifse n ekle
▶    return r;
11 }
```

DEĞİŞKENLER

`x = 41` `n = 10` `r = 1`

ÇAĞRI YIĞINI

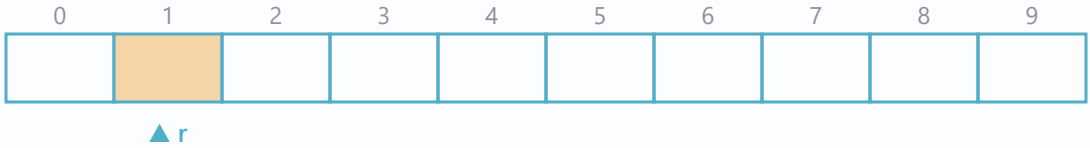
`floorMod(41, 10)`

`getKovaIndisi(41)`

Kod İzleme: `getKovaIndisi(41)`, `(-7)`

anahtar

41



Kova indisi **1**. (21 de kova 1'e düşer: çakışma.)

Adım 7 / 14

```
1  private int getKovaIndisi(Integer anahtar) {
2    int h = anahtar.hashCode();  // = anahtar
▶    return Math.floorMod(h, kovalar.length);
4  }
5
6  // floorMod'un n > 0 için eşdeğeri:
7  static int floorMod(int x, int n) {
8    int r = x % n;       // işaret x'ten gelir
9    if (r < 0) r += n;   // negatifse n ekle
10   return r;
11 }
```

DEĞİŞKENLER

`anahtar = 41` `h = 41`

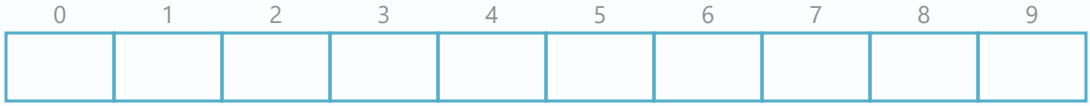
ÇAĞRI YIĞINI

`getKovaIndisi(41)`

Kod İzleme: `getKovaIndisi(41)`, `(-7)`

anahtar

-7



`getKovaIndisi(-7)` çağrılır; tabloda 10 kova var.

Adım 8 / 14

```
▶ private int getKovaIndisi(Integer anahtar) {
2   int h = anahtar.hashCode();  // = anahtar
3   return Math.floorMod(h, kovalar.length);
4 }
5
6 // floorMod'un n > 0 için eşdeğeri:
7 static int floorMod(int x, int n) {
8   int r = x % n;       // işaret x'ten gelir
9   if (r < 0) r += n;   // negatifse n ekle
10  return r;
11 }
```

DEĞİŞKENLER

`anahtar = -7` `kovalar.length = 10`

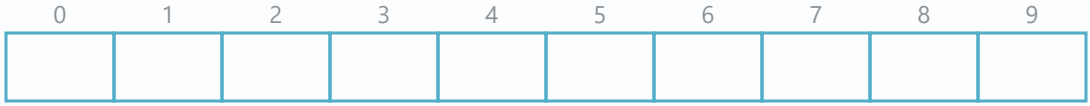
ÇAĞRI YIĞINI

`getKovaIndisi(-7)`

Kod İzleme: `getKovaIndisi(41)`, `(-7)`

anahtar

-7



`Integer.hashCode()` sayının kendisidir: **h = -7**.

Adım 9 / 14

```
1  private int getKovaIndisi(Integer anahtar) {
▶    int h = anahtar.hashCode();  // = anahtar
3    return Math.floorMod(h, kovalar.length);
4  }
5
6  // floorMod'un n > 0 için eşdeğeri:
7  static int floorMod(int x, int n) {
8    int r = x % n;       // işaret x'ten gelir
9    if (r < 0) r += n;   // negatifse n ekle
10   return r;
11 }
```

DEĞİŞKENLER

`anahtar = -7` `h = -7`

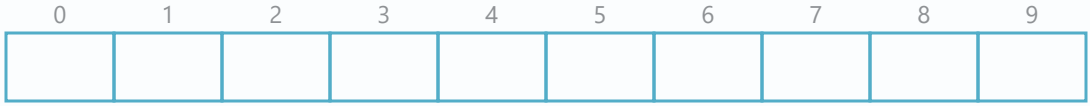
ÇAĞRI YIĞINI

`getKovaIndisi(-7)`

Kod İzleme: `getKovaIndisi(41)`, `(-7)`

anahtar

-7



`floorMod(-7, 10)` çağrılır.

Adım 10 / 14

```
1  private int getKovaIndisi(Integer anahtar) {
2    int h = anahtar.hashCode();  // = anahtar
▶    return Math.floorMod(h, kovalar.length);
4  }
5
6  // floorMod'un n > 0 için eşdeğeri:
7  static int floorMod(int x, int n) {
8    int r = x % n;       // işaret x'ten gelir
9    if (r < 0) r += n;   // negatifse n ekle
10   return r;
11 }
```

DEĞİŞKENLER

`anahtar = -7` `h = -7`

ÇAĞRI YIĞINI

`floorMod(-7, 10)`

`getKovaIndisi(-7)`

Kod İzleme: `getKovaIndisi(41)`, `(-7)`

anahtar

-7



◄ r = -7: kovalar[-7] dizinin dışında ✗

Java'da **-7 % 10 = -7**: sonuç bölünenin işaretini alır.

Adım 11 / 14

```
1  private int getKovaIndisi(Integer anahtar) {
2    int h = anahtar.hashCode();  // = anahtar
3    return Math.floorMod(h, kovalar.length);
4  }
5
6  // floorMod'un n > 0 için eşdeğeri:
7  static int floorMod(int x, int n) {
▶    int r = x % n;        // işaret x'ten gelir
9    if (r < 0) r += n;    // negatifse n ekle
10   return r;
11 }
```

DEĞİŞKENLER

`x = -7` `n = 10` `r = -7`

ÇAĞRI YIĞINI

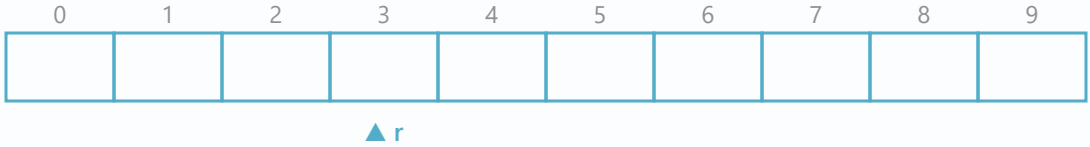
`floorMod(-7, 10)`

`getKovaIndisi(-7)`

Kod İzleme: `getKovaIndisi(41)`, `(-7)`

anahtar

-7



r < 0 → **r += 10** → r = 3. Artık 0 … 9 aralığında.

Adım 12 / 14

```
1  private int getKovaIndisi(Integer anahtar) {
2    int h = anahtar.hashCode();  // = anahtar
3    return Math.floorMod(h, kovalar.length);
4  }
5
6  // floorMod'un n > 0 için eşdeğeri:
7  static int floorMod(int x, int n) {
8    int r = x % n;       // işaret x'ten gelir
▶    if (r < 0) r += n;   // negatifse n ekle
10   return r;
11 }
```

DEĞİŞKENLER

`x = -7` `n = 10` `r = 3`

ÇAĞRI YIĞINI

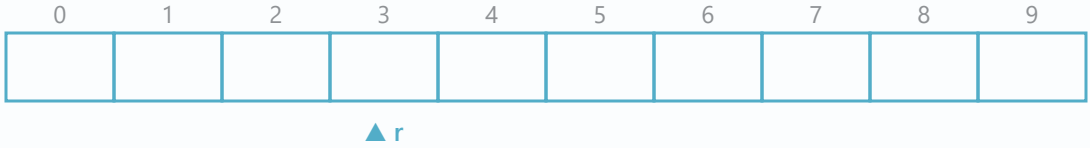
`floorMod(-7, 10)`

`getKovaIndisi(-7)`

Kod İzleme: `getKovaIndisi(41)`, `(-7)`

anahtar

-7



`return 3` .

Adım 13 / 14

```
1  private int getKovaIndisi(Integer anahtar) {
2    int h = anahtar.hashCode();  // = anahtar
3    return Math.floorMod(h, kovalar.length);
4  }
5
6  // floorMod'un n > 0 için eşdeğeri:
7  static int floorMod(int x, int n) {
8    int r = x % n;       // işaret x'ten gelir
9    if (r < 0) r += n;   // negatifse n ekle
▶    return r;
11 }
```

DEĞİŞKENLER

`x = -7` `n = 10` `r = 3`

ÇAĞRI YIĞINI

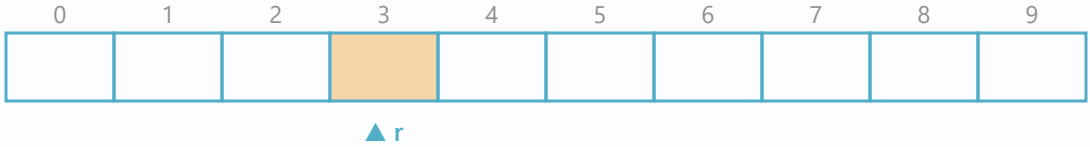
`floorMod(-7, 10)`

`getKovaIndisi(-7)`

Kod İzleme: `getKovaIndisi(41)`, `(-7)`

anahtar

-7



Kova indisi **3**. `%` kullanılsaydı `kovalar[-7]` → istisna.

Adım 14 / 14

```
1  private int getKovaIndisi(Integer anahtar) {
2    int h = anahtar.hashCode();  // = anahtar
▶    return Math.floorMod(h, kovalar.length);
4  }
5
6  // floorMod'un n > 0 için eşdeğeri:
7  static int floorMod(int x, int n) {
8    int r = x % n;       // işaret x'ten gelir
9    if (r < 0) r += n;   // negatifse n ekle
10   return r;
11 }
```

DEĞİŞKENLER

`anahtar = -7` `h = -7`

ÇAĞRI YIĞINI

`getKovaIndisi(-7)`

Anahtar-Değer İkilisi Yerleştirme

```java
public void yerlestir(Integer anahtar,
                      String deger) {
  int kovaIndisi = getKovaIndisi(anahtar);
  HashDugumu ilk = kovalar[kovaIndisi];
  while (ilk != null) {    // anahtar var mı?
    if (ilk.anahtar.equals(anahtar)) {
      ilk.deger = deger;   // varsa güncelle
      return;
    }
    ilk = ilk.sonraki;
  }
  buyukluk++;
  HashDugumu dugum =
      new HashDugumu(anahtar, deger);
  dugum.sonraki = kovalar[kovaIndisi];
  kovalar[kovaIndisi] = dugum;   // başa ekle
}
```

1. Kovayı bul

`h(anahtar)` ile O(1).

2. Zinciri tara

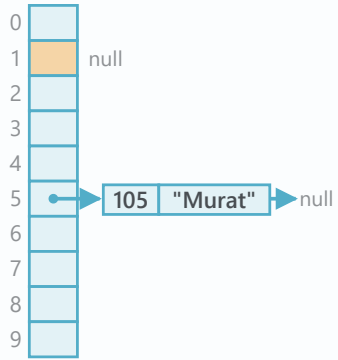
Anahtar varsa yalnızca değer değişir; buyukluk artmaz.

3. Başa ekle

Yoksa yeni düğüm zincirin başına eklenir: O(1).

Kod İzleme: `yerlestir(21)`, `yerlestir(41)`

```
1  public void yerlestir(Integer anahtar,
2                        String deger) {
3    int kovaIndisi = getKovaIndisi(anahtar);
4    HashDugumu ilk = kovalar[kovaIndisi];
5    while (ilk != null) {    // anahtar var mı?
6      if (ilk.anahtar.equals(anahtar)) {
7        ilk.deger = deger;   // varsa güncelle
8        return;
9      }
10     ilk = ilk.sonraki;
11   }
12   buyukluk++;
13   HashDugumu dugum =
14       new HashDugumu(anahtar, deger);
15   dugum.sonraki = kovalar[kovaIndisi];
16   kovalar[kovaIndisi] = dugum;   // başa ekle
17 }
```



`yerlestir(21, "Leyla")` : h(21) = 21 mod 10 = **1** → yalnızca kova 1'e bakılır.

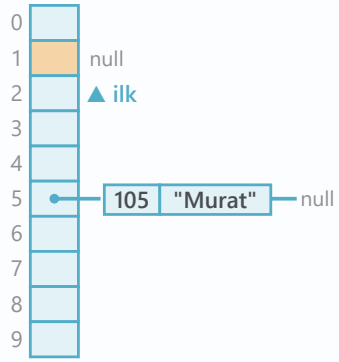
Adım 1 / 17

DEĞİŞKENLER

`anahtar = 21` `deger = "Leyla"` `kovaIndisi = 1` `ilk = –` `buyukluk = 1`

Kod İzleme: `yerlestir(21)`, `yerlestir(41)`

```
1  public void yerlestir(Integer anahtar,
2                        String deger) {
3    int kovaIndisi = getKovaIndisi(anahtar);
▶    HashDugumu ilk = kovalar[kovaIndisi];
5    while (ilk != null) {    // anahtar var mı?
6      if (ilk.anahtar.equals(anahtar)) {
7        ilk.deger = deger;   // varsa güncelle
8        return;
9      }
10     ilk = ilk.sonraki;
11   }
12   buyukluk++;
13   HashDugumu dugum =
14       new HashDugumu(anahtar, deger);
15   dugum.sonraki = kovalar[kovaIndisi];
16   kovalar[kovaIndisi] = dugum;   // başa ekle
17 }
```



Kova 1 boş: `ilk = null`.

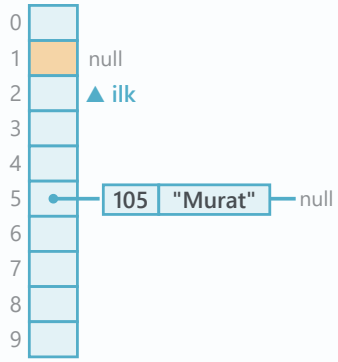
Adım 2 / 17

DEĞİŞKENLER

`anahtar = 21` `deger = "Leyla"` `kovaIndisi = 1` `ilk = null` `buyukluk = 1`

Kod İzleme: `yerlestir(21)`, `yerlestir(41)`

```
1  public void yerlestir(Integer anahtar,
2                        String deger) {
3    int kovaIndisi = getKovaIndisi(anahtar);
4    HashDugumu ilk = kovalar[kovaIndisi];
▶   while (ilk != null) {    // anahtar var mı?
6      if (ilk.anahtar.equals(anahtar)) {
7        ilk.deger = deger;   // varsa güncelle
8        return;
9      }
10     ilk = ilk.sonraki;
11   }
12   buyukluk++;
13   HashDugumu dugum =
14       new HashDugumu(anahtar, deger);
15   dugum.sonraki = kovalar[kovaIndisi];
16   kovalar[kovaIndisi] = dugum;   // başa ekle
17 }
```



`ilk == null` → döngü biter: 21 tabloda **yok**, yeni düğüm gerekir.

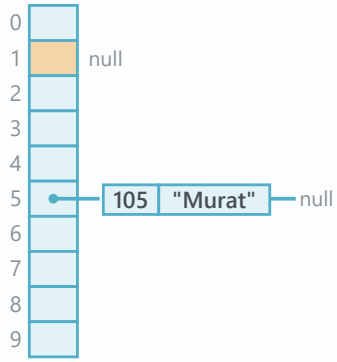
Adım 3 / 17

DEĞİŞKENLER

`anahtar = 21` `deger = "Leyla"` `kovaIndisi = 1` `ilk = null` `buyukluk = 1`

Kod İzleme: `yerlestir(21)`, `yerlestir(41)`

```
1  public void yerlestir(Integer anahtar,
2                        String deger) {
3    int kovaIndisi = getKovaIndisi(anahtar);
4    HashDugumu ilk = kovalar[kovaIndisi];
5    while (ilk != null) {    // anahtar var mı?
6      if (ilk.anahtar.equals(anahtar)) {
7        ilk.deger = deger;   // varsa güncelle
8        return;
9      }
10     ilk = ilk.sonraki;
11   }
▶  buyukluk++;
13   HashDugumu dugum =
14       new HashDugumu(anahtar, deger);
15   dugum.sonraki = kovalar[kovaIndisi];
16   kovalar[kovaIndisi] = dugum;   // başa ekle
17 }
```



Yeni çift eklenecek: **buyukluk = 2**.

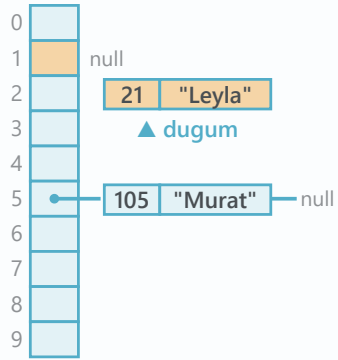
Adım 4 / 17

DEĞİŞKENLER

`anahtar = 21` `deger = "Leyla"` `kovaIndisi = 1` `ilk = null` `buyukluk = 2`

Kod İzleme: `yerlestir(21)`, `yerlestir(41)`

```
1  public void yerlestir(Integer anahtar,
2                        String deger) {
3    int kovaIndisi = getKovaIndisi(anahtar);
4    HashDugumu ilk = kovalar[kovaIndisi];
5    while (ilk != null) {    // anahtar var mı?
6      if (ilk.anahtar.equals(anahtar)) {
7        ilk.deger = deger;   // varsa güncelle
8        return;
9      }
10     ilk = ilk.sonraki;
11   }
12   buyukluk++;
▶    HashDugumu dugum =
▶        new HashDugumu(anahtar, deger);
15   dugum.sonraki = kovalar[kovaIndisi];
16   kovalar[kovaIndisi] = dugum;   // başa ekle
17 }
```



Yeni düğüm (21, "Leyla") oluşturulur; `sonraki` henüz null, tabloya bağlı değil.

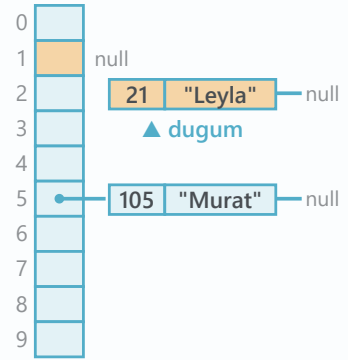
Adım 5 / 17

DEĞİŞKENLER

`anahtar = 21` `deger = "Leyla"` `kovaIndisi = 1` `ilk = null` `buyukluk = 2` `dugum = düğüm(21)`

Kod İzleme: `yerlestir(21)`, `yerlestir(41)`

```
1  public void yerlestir(Integer anahtar,
2                        String deger) {
3    int kovaIndisi = getKovaIndisi(anahtar);
4    HashDugumu ilk = kovalar[kovaIndisi];
5    while (ilk != null) {    // anahtar var mı?
6      if (ilk.anahtar.equals(anahtar)) {
7        ilk.deger = deger;   // varsa güncelle
8        return;
9      }
10     ilk = ilk.sonraki;
11   }
12   buyukluk++;
13   HashDugumu dugum =
14       new HashDugumu(anahtar, deger);
▶    dugum.sonraki = kovalar[kovaIndisi];
16   kovalar[kovaIndisi] = dugum;   // başa ekle
17 }
```



`dugum.sonraki = kovalar[1]` = null (kova boştu).

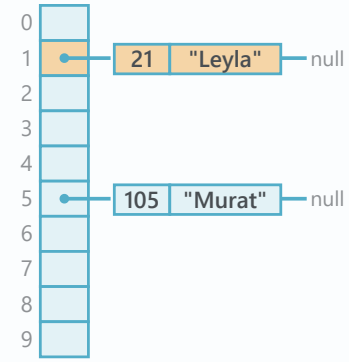
Adım 6 / 17

DEĞİŞKENLER

`anahtar = 21` `deger = "Leyla"` `kovaIndisi = 1` `ilk = null` `buyukluk = 2` `dugum = düğüm(21)`

Kod İzleme: `yerlestir(21)`, `yerlestir(41)`

```
1  public void yerlestir(Integer anahtar,
2                        String deger) {
3    int kovaIndisi = getKovaIndisi(anahtar);
4    HashDugumu ilk = kovalar[kovaIndisi];
5    while (ilk != null) {    // anahtar var mı?
6      if (ilk.anahtar.equals(anahtar)) {
7        ilk.deger = deger;   // varsa güncelle
8        return;
9      }
10     ilk = ilk.sonraki;
11   }
12   buyukluk++;
13   HashDugumu dugum =
14       new HashDugumu(anahtar, deger);
15   dugum.sonraki = kovalar[kovaIndisi];
▶    kovalar[kovaIndisi] = dugum;   // başa ekle
17 }
```



`kovalar[1] = dugum` : yeni düğüm zincirin **başında**. Sona gitmeye gerek yok → O(1).

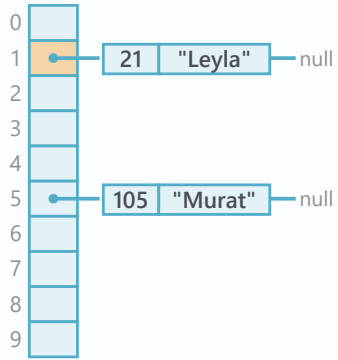
Adım 7 / 17

DEĞİŞKENLER

`anahtar = 21` `deger = "Leyla"`
`kovaIndisi = 1` `ilk = null`
`buyukluk = 2` `dugum = düğüm(21)`

Kod İzleme: `yerlestir(21)`, `yerlestir(41)`

```
1  public void yerlestir(Integer anahtar,
2                        String deger) {
3    int kovaIndisi = getKovaIndisi(anahtar);
4    HashDugumu ilk = kovalar[kovaIndisi];
5    while (ilk != null) {    // anahtar var mı?
6      if (ilk.anahtar.equals(anahtar)) {
7        ilk.deger = deger;   // varsa güncelle
8        return;
9      }
10     ilk = ilk.sonraki;
11   }
12   buyukluk++;
13   HashDugumu dugum =
14       new HashDugumu(anahtar, deger);
15   dugum.sonraki = kovalar[kovaIndisi];
16   kovalar[kovaIndisi] = dugum;   // başa ekle
17 }
```



`yerlestir(41, "Sena")` : h(41) = 41 mod 10 = **1** → yalnızca kova 1'e bakılır.

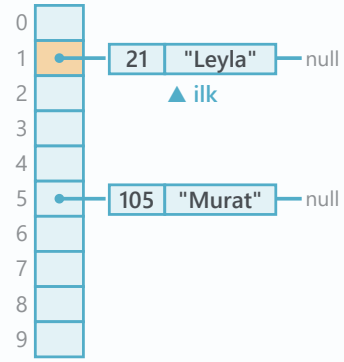
Adım 8 / 17

DEĞİŞKENLER

`anahtar = 41` `deger = "Sena"` `kovaIndisi = 1` `ilk = –` `buyukluk = 2`

Kod İzleme: `yerlestir(21)`, `yerlestir(41)`

```
1  public void yerlestir(Integer anahtar,
2                        String deger) {
3    int kovaIndisi = getKovaIndisi(anahtar);
▶    HashDugumu ilk = kovalar[kovaIndisi];
5    while (ilk != null) {    // anahtar var mı?
6      if (ilk.anahtar.equals(anahtar)) {
7        ilk.deger = deger;   // varsa güncelle
8        return;
9      }
10     ilk = ilk.sonraki;
11   }
12   buyukluk++;
13   HashDugumu dugum =
14       new HashDugumu(anahtar, deger);
15   dugum.sonraki = kovalar[kovaIndisi];
16   kovalar[kovaIndisi] = dugum;   // başa ekle
17 }
```



`ilk` zincirin ilk düğümünü gösterir.

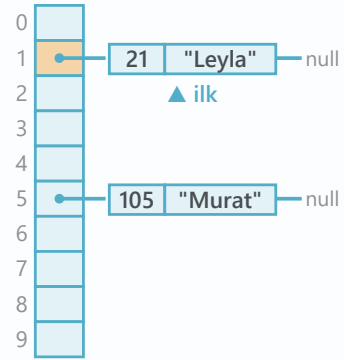
Adım 9 / 17

DEĞİŞKENLER

`anahtar = 41` `deger = "Sena"` `kovaIndisi = 1` `ilk = düğüm(21)` `buyukluk = 2`

Kod İzleme: `yerlestir(21)`, `yerlestir(41)`

```
1  public void yerlestir(Integer anahtar,
2                        String deger) {
3    int kovaIndisi = getKovaIndisi(anahtar);
4    HashDugumu ilk = kovalar[kovaIndisi];
▶    while (ilk != null) {    // anahtar var mı?
6      if (ilk.anahtar.equals(anahtar)) {
7        ilk.deger = deger;   // varsa güncelle
8        return;
9      }
10     ilk = ilk.sonraki;
11   }
12   buyukluk++;
13   HashDugumu dugum =
14       new HashDugumu(anahtar, deger);
15   dugum.sonraki = kovalar[kovaIndisi];
16   kovalar[kovaIndisi] = dugum;   // başa ekle
17 }
```



`ilk != null` → döngüye girilir, bu düğüm incelenecek.

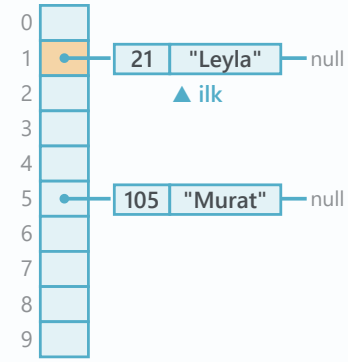
Adım 10 / 17

DEĞİŞKENLER

`anahtar = 41` `deger = "Sena"` `kovaIndisi = 1` `ilk = düğüm(21)` `buyukluk = 2`

Kod İzleme: `yerlestir(21)`, `yerlestir(41)`

```
1  public void yerlestir(Integer anahtar,
2                        String deger) {
3    int kovaIndisi = getKovaIndisi(anahtar);
4    HashDugumu ilk = kovalar[kovaIndisi];
5    while (ilk != null) {    // anahtar var mı?
▶      if (ilk.anahtar.equals(anahtar)) {
7        ilk.deger = deger;   // varsa güncelle
8        return;
9      }
10     ilk = ilk.sonraki;
11   }
12   buyukluk++;
13   HashDugumu dugum =
14       new HashDugumu(anahtar, deger);
15   dugum.sonraki = kovalar[kovaIndisi];
16   kovalar[kovaIndisi] = dugum;   // başa ekle
17 }
```



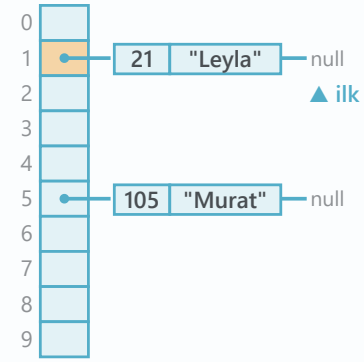
21 ≠ 41 → eşleşme yok, zincirde ilerle.

DEĞİŞKENLER

`anahtar` = **41** | `deger` = **"Sena"** | `kovaIndisi` = **1** | `ilk` = **düğüm(21)** | `buyukluk` = **2**

Kod İzleme: `yerlestir(21)`, `yerlestir(41)`

```
1  public void yerlestir(Integer anahtar,
2                        String deger) {
3    int kovaIndisi = getKovaIndisi(anahtar);
4    HashDugumu ilk = kovalar[kovaIndisi];
5    while (ilk != null) {    // anahtar var mı?
6      if (ilk.anahtar.equals(anahtar)) {
7        ilk.deger = deger;   // varsa güncelle
8        return;
9      }
▶      ilk = ilk.sonraki;
11   }
12   buyukluk++;
13   HashDugumu dugum =
14       new HashDugumu(anahtar, deger);
15   dugum.sonraki = kovalar[kovaIndisi];
16   kovalar[kovaIndisi] = dugum;   // başa ekle
17 }
```



`ilk = ilk.sonraki` → null (zincir bitti).

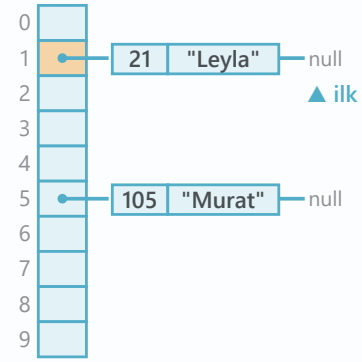
Adım 12 / 17

DEĞİŞKENLER

`anahtar = 41` `deger = "Sena"` `kovaIndisi = 1` `ilk = null` `buyukluk = 2`

Kod İzleme: `yerlestir(21)`, `yerlestir(41)`

```
1  public void yerlestir(Integer anahtar,
2                        String deger) {
3    int kovaIndisi = getKovaIndisi(anahtar);
4    HashDugumu ilk = kovalar[kovaIndisi];
▶    while (ilk != null) {    // anahtar var mı?
6      if (ilk.anahtar.equals(anahtar)) {
7        ilk.deger = deger;   // varsa güncelle
8        return;
9      }
10     ilk = ilk.sonraki;
11   }
12   buyukluk++;
13   HashDugumu dugum =
14       new HashDugumu(anahtar, deger);
15   dugum.sonraki = kovalar[kovaIndisi];
16   kovalar[kovaIndisi] = dugum;   // başa ekle
17 }
```



`ilk == null` → döngü biter: 41 tabloda **yok**, yeni düğüm gerekir.

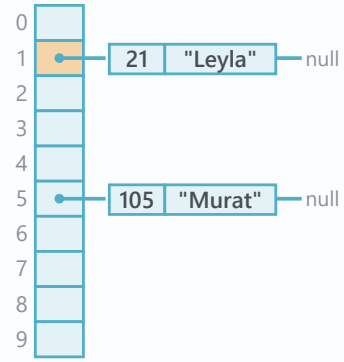
Adım 13 / 17

DEĞİŞKENLER

`anahtar = 41` `deger = "Sena"` `kovaIndisi = 1` `ilk = null` `buyukluk = 2`

Kod İzleme: `yerlestir(21)`, `yerlestir(41)`

```
1  public void yerlestir(Integer anahtar,
2                        String deger) {
3    int kovaIndisi = getKovaIndisi(anahtar);
4    HashDugumu ilk = kovalar[kovaIndisi];
5    while (ilk != null) {    // anahtar var mı?
6      if (ilk.anahtar.equals(anahtar)) {
7        ilk.deger = deger;   // varsa güncelle
8        return;
9      }
10     ilk = ilk.sonraki;
11   }
▶  buyukluk++;
13   HashDugumu dugum =
14       new HashDugumu(anahtar, deger);
15   dugum.sonraki = kovalar[kovaIndisi];
16   kovalar[kovaIndisi] = dugum;   // başa ekle
17 }
```



Yeni çift eklenecek: **buyukluk = 3**.

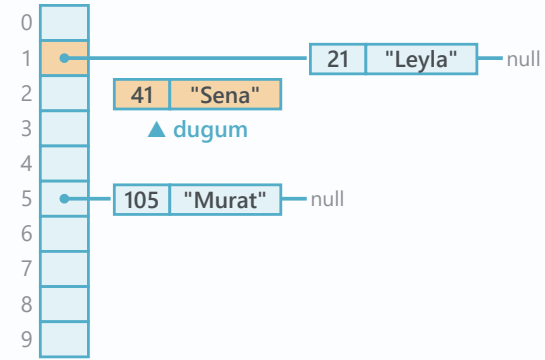
Adım 14 / 17

DEĞİŞKENLER

`anahtar = 41` `deger = "Sena"` `kovaIndisi = 1` `ilk = null` `buyukluk = 3`

Kod İzleme: `yerlestir(21)`, `yerlestir(41)`

```
1  public void yerlestir(Integer anahtar,
2                        String deger) {
3    int kovaIndisi = getKovaIndisi(anahtar);
4    HashDugumu ilk = kovalar[kovaIndisi];
5    while (ilk != null) {    // anahtar var mı?
6      if (ilk.anahtar.equals(anahtar)) {
7        ilk.deger = deger;   // varsa güncelle
8        return;
9      }
10     ilk = ilk.sonraki;
11   }
12   buyukluk++;
▶    HashDugumu dugum =
▶        new HashDugumu(anahtar, deger);
15   dugum.sonraki = kovalar[kovaIndisi];
16   kovalar[kovaIndisi] = dugum;   // başa ekle
17 }
```



Yeni düğüm (41, "Sena") oluşturulur; `sonraki` henüz null, tabloya bağlı değil.

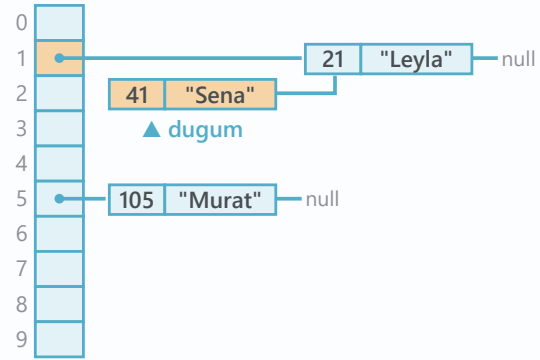
Adım 15 / 17

DEĞİŞKENLER

`anahtar = 41` `deger = "Sena"` `kovaIndisi = 1` `ilk = null` `buyukluk = 3` `dugum = düğüm(41)`

Kod İzleme: `yerlestir(21)`, `yerlestir(41)`

```
1  public void yerlestir(Integer anahtar,
2                        String deger) {
3    int kovaIndisi = getKovaIndisi(anahtar);
4    HashDugumu ilk = kovalar[kovaIndisi];
5    while (ilk != null) {    // anahtar var mı?
6      if (ilk.anahtar.equals(anahtar)) {
7        ilk.deger = deger;   // varsa güncelle
8        return;
9      }
10     ilk = ilk.sonraki;
11   }
12   buyukluk++;
13   HashDugumu dugum =
14       new HashDugumu(anahtar, deger);
▶    dugum.sonraki = kovalar[kovaIndisi];
16   kovalar[kovaIndisi] = dugum;   // başa ekle
17 }
```



`dugum.sonraki` eski baş düğümü (21) gösterir; mevcut zincir kaybolmaz.

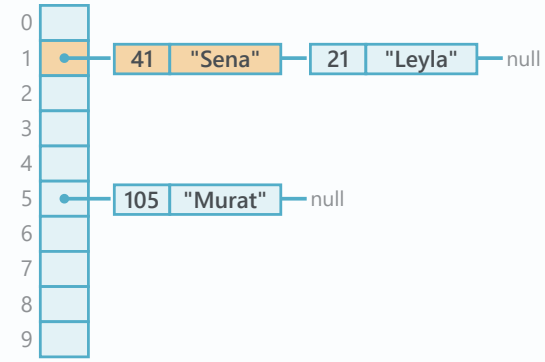
Adım 16 / 17

DEĞİŞKENLER

`anahtar = 41` `deger = "Sena"` `kovaIndisi = 1` `ilk = null` `buyukluk = 3` `dugum = düğüm(41)`

Kod İzleme: `yerlestir(21)`, `yerlestir(41)`

```
1  public void yerlestir(Integer anahtar,
2                        String deger) {
3    int kovaIndisi = getKovaIndisi(anahtar);
4    HashDugumu ilk = kovalar[kovaIndisi];
5    while (ilk != null) {    // anahtar var mı?
6      if (ilk.anahtar.equals(anahtar)) {
7        ilk.deger = deger;   // varsa güncelle
8        return;
9      }
10     ilk = ilk.sonraki;
11   }
12   buyukluk++;
13   HashDugumu dugum =
14       new HashDugumu(anahtar, deger);
15   dugum.sonraki = kovalar[kovaIndisi];
▶    kovalar[kovaIndisi] = dugum;   // başa ekle
17 }
```



`kovalar[1] = dugum` : yeni düğüm zincirin **başında**. Sona gitmeye gerek yok → O(1).

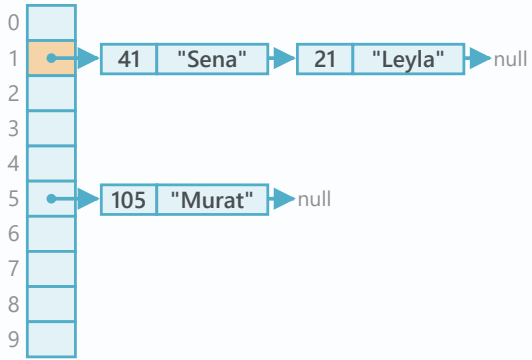
Adım 17 / 17

DEĞİŞKENLER

`anahtar = 41` `deger = "Sena"` `kovaIndisi = 1` `ilk = null` `buyukluk = 3` `dugum = düğüm(41)`

Kod İzleme: `yerlestir(21, "Ali")` (güncelleme)

```
1  public void yerlestir(Integer anahtar,
2                        String deger) {
3    int kovaIndisi = getKovaIndisi(anahtar);
4    HashDugumu ilk = kovalar[kovaIndisi];
5    while (ilk != null) {    // anahtar var mı?
6      if (ilk.anahtar.equals(anahtar)) {
7        ilk.deger = deger;   // varsa güncelle
8        return;
9      }
10     ilk = ilk.sonraki;
11   }
12   buyukluk++;
13   HashDugumu dugum =
14       new HashDugumu(anahtar, deger);
15   dugum.sonraki = kovalar[kovaIndisi];
16   kovalar[kovaIndisi] = dugum;   // başa ekle
17 }
```



`yerlestir(21, "Ali")` : h(21) = 21 mod 10 = **1** → yalnızca kova 1'e bakılır.

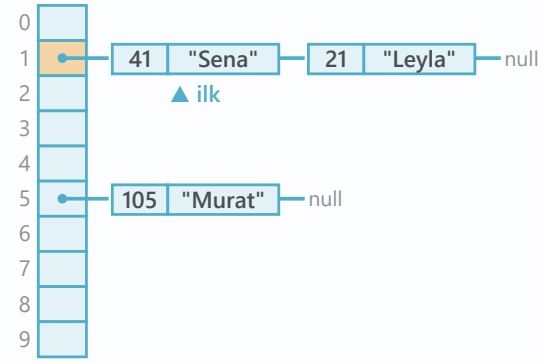
Adım 1 / 9

DEĞİŞKENLER

`anahtar = 21` `deger = "Ali"` `kovaIndisi = 1` `ilk = –` `buyukluk = 3`

Kod İzleme: `yerlestir(21, "Ali")` (güncelleme)

```
1  public void yerlestir(Integer anahtar,
2                        String deger) {
3    int kovaIndisi = getKovaIndisi(anahtar);
▶    HashDugumu ilk = kovalar[kovaIndisi];
5    while (ilk != null) {    // anahtar var mı?
6      if (ilk.anahtar.equals(anahtar)) {
7        ilk.deger = deger;   // varsa güncelle
8        return;
9      }
10     ilk = ilk.sonraki;
11   }
12   buyukluk++;
13   HashDugumu dugum =
14       new HashDugumu(anahtar, deger);
15   dugum.sonraki = kovalar[kovaIndisi];
16   kovalar[kovaIndisi] = dugum;   // başa ekle
17 }
```



`ilk` zincirin ilk düğümünü gösterir.

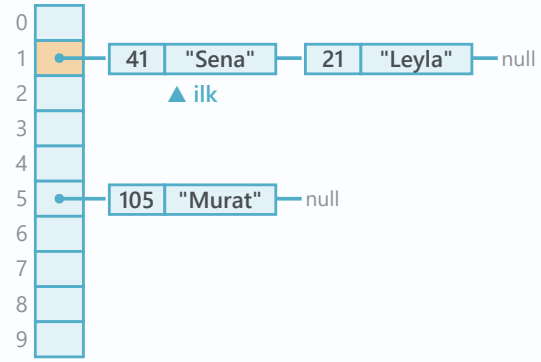
Adım 2 / 9

DEĞİŞKENLER

`anahtar = 21` `deger = "Ali"` `kovaIndisi = 1` `ilk = düğüm(41)` `buyukluk = 3`

Kod İzleme: `yerlestir(21, "Ali")` (güncelleme)

```
1  public void yerlestir(Integer anahtar,
2                        String deger) {
3    int kovaIndisi = getKovaIndisi(anahtar);
4    HashDugumu ilk = kovalar[kovaIndisi];
▶   while (ilk != null) {    // anahtar var mı?
6      if (ilk.anahtar.equals(anahtar)) {
7        ilk.deger = deger;   // varsa güncelle
8        return;
9      }
10     ilk = ilk.sonraki;
11   }
12   buyukluk++;
13   HashDugumu dugum =
14       new HashDugumu(anahtar, deger);
15   dugum.sonraki = kovalar[kovaIndisi];
16   kovalar[kovaIndisi] = dugum;   // başa ekle
17 }
```



`ilk != null` → döngüye girilir, bu düğüm incelenecek.

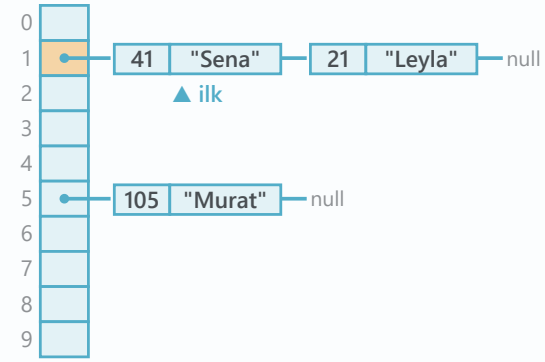
Adım 3 / 9

DEĞİŞKENLER

`anahtar = 21` `deger = "Ali"` `kovaIndisi = 1` `ilk = düğüm(41)` `buyukluk = 3`

Kod İzleme: `yerlestir(21, "Ali")` (güncelleme)

```
1  public void yerlestir(Integer anahtar,
2                        String deger) {
3    int kovaIndisi = getKovaIndisi(anahtar);
4    HashDugumu ilk = kovalar[kovaIndisi];
5    while (ilk != null) {    // anahtar var mı?
▶      if (ilk.anahtar.equals(anahtar)) {
7        ilk.deger = deger;   // varsa güncelle
8        return;
9      }
10     ilk = ilk.sonraki;
11   }
12   buyukluk++;
13   HashDugumu dugum =
14       new HashDugumu(anahtar, deger);
15   dugum.sonraki = kovalar[kovaIndisi];
16   kovalar[kovaIndisi] = dugum;   // başa ekle
17 }
```



41 ≠ 21 → eşleşme yok, zincirde ilerle.

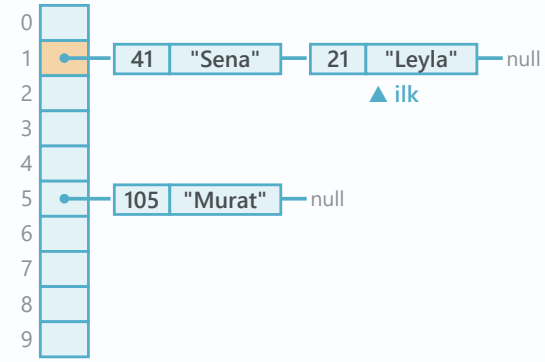
Adım 4 / 9

DEĞİŞKENLER

`anahtar = 21` `deger = "Ali"` `kovaIndisi = 1` `ilk = düğüm(41)` `buyukluk = 3`

Kod İzleme: `yerlestir(21, "Ali")` (güncelleme)

```
1  public void yerlestir(Integer anahtar,
2                        String deger) {
3    int kovaIndisi = getKovaIndisi(anahtar);
4    HashDugumu ilk = kovalar[kovaIndisi];
5    while (ilk != null) {    // anahtar var mı?
6      if (ilk.anahtar.equals(anahtar)) {
7        ilk.deger = deger;   // varsa güncelle
8        return;
9      }
▶      ilk = ilk.sonraki;
11   }
12   buyukluk++;
13   HashDugumu dugum =
14       new HashDugumu(anahtar, deger);
15   dugum.sonraki = kovalar[kovaIndisi];
16   kovalar[kovaIndisi] = dugum;   // başa ekle
17 }
```



`ilk = ilk.sonraki` → düğüm 21.

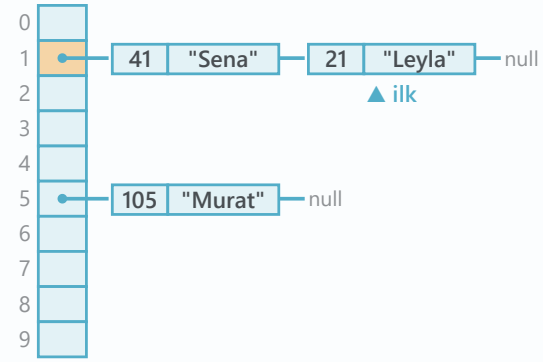
Adım 5 / 9

DEĞİŞKENLER

`anahtar = 21` `deger = "Ali"` `kovaIndisi = 1` `ilk = düğüm(21)` `buyukluk = 3`

Kod İzleme: `yerlestir(21, "Ali")` (güncelleme)

```
1  public void yerlestir(Integer anahtar,
2                        String deger) {
3    int kovaIndisi = getKovaIndisi(anahtar);
4    HashDugumu ilk = kovalar[kovaIndisi];
▶    while (ilk != null) {    // anahtar var mı?
6      if (ilk.anahtar.equals(anahtar)) {
7        ilk.deger = deger;   // varsa güncelle
8        return;
9      }
10     ilk = ilk.sonraki;
11   }
12   buyukluk++;
13   HashDugumu dugum =
14       new HashDugumu(anahtar, deger);
15   dugum.sonraki = kovalar[kovaIndisi];
16   kovalar[kovaIndisi] = dugum;   // başa ekle
17 }
```



`ilk != null` → döngüye girilir, bu düğüm incelenecek.

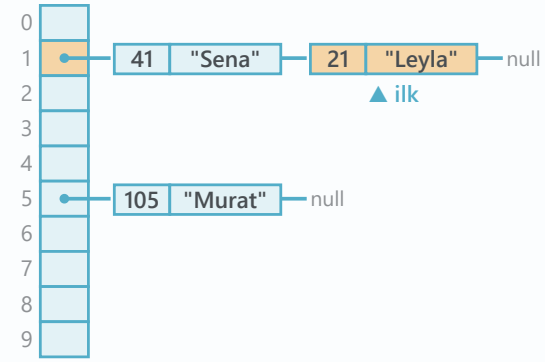
Adım 6 / 9

DEĞİŞKENLER

`anahtar = 21` `deger = "Ali"` `kovaIndisi = 1` `ilk = düğüm(21)` `buyukluk = 3`

Kod İzleme: `yerlestir(21, "Ali")` (güncelleme)

```
1  public void yerlestir(Integer anahtar,
2                        String deger) {
3    int kovaIndisi = getKovaIndisi(anahtar);
4    HashDugumu ilk = kovalar[kovaIndisi];
5    while (ilk != null) {    // anahtar var mı?
▶      if (ilk.anahtar.equals(anahtar)) {
7        ilk.deger = deger;   // varsa güncelle
8        return;
9      }
10     ilk = ilk.sonraki;
11   }
12   buyukluk++;
13   HashDugumu dugum =
14       new HashDugumu(anahtar, deger);
15   dugum.sonraki = kovalar[kovaIndisi];
16   kovalar[kovaIndisi] = dugum;   // başa ekle
17 }
```



`21.equals(21)` → **eşit**: anahtar zaten tabloda.

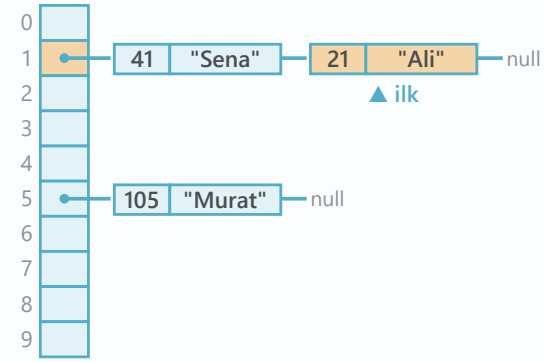
Adım 7 / 9

DEĞİŞKENLER

`anahtar = 21` `deger = "Ali"` `kovaIndisi = 1` `ilk = düğüm(21)` `buyukluk = 3`

Kod İzleme: `yerlestir(21, "Ali")` (güncelleme)

```
1  public void yerlestir(Integer anahtar,
2                        String deger) {
3    int kovaIndisi = getKovaIndisi(anahtar);
4    HashDugumu ilk = kovalar[kovaIndisi];
5    while (ilk != null) {    // anahtar var mı?
6      if (ilk.anahtar.equals(anahtar)) {
▶        ilk.deger = deger;   // varsa güncelle
8        return;
9      }
10     ilk = ilk.sonraki;
11   }
12   buyukluk++;
13   HashDugumu dugum =
14       new HashDugumu(anahtar, deger);
15   dugum.sonraki = kovalar[kovaIndisi];
16   kovalar[kovaIndisi] = dugum;   // başa ekle
17 }
```



Değer güncellenir: "Leyla" → **"Ali"**. Yeni düğüm eklenmez.

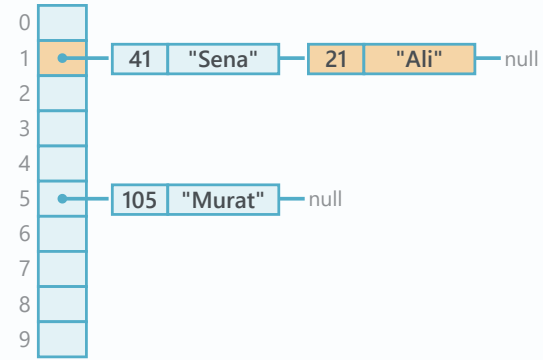
Adım 8 / 9

DEĞİŞKENLER

`anahtar = 21` `deger = "Ali"` `kovaIndisi = 1` `ilk = düğüm(21)` `buyukluk = 3`

Kod İzleme: `yerlestir(21, "Ali")` (güncelleme)

```
1  public void yerlestir(Integer anahtar,
2                        String deger) {
3    int kovaIndisi = getKovaIndisi(anahtar);
4    HashDugumu ilk = kovalar[kovaIndisi];
5    while (ilk != null) {    // anahtar var mı?
6      if (ilk.anahtar.equals(anahtar)) {
7        ilk.deger = deger;   // varsa güncelle
▶        return;
9      }
10     ilk = ilk.sonraki;
11   }
12   buyukluk++;
13   HashDugumu dugum =
14       new HashDugumu(anahtar, deger);
15   dugum.sonraki = kovalar[kovaIndisi];
16   kovalar[kovaIndisi] = dugum;   // başa ekle
17 }
```



`return` : **buyukluk = 3** değişmez (çift sayısı aynı).

Adım 9 / 9

DEĞİŞKENLER

`anahtar = 21` `deger = "Ali"` `kovaIndisi = 1` `ilk = düğüm(21)` `buyukluk = 3`

Anahtar ile Değer Getirme

```java
public String getir(Integer anahtar) {
  int kovaIndisi = getKovaIndisi(anahtar);
  HashDugumu ilk = kovalar[kovaIndisi];
  while (ilk != null) {
    if (ilk.anahtar.equals(anahtar))
      return ilk.deger;          // bulundu
    ilk = ilk.sonraki;
  }
  return null;                   // anahtar yok
}
```

Yalnızca bir kova

Diğer kovalara hiç bakılmaz.

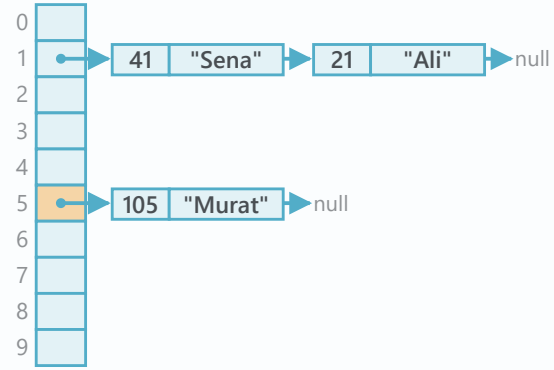
Maliyet

O(1 + zincir uzunluğu). Ortalama zincir uzunluğu = n/N.

Çağrı	Kova	İncelenen düğümler	Dönüş
`tablo.getir(105)`	105 % 10 = 5	105	"Murat"
`tablo.getir(21)`	21 % 10 = 1	41, 21	"Ali"
`tablo.getir(88)`	88 % 10 = 8	— (kova boş)	`null`

Kod İzleme: `getir(105)`, `getir(21)`, `getir(88)`

```
1  public String getir(Integer anahtar) {
2    int kovaIndisi = getKovaIndisi(anahtar);
3    HashDugumu ilk = kovalar[kovaIndisi];
4    while (ilk != null) {
5      if (ilk.anahtar.equals(anahtar))
6        return ilk.deger;          // bulundu
7      ilk = ilk.sonraki;
8    }
9    return null;                   // anahtar yok
10 }
```



`getir(105)` : h(105) = 105 mod 10 = **5**.
Diğer kovalara hiç bakılmaz.

Adım 1 / 17

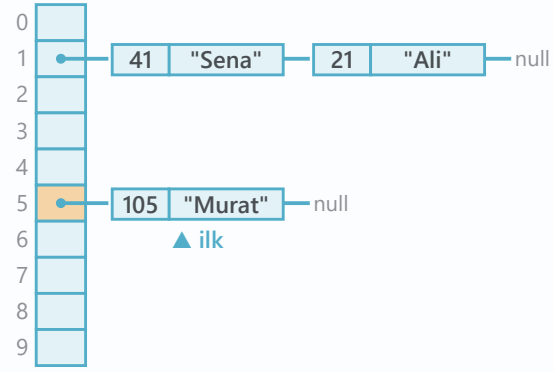
DEĞİŞKENLER

`anahtar = 105` `kovaIndisi = 5`

`ilk = –` `karşılaştırma = 0`

Kod İzleme: `getir(105)`, `getir(21)`, `getir(88)`

```
1  public String getir(Integer anahtar) {
2    int kovaIndisi = getKovaIndisi(anahtar);
▶    HashDugumu ilk = kovalar[kovaIndisi];
4    while (ilk != null) {
5      if (ilk.anahtar.equals(anahtar))
6        return ilk.deger;          // bulundu
7      ilk = ilk.sonraki;
8    }
9    return null;                   // anahtar yok
10 }
```



`ilk` kovanın ilk düğümünü gösterir.

Adım 2 / 17

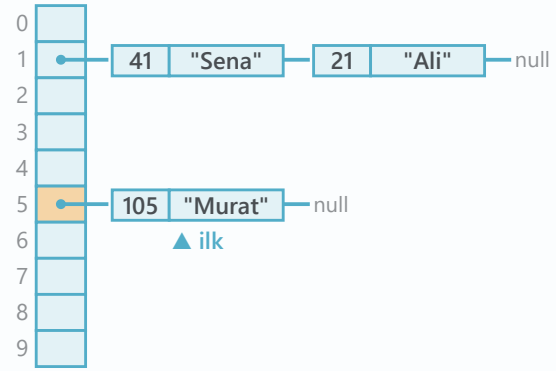
DEĞİŞKENLER

`anahtar = 105` `kovaIndisi = 5`

`ilk = düğüm(105)` `karşılaştırma = 0`

Kod İzleme: `getir(105)`, `getir(21)`, `getir(88)`

```
1  public String getir(Integer anahtar) {
2    int kovaIndisi = getKovaIndisi(anahtar);
3    HashDugumu ilk = kovalar[kovaIndisi];
▶    while (ilk != null) {
5      if (ilk.anahtar.equals(anahtar))
6        return ilk.deger;          // bulundu
7      ilk = ilk.sonraki;
8    }
9    return null;                   // anahtar yok
10 }
```



`ilk != null` → döngü gövdesine girilir.

Adım 3 / 17

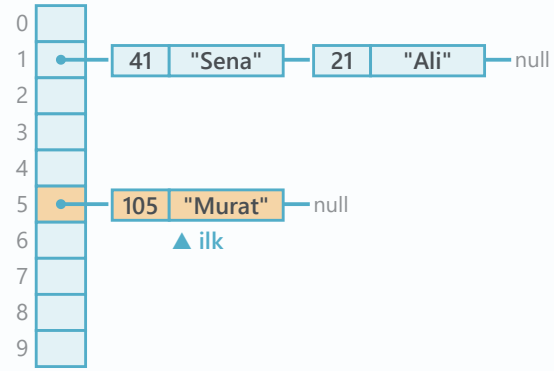
DEĞİŞKENLER

`anahtar = 105` `kovaIndisi = 5`

`ilk = düğüm(105)` `karşılaştırma = 0`

Kod İzleme: `getir(105)`, `getir(21)`, `getir(88)`

```
1  public String getir(Integer anahtar) {
2    int kovaIndisi = getKovaIndisi(anahtar);
3    HashDugumu ilk = kovalar[kovaIndisi];
4    while (ilk != null) {
▶      if (ilk.anahtar.equals(anahtar))
6        return ilk.deger;          // bulundu
7      ilk = ilk.sonraki;
8    }
9    return null;                  // anahtar yok
10 }
```



105.equals(105) → **eşit**.

Adım 4 / 17

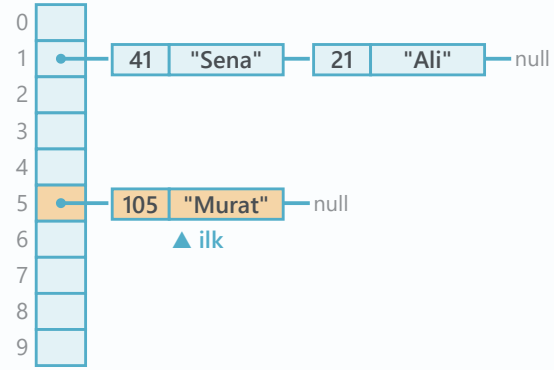
DEĞİŞKENLER

`anahtar = 105` `kovaIndisi = 5`

`ilk = düğüm(105)` `karşılaştırma = 1`

Kod İzleme: `getir(105)`, `getir(21)`, `getir(88)`

```
1  public String getir(Integer anahtar) {
2    int kovaIndisi = getKovaIndisi(anahtar);
3    HashDugumu ilk = kovalar[kovaIndisi];
4    while (ilk != null) {
5      if (ilk.anahtar.equals(anahtar))
▶        return ilk.deger;          // bulundu
7      ilk = ilk.sonraki;
8    }
9    return null;                   // anahtar yok
10 }
```



`return "Murat"` — 1 karşılaştırma yetti.

Adım 5 / 17

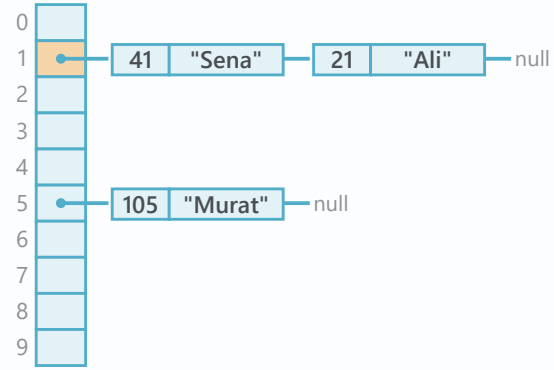
DEĞİŞKENLER

`anahtar = 105` `kovaIndisi = 5`

`ilk = düğüm(105)` `karşılaştırma = 1`

Kod İzleme: `getir(105)`, `getir(21)`, `getir(88)`

```
1  public String getir(Integer anahtar) {
2    int kovaIndisi = getKovaIndisi(anahtar);
3    HashDugumu ilk = kovalar[kovaIndisi];
4    while (ilk != null) {
5      if (ilk.anahtar.equals(anahtar))
6        return ilk.deger;          // bulundu
7      ilk = ilk.sonraki;
8    }
9    return null;                   // anahtar yok
10 }
```



`getir(21)` : h(21) = 21 mod 10 = **1**. Diğer kovalara hiç bakılmaz.

Adım 6 / 17

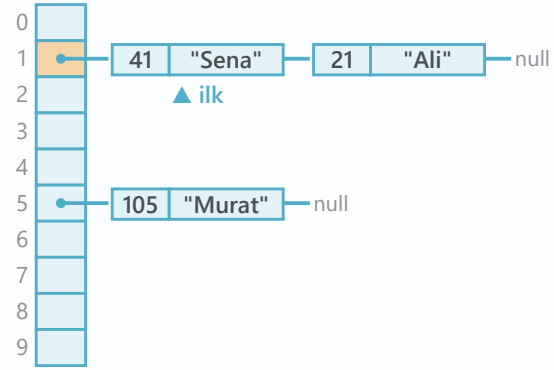
DEĞİŞKENLER

`anahtar = 21` `kovaIndisi = 1`

`ilk = –` `karşılaştırma = 0`

Kod İzleme: `getir(105)`, `getir(21)`, `getir(88)`

```
1  public String getir(Integer anahtar) {
2    int kovaIndisi = getKovaIndisi(anahtar);
▶    HashDugumu ilk = kovalar[kovaIndisi];
4    while (ilk != null) {
5      if (ilk.anahtar.equals(anahtar))
6        return ilk.deger;          // bulundu
7      ilk = ilk.sonraki;
8    }
9    return null;                   // anahtar yok
10 }
```



`ilk` kovanın ilk düğümünü gösterir.

Adım 7 / 17

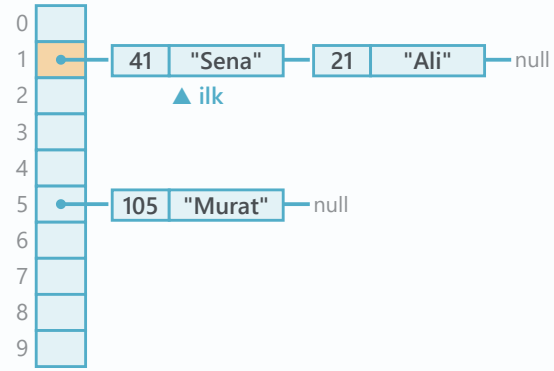
DEĞİŞKENLER

`anahtar = 21` `kovaIndisi = 1`

`ilk = düğüm(41)` `karşılaştırma = 0`

Kod İzleme: `getir(105)`, `getir(21)`, `getir(88)`

```
1  public String getir(Integer anahtar) {
2    int kovaIndisi = getKovaIndisi(anahtar);
3    HashDugumu ilk = kovalar[kovaIndisi];
▶    while (ilk != null) {
5      if (ilk.anahtar.equals(anahtar))
6        return ilk.deger;          // bulundu
7      ilk = ilk.sonraki;
8    }
9    return null;                   // anahtar yok
10 }
```



`ilk != null` → döngü gövdesine girilir.

Adım 8 / 17

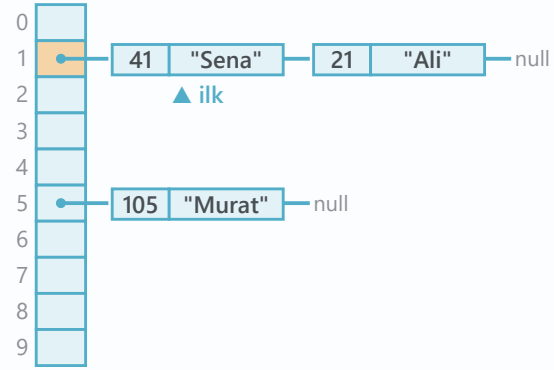
DEĞİŞKENLER

`anahtar = 21` `kovaIndisi = 1`

`ilk = düğüm(41)` `karşılaştırma = 0`

Kod İzleme: `getir(105)`, `getir(21)`, `getir(88)`

```
1  public String getir(Integer anahtar) {
2    int kovaIndisi = getKovaIndisi(anahtar);
3    HashDugumu ilk = kovalar[kovaIndisi];
4    while (ilk != null) {
▶      if (ilk.anahtar.equals(anahtar))
6        return ilk.deger;          // bulundu
7      ilk = ilk.sonraki;
8    }
9    return null;                  // anahtar yok
10 }
```



41 ≠ 21 → aynı kovaya düşmüş başka bir anahtar (çakışma).

Adım 9 / 17

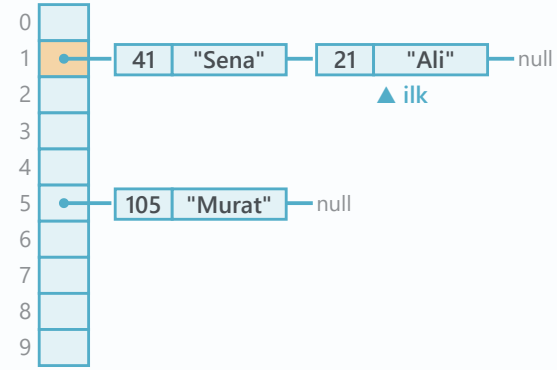
DEĞİŞKENLER

`anahtar = 21` `kovaIndisi = 1`

`ilk = düğüm(41)` `karşılaştırma = 1`

Kod İzleme: `getir(105)`, `getir(21)`, `getir(88)`

```
1  public String getir(Integer anahtar) {
2    int kovaIndisi = getKovaIndisi(anahtar);
3    HashDugumu ilk = kovalar[kovaIndisi];
4    while (ilk != null) {
5      if (ilk.anahtar.equals(anahtar))
6        return ilk.deger;          // bulundu
       ilk = ilk.sonraki;
8    }
9    return null;                   // anahtar yok
10 }
```



`ilk = ilk.sonraki` → düğüm 21.

Adım 10 / 17

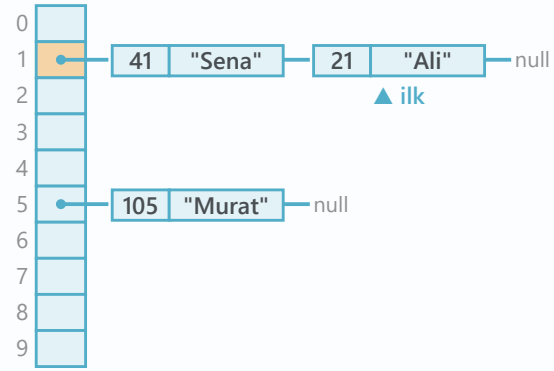
DEĞİŞKENLER

`anahtar = 21` `kovaIndisi = 1`

`ilk = düğüm(21)` `karşılaştırma = 1`

Kod İzleme: `getir(105)`, `getir(21)`, `getir(88)`

```
1  public String getir(Integer anahtar) {
2    int kovaIndisi = getKovaIndisi(anahtar);
3    HashDugumu ilk = kovalar[kovaIndisi];
▶  while (ilk != null) {
5      if (ilk.anahtar.equals(anahtar))
6        return ilk.deger;          // bulundu
7      ilk = ilk.sonraki;
8    }
9    return null;                   // anahtar yok
10 }
```



`ilk != null` → döngü gövdesine girilir.

Adım 11 / 17

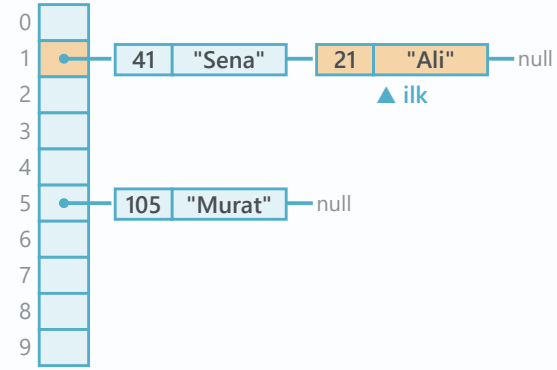
DEĞİŞKENLER

`anahtar = 21` `kovaIndisi = 1`

`ilk = düğüm(21)` `karşılaştırma = 1`

Kod İzleme: `getir(105)`, `getir(21)`, `getir(88)`

```
1  public String getir(Integer anahtar) {
2    int kovaIndisi = getKovaIndisi(anahtar);
3    HashDugumu ilk = kovalar[kovaIndisi];
4    while (ilk != null) {
▶      if (ilk.anahtar.equals(anahtar))
6        return ilk.deger;          // bulundu
7      ilk = ilk.sonraki;
8    }
9    return null;                  // anahtar yok
10 }
```



21.equals(21) → **eşit**.

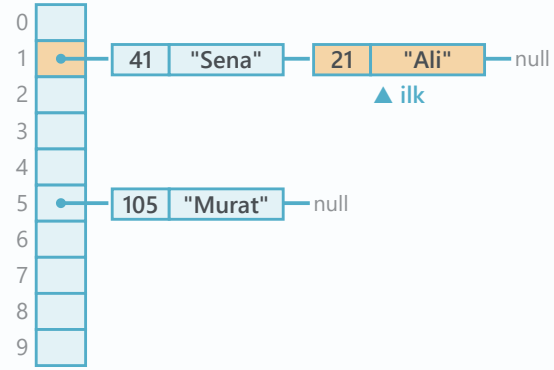
Adım 12 / 17

DEĞİŞKENLER

`anahtar` = **21** | `kovaIndisi` = **1** | `ilk` = **düğüm(21)** | `karşılaştırma` = **2**

Kod İzleme: `getir(105)`, `getir(21)`, `getir(88)`

```
1  public String getir(Integer anahtar) {
2    int kovaIndisi = getKovaIndisi(anahtar);
3    HashDugumu ilk = kovalar[kovaIndisi];
4    while (ilk != null) {
5      if (ilk.anahtar.equals(anahtar))
▶        return ilk.deger;          // bulundu
7      ilk = ilk.sonraki;
8    }
9    return null;                   // anahtar yok
10 }
```



`return "Ali"` — 2 karşılaştırma yetti.

Adım 13 / 17

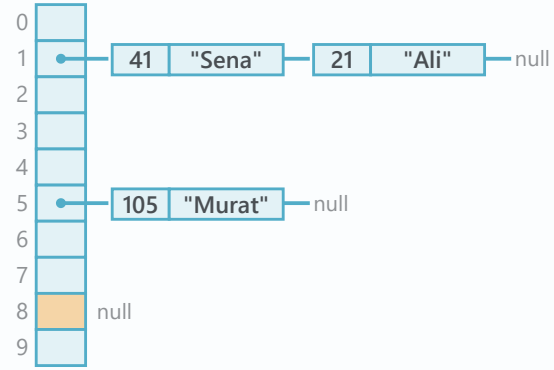
DEĞİŞKENLER

`anahtar = 21` `kovaIndisi = 1`

`ilk = düğüm(21)` `karşılaştırma = 2`

Kod İzleme: `getir(105)`, `getir(21)`, `getir(88)`

```
1  public String getir(Integer anahtar) {
2    int kovaIndisi = getKovaIndisi(anahtar);
3    HashDugumu ilk = kovalar[kovaIndisi];
4    while (ilk != null) {
5      if (ilk.anahtar.equals(anahtar))
6        return ilk.deger;          // bulundu
7      ilk = ilk.sonraki;
8    }
9    return null;                   // anahtar yok
10 }
```



`getir(88)` : h(88) = 88 mod 10 = **8**. Diğer kovalara hiç bakılmaz.

Adım 14 / 17

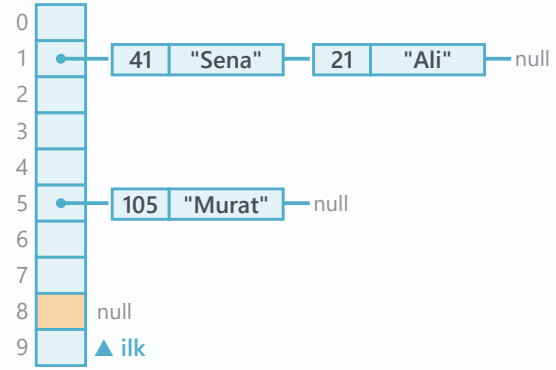
DEĞİŞKENLER

`anahtar = 88` `kovaIndisi = 8`

`ilk = –` `karşılaştırma = 0`

Kod İzleme: `getir(105)`, `getir(21)`, `getir(88)`

```
1  public String getir(Integer anahtar) {
2    int kovaIndisi = getKovaIndisi(anahtar);
▶    HashDugumu ilk = kovalar[kovaIndisi];
4    while (ilk != null) {
5      if (ilk.anahtar.equals(anahtar))
6        return ilk.deger;          // bulundu
7      ilk = ilk.sonraki;
8    }
9    return null;                   // anahtar yok
10 }
```



Kova 8 boş: `ilk = null`.

Adım 15 / 17

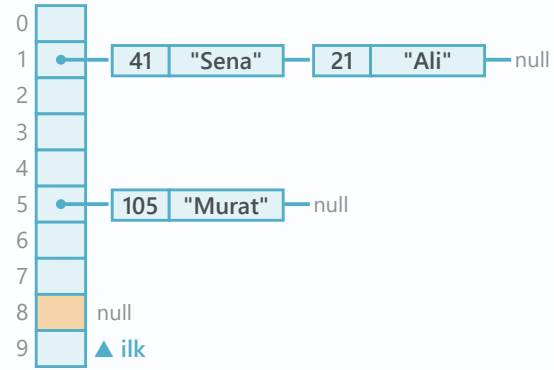
DEĞİŞKENLER

`anahtar = 88` `kovaIndisi = 8`

`ilk = null` `karşılaştırma = 0`

Kod İzleme: `getir(105)`, `getir(21)`, `getir(88)`

```
1  public String getir(Integer anahtar) {
2    int kovaIndisi = getKovaIndisi(anahtar);
3    HashDugumu ilk = kovalar[kovaIndisi];
▶    while (ilk != null) {
5      if (ilk.anahtar.equals(anahtar))
6        return ilk.deger;          // bulundu
7      ilk = ilk.sonraki;
8    }
9    return null;                   // anahtar yok
10 }
```



`ilk == null` → döngüye girilmez.

Adım 16 / 17

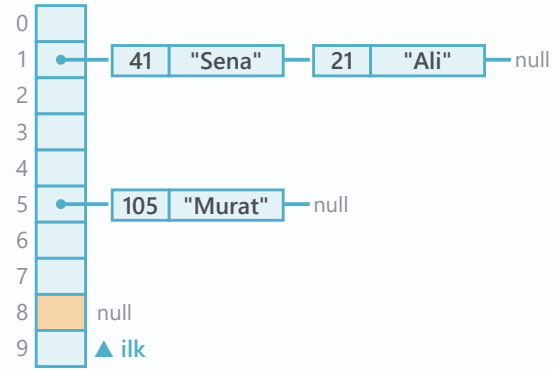
DEĞİŞKENLER

`anahtar = 88` `kovaIndisi = 8`

`ilk = null` `karşılaştırma = 0`

Kod İzleme: `getir(105)`, `getir(21)`, `getir(88)`

```
1  public String getir(Integer anahtar) {
2    int kovaIndisi = getKovaIndisi(anahtar);
3    HashDugumu ilk = kovalar[kovaIndisi];
4    while (ilk != null) {
5      if (ilk.anahtar.equals(anahtar))
6        return ilk.deger;          // bulundu
7      ilk = ilk.sonraki;
8    }
▶    return null;                  // anahtar yok
10 }
```



`return null` : 88 tabloda yok. Boş kova → tek adımda cevap.

Adım 17 / 17

DEĞİŞKENLER

`anahtar = 88` `kovaIndisi = 8`

`ilk = null` `karşılaştırma = 0`

Hash Tablosundan Anahtar Silme

```java
public String sil(Integer anahtar) {
  int kovaIndisi = getKovaIndisi(anahtar);
  HashDugumu ilk = kovalar[kovaIndisi];
  HashDugumu onceki = null;
  while (ilk != null &&
         !ilk.anahtar.equals(anahtar)) {
    onceki = ilk;
    ilk = ilk.sonraki;
  }
  if (ilk == null) return null;   // yok
  buyukluk--;
  if (onceki != null)       // ortadan/sondan
    onceki.sonraki = ilk.sonraki;
  else                      // baştan
    kovalar[kovaIndisi] = ilk.sonraki;
  return ilk.deger;
}
```

onceki

Tek yönlü listede düğümü koparmak için bir önceki düğüm gerekir.

İki durum

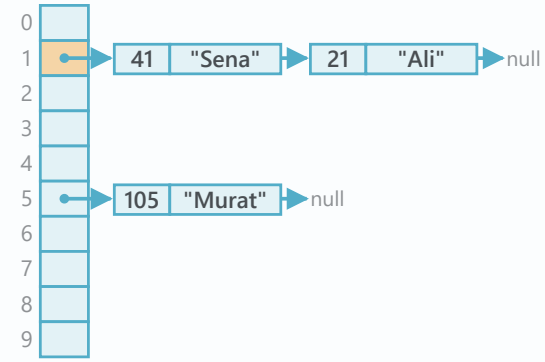
Silinen ilk düğümse kova güncellenir; değilse `onceki.sonraki`.

Maliyet

Getirme ile aynı: O(1 + zincir uzunluğu).

Kod İzleme: `sil(21)` (zincirin ortasından)

```
1  public String sil(Integer anahtar) {
2    int kovaIndisi = getKovaIndisi(anahtar);
3    HashDugumu ilk = kovalar[kovaIndisi];
4    HashDugumu onceki = null;
5    while (ilk != null &&
6           !ilk.anahtar.equals(anahtar)) {
7      onceki = ilk;
8      ilk = ilk.sonraki;
9    }
10   if (ilk == null) return null;   // yok
11   buyukluk--;
12   if (onceki != null)       // ortadan/sondan
13     onceki.sonraki = ilk.sonraki;
14   else                      // baştan
15     kovalar[kovaIndisi] = ilk.sonraki;
16   return ilk.deger;
17 }
```



`sil(21)` : h(21) = 21 mod 10 = **1**.

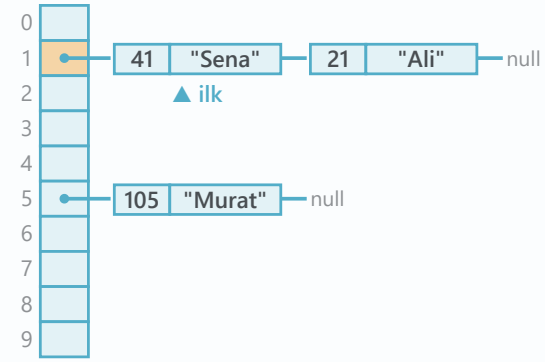
Adım 1 / 12

DEĞİŞKENLER

`anahtar` = **21** | `kovaIndisi` = **1** | `ilk` = **–** | `onceki` = **–** | `buyukluk` = **3**

Kod İzleme: `sil(21)` (zincirin ortasından)

```
1  public String sil(Integer anahtar) {
2    int kovaIndisi = getKovaIndisi(anahtar);
▶    HashDugumu ilk = kovalar[kovaIndisi];
4    HashDugumu onceki = null;
5    while (ilk != null &&
6           !ilk.anahtar.equals(anahtar)) {
7      onceki = ilk;
8      ilk = ilk.sonraki;
9    }
10   if (ilk == null) return null;   // yok
11   buyukluk--;
12   if (onceki != null)       // ortadan/sondan
13     onceki.sonraki = ilk.sonraki;
14   else                      // baştan
15     kovalar[kovaIndisi] = ilk.sonraki;
16   return ilk.deger;
17 }
```



`ilk` baş düğümü gösterir.

Adım 2 / 12

DEĞİŞKENLER

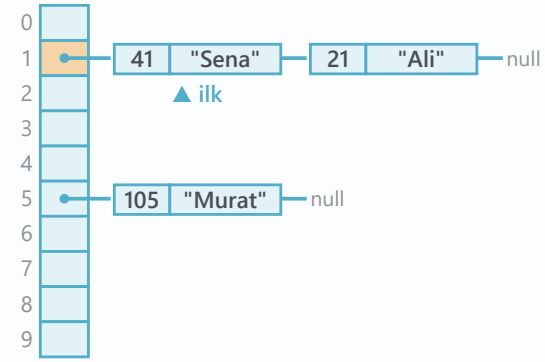
`anahtar = 21` `kovaIndisi = 1`

`ilk = düğüm(41)` `onceki = –`

`buyukluk = 3`

Kod İzleme: `sil(21)` (zincirin ortasından)

```
1  public String sil(Integer anahtar) {
2    int kovaIndisi = getKovaIndisi(anahtar);
3    HashDugumu ilk = kovalar[kovaIndisi];
▶    HashDugumu onceki = null;
5    while (ilk != null &&
6           !ilk.anahtar.equals(anahtar)) {
7      onceki = ilk;
8      ilk = ilk.sonraki;
9    }
10   if (ilk == null) return null;   // yok
11   buyukluk--;
12   if (onceki != null)       // ortadan/sondan
13     onceki.sonraki = ilk.sonraki;
14   else                      // baştan
15     kovalar[kovaIndisi] = ilk.sonraki;
16   return ilk.deger;
17 }
```



`onceki = null` : tek yönlü listede düğümü koparmak için bir önceki düğüm gerekir.

Adım 3 / 12

DEĞİŞKENLER

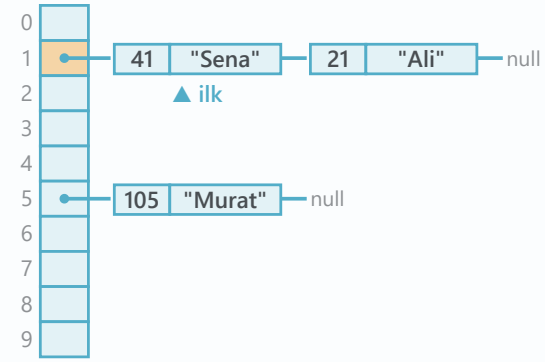
`anahtar = 21` `kovaIndisi = 1`

`ilk = düğüm(41)` `onceki = null`

`buyukluk = 3`

Kod İzleme: `sil(21)` (zincirin ortasından)

```
1  public String sil(Integer anahtar) {
2    int kovaIndisi = getKovaIndisi(anahtar);
3    HashDugumu ilk = kovalar[kovaIndisi];
4    HashDugumu onceki = null;
▶   while (ilk != null &&
▶          !ilk.anahtar.equals(anahtar)) {
7      onceki = ilk;
8      ilk = ilk.sonraki;
9    }
10   if (ilk == null) return null;   // yok
11   buyukluk--;
12   if (onceki != null)       // ortadan/sondan
13     onceki.sonraki = ilk.sonraki;
14   else                      // baştan
15     kovalar[kovaIndisi] = ilk.sonraki;
16   return ilk.deger;
17 }
```



41 ≠ 21 → koşul doğru, döngü gövdesine girilir.

Adım 4 / 12

DEĞİŞKENLER

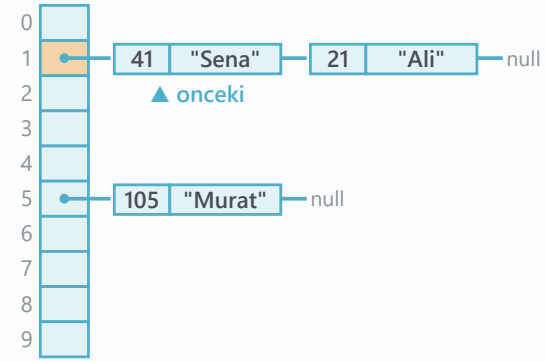
`anahtar = 21` `kovaIndisi = 1`

`ilk = düğüm(41)` `onceki = null`

`buyukluk = 3`

Kod İzleme: `sil(21)` (zincirin ortasından)

```
1  public String sil(Integer anahtar) {
2    int kovaIndisi = getKovaIndisi(anahtar);
3    HashDugumu ilk = kovalar[kovaIndisi];
4    HashDugumu onceki = null;
5    while (ilk != null &&
6           !ilk.anahtar.equals(anahtar)) {
▶      onceki = ilk;
8      ilk = ilk.sonraki;
9    }
10   if (ilk == null) return null;   // yok
11   buyukluk--;
12   if (onceki != null)       // ortadan/sondan
13     onceki.sonraki = ilk.sonraki;
14   else                      // baştan
15     kovalar[kovaIndisi] = ilk.sonraki;
16   return ilk.deger;
17 }
```



`onceki = ilk` → düğüm 41.

Adım 5 / 12

DEĞİŞKENLER

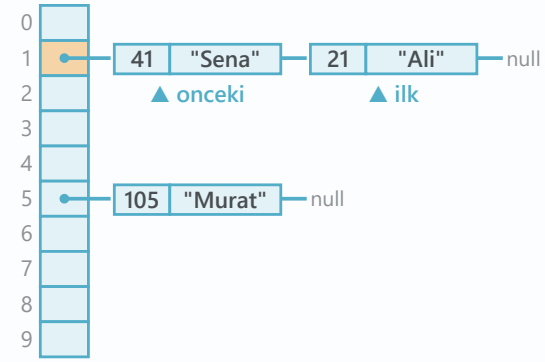
`anahtar = 21` `kovaIndisi = 1`

`ilk = düğüm(41)` `onceki = düğüm(41)`

`buyukluk = 3`

Kod İzleme: `sil(21)` (zincirin ortasından)

```
1  public String sil(Integer anahtar) {
2    int kovaIndisi = getKovaIndisi(anahtar);
3    HashDugumu ilk = kovalar[kovaIndisi];
4    HashDugumu onceki = null;
5    while (ilk != null &&
6           !ilk.anahtar.equals(anahtar)) {
7      onceki = ilk;
▶      ilk = ilk.sonraki;
9    }
10   if (ilk == null) return null;   // yok
11   buyukluk--;
12   if (onceki != null)        // ortadan/sondan
13     onceki.sonraki = ilk.sonraki;
14   else                       // baştan
15     kovalar[kovaIndisi] = ilk.sonraki;
16   return ilk.deger;
17 }
```



`ilk = ilk.sonraki` → düğüm 21.

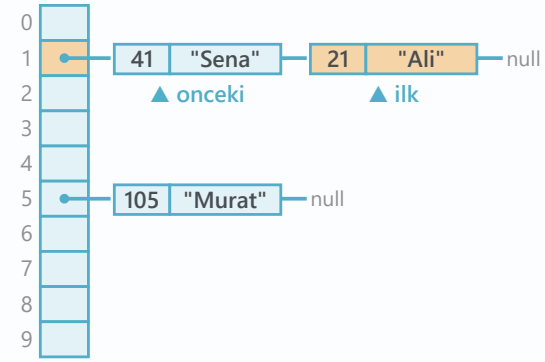
Adım 6 / 12

DEĞİŞKENLER

`anahtar = 21` `kovaIndisi = 1` `ilk = düğüm(21)` `onceki = düğüm(41)` `buyukluk = 3`

Kod İzleme: `sil(21)` (zincirin ortasından)

```
1  public String sil(Integer anahtar) {
2    int kovaIndisi = getKovaIndisi(anahtar);
3    HashDugumu ilk = kovalar[kovaIndisi];
4    HashDugumu onceki = null;
▶   while (ilk != null &&
▶          !ilk.anahtar.equals(anahtar)) {
7      onceki = ilk;
8      ilk = ilk.sonraki;
9    }
10   if (ilk == null) return null;   // yok
11   buyukluk--;
12   if (onceki != null)       // ortadan/sondan
13     onceki.sonraki = ilk.sonraki;
14   else                      // baştan
15     kovalar[kovaIndisi] = ilk.sonraki;
16   return ilk.deger;
17 }
```



21.equals(21) → eşit: koşul yanlış, döngü biter. Silinecek düğüm bulundu.

Adım 7 / 12

DEĞİŞKENLER

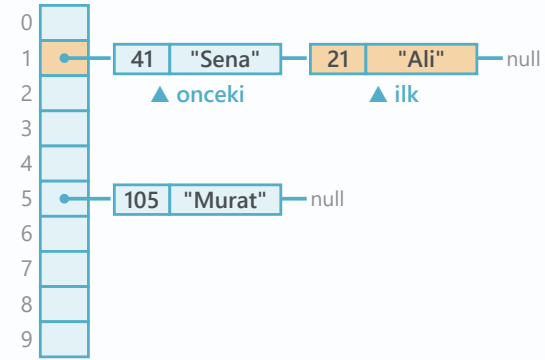
`anahtar = 21` `kovaIndisi = 1`

`ilk = düğüm(21)` `onceki = düğüm(41)`

`buyukluk = 3`

Kod İzleme: `sil(21)` (zincirin ortasından)

```
1  public String sil(Integer anahtar) {
2    int kovaIndisi = getKovaIndisi(anahtar);
3    HashDugumu ilk = kovalar[kovaIndisi];
4    HashDugumu onceki = null;
5    while (ilk != null &&
6           !ilk.anahtar.equals(anahtar)) {
7      onceki = ilk;
8      ilk = ilk.sonraki;
9    }
▶    if (ilk == null) return null;   // yok
11   buyukluk--;
12   if (onceki != null)       // ortadan/sondan
13     onceki.sonraki = ilk.sonraki;
14   else                      // baştan
15     kovalar[kovaIndisi] = ilk.sonraki;
16   return ilk.deger;
17 }
```



`ilk != null` → anahtar var, silme yapılacak.

Adım 8 / 12

DEĞİŞKENLER

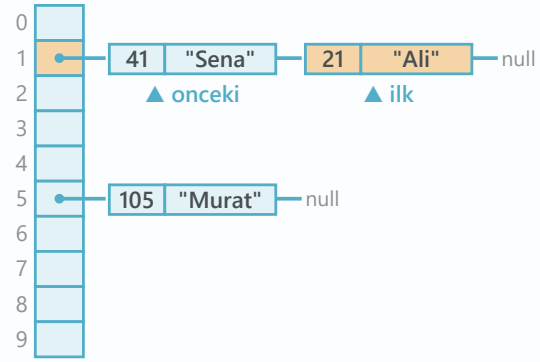
`anahtar = 21` `kovaIndisi = 1`

`ilk = düğüm(21)` `onceki = düğüm(41)`

`buyukluk = 3`

Kod İzleme: `sil(21)` (zincirin ortasından)

```
1  public String sil(Integer anahtar) {
2    int kovaIndisi = getKovaIndisi(anahtar);
3    HashDugumu ilk = kovalar[kovaIndisi];
4    HashDugumu onceki = null;
5    while (ilk != null &&
6           !ilk.anahtar.equals(anahtar)) {
7      onceki = ilk;
8      ilk = ilk.sonraki;
9    }
10   if (ilk == null) return null;   // yok
▶    buyukluk--;
12   if (onceki != null)       // ortadan/sondan
13     onceki.sonraki = ilk.sonraki;
14   else                      // baştan
15     kovalar[kovaIndisi] = ilk.sonraki;
16   return ilk.deger;
17 }
```



buyukluk = 2.

Adım 9 / 12

DEĞİŞKENLER

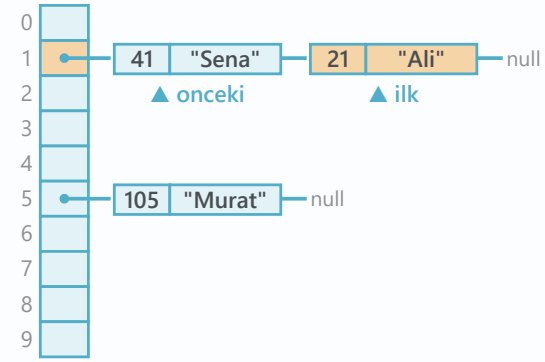
`anahtar = 21` `kovaIndisi = 1`

`ilk = düğüm(21)` `onceki = düğüm(41)`

`buyukluk = 2`

Kod İzleme: `sil(21)` (zincirin ortasından)

```
1  public String sil(Integer anahtar) {
2    int kovaIndisi = getKovaIndisi(anahtar);
3    HashDugumu ilk = kovalar[kovaIndisi];
4    HashDugumu onceki = null;
5    while (ilk != null &&
6           !ilk.anahtar.equals(anahtar)) {
7      onceki = ilk;
8      ilk = ilk.sonraki;
9    }
10   if (ilk == null) return null;   // yok
11   buyukluk--;
▶    if (onceki != null)       // ortadan/sondan
13     onceki.sonraki = ilk.sonraki;
14   else                      // baştan
15     kovalar[kovaIndisi] = ilk.sonraki;
16   return ilk.deger;
17 }
```



`onceki != null` → düğüm ortada/sonda: bağı `onceki` üzerinden atlanır.

Adım 10 / 12

DEĞİŞKENLER

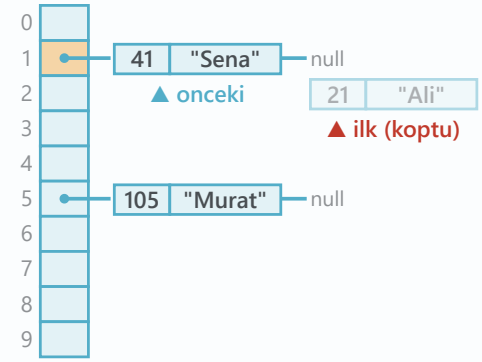
`anahtar = 21` `kovaIndisi = 1`

`ilk = düğüm(21)` `onceki = düğüm(41)`

`buyukluk = 2`

Kod İzleme: `sil(21)` (zincirin ortasından)

```
1  public String sil(Integer anahtar) {
2    int kovaIndisi = getKovaIndisi(anahtar);
3    HashDugumu ilk = kovalar[kovaIndisi];
4    HashDugumu onceki = null;
5    while (ilk != null &&
6           !ilk.anahtar.equals(anahtar)) {
7      onceki = ilk;
8      ilk = ilk.sonraki;
9    }
10   if (ilk == null) return null;   // yok
11   buyukluk--;
12   if (onceki != null)       // ortadan/sondan
▶      onceki.sonraki = ilk.sonraki;
14   else                      // baştan
15     kovalar[kovaIndisi] = ilk.sonraki;
16   return ilk.deger;
17 }
```



`onceki.sonraki = ilk.sonraki` (null): 41 artık 21'i atlar; 21 zincirden koptu.

Adım 11 / 12

DEĞİŞKENLER

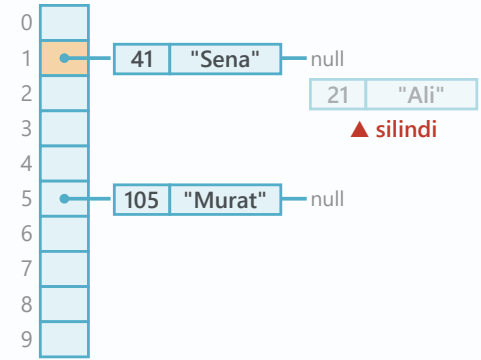
`anahtar = 21` `kovaIndisi = 1`

`ilk = düğüm(21)` `onceki = düğüm(41)`

`buyukluk = 2`

Kod İzleme: `sil(21)` (zincirin ortasından)

```
1  public String sil(Integer anahtar) {
2    int kovaIndisi = getKovaIndisi(anahtar);
3    HashDugumu ilk = kovalar[kovaIndisi];
4    HashDugumu onceki = null;
5    while (ilk != null &&
6           !ilk.anahtar.equals(anahtar)) {
7      onceki = ilk;
8      ilk = ilk.sonraki;
9    }
10   if (ilk == null) return null;   // yok
11   buyukluk--;
12   if (onceki != null)       // ortadan/sondan
13     onceki.sonraki = ilk.sonraki;
14   else                      // baştan
15     kovalar[kovaIndisi] = ilk.sonraki;
▶    return ilk.deger;
17 }
```



`return "Ali"` . Kopan düğümü çöp toplayıcı temizler.

Adım 12 / 12

DEĞİŞKENLER

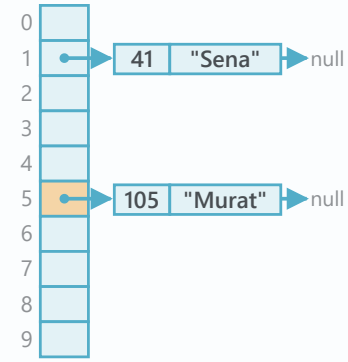
`anahtar = 21` `kovaIndisi = 1`

`ilk = düğüm(21)` `onceki = düğüm(41)`

`buyukluk = 2`

Kod İzleme: `sil(105)` (baştan), `sil(88)` (yok)

```
1  public String sil(Integer anahtar) {
2    int kovaIndisi = getKovaIndisi(anahtar);
3    HashDugumu ilk = kovalar[kovaIndisi];
4    HashDugumu onceki = null;
5    while (ilk != null &&
6           !ilk.anahtar.equals(anahtar)) {
7      onceki = ilk;
8      ilk = ilk.sonraki;
9    }
10   if (ilk == null) return null;   // yok
11   buyukluk--;
12   if (onceki != null)       // ortadan/sondan
13     onceki.sonraki = ilk.sonraki;
14   else                      // baştan
15     kovalar[kovaIndisi] = ilk.sonraki;
16   return ilk.deger;
17 }
```



`sil(105)` : h(105) = 105 mod 10 = **5**.

Adım 1 / 14

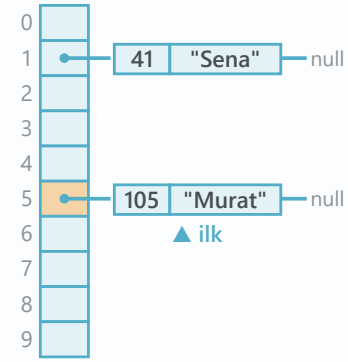
DEĞİŞKENLER

`anahtar = 105` `kovaIndisi = 5`

`ilk = –` `onceki = –` `buyukluk = 2`

Kod İzleme: `sil(105)` (baştan), `sil(88)` (yok)

```
1  public String sil(Integer anahtar) {
2    int kovaIndisi = getKovaIndisi(anahtar);
▶    HashDugumu ilk = kovalar[kovaIndisi];
4    HashDugumu onceki = null;
5    while (ilk != null &&
6           !ilk.anahtar.equals(anahtar)) {
7      onceki = ilk;
8      ilk = ilk.sonraki;
9    }
10   if (ilk == null) return null;   // yok
11   buyukluk--;
12   if (onceki != null)       // ortadan/sondan
13     onceki.sonraki = ilk.sonraki;
14   else                      // baştan
15     kovalar[kovaIndisi] = ilk.sonraki;
16   return ilk.deger;
17 }
```



`ilk` baş düğümü gösterir.

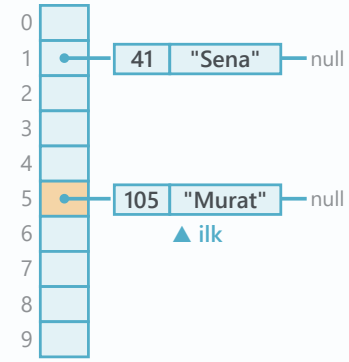
Adım 2 / 14

DEĞİŞKENLER

`anahtar = 105` `kovaIndisi = 5` `ilk = düğüm(105)` `onceki = –` `buyukluk = 2`

Kod İzleme: `sil(105)` (baştan), `sil(88)` (yok)

```
1  public String sil(Integer anahtar) {
2    int kovaIndisi = getKovaIndisi(anahtar);
3    HashDugumu ilk = kovalar[kovaIndisi];
▶    HashDugumu onceki = null;
5    while (ilk != null &&
6           !ilk.anahtar.equals(anahtar)) {
7      onceki = ilk;
8      ilk = ilk.sonraki;
9    }
10   if (ilk == null) return null;   // yok
11   buyukluk--;
12   if (onceki != null)       // ortadan/sondan
13     onceki.sonraki = ilk.sonraki;
14   else                      // baştan
15     kovalar[kovaIndisi] = ilk.sonraki;
16   return ilk.deger;
17 }
```



`onceki = null` : tek yönlü listede düğümü koparmak için bir önceki düğüm gerekir.

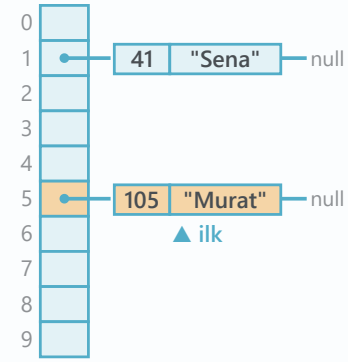
Adım 3 / 14

DEĞİŞKENLER

`anahtar = 105` `kovaIndisi = 5` `ilk = düğüm(105)` `onceki = null` `buyukluk = 2`

Kod İzleme: `sil(105)` (baştan), `sil(88)` (yok)

```
1  public String sil(Integer anahtar) {
2    int kovaIndisi = getKovaIndisi(anahtar);
3    HashDugumu ilk = kovalar[kovaIndisi];
4    HashDugumu onceki = null;
▶    while (ilk != null &&
▶           !ilk.anahtar.equals(anahtar)) {
7      onceki = ilk;
8      ilk = ilk.sonraki;
9    }
10   if (ilk == null) return null;   // yok
11   buyukluk--;
12   if (onceki != null)       // ortadan/sondan
13     onceki.sonraki = ilk.sonraki;
14   else                      // baştan
15     kovalar[kovaIndisi] = ilk.sonraki;
16   return ilk.deger;
17 }
```



105.equals(105) → eşit: koşul yanlış, döngü biter. Silinecek düğüm bulundu.

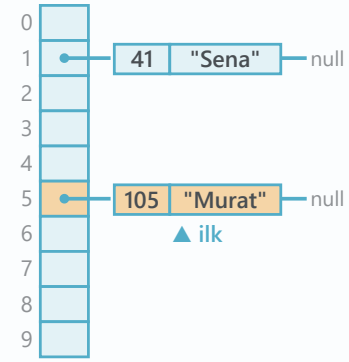
Adım 4 / 14

DEĞİŞKENLER

`anahtar = 105` `kovaIndisi = 5` `ilk = düğüm(105)` `onceki = null` `buyukluk = 2`

Kod İzleme: `sil(105)` (baştan), `sil(88)` (yok)

```
1  public String sil(Integer anahtar) {
2    int kovaIndisi = getKovaIndisi(anahtar);
3    HashDugumu ilk = kovalar[kovaIndisi];
4    HashDugumu onceki = null;
5    while (ilk != null &&
6           !ilk.anahtar.equals(anahtar)) {
7      onceki = ilk;
8      ilk = ilk.sonraki;
9    }
▶    if (ilk == null) return null;   // yok
11   buyukluk--;
12   if (onceki != null)       // ortadan/sondan
13     onceki.sonraki = ilk.sonraki;
14   else                      // baştan
15     kovalar[kovaIndisi] = ilk.sonraki;
16   return ilk.deger;
17 }
```



`ilk != null` → anahtar var, silme yapılacak.

Adım 5 / 14

DEĞİŞKENLER

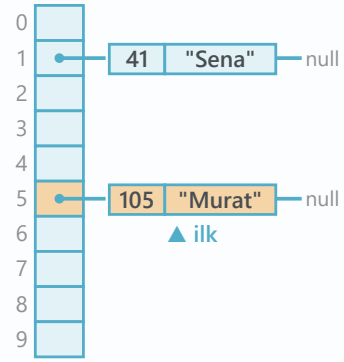
`anahtar = 105` `kovaIndisi = 5`

`ilk = düğüm(105)` `onceki = null`

`buyukluk = 2`

Kod İzleme: `sil(105)` (baştan), `sil(88)` (yok)

```
1  public String sil(Integer anahtar) {
2    int kovaIndisi = getKovaIndisi(anahtar);
3    HashDugumu ilk = kovalar[kovaIndisi];
4    HashDugumu onceki = null;
5    while (ilk != null &&
6           !ilk.anahtar.equals(anahtar)) {
7      onceki = ilk;
8      ilk = ilk.sonraki;
9    }
10   if (ilk == null) return null;   // yok
▶    buyukluk--;
12   if (onceki != null)       // ortadan/sondan
13     onceki.sonraki = ilk.sonraki;
14   else                      // baştan
15     kovalar[kovaIndisi] = ilk.sonraki;
16   return ilk.deger;
17 }
```



buyukluk = 1.

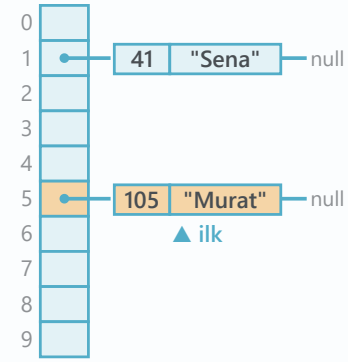
Adım 6 / 14

DEĞİŞKENLER

`anahtar = 105` `kovaIndisi = 5` `ilk = düğüm(105)` `onceki = null` `buyukluk = 1`

Kod İzleme: `sil(105)` (baştan), `sil(88)` (yok)

```
1  public String sil(Integer anahtar) {
2    int kovaIndisi = getKovaIndisi(anahtar);
3    HashDugumu ilk = kovalar[kovaIndisi];
4    HashDugumu onceki = null;
5    while (ilk != null &&
6           !ilk.anahtar.equals(anahtar)) {
7      onceki = ilk;
8      ilk = ilk.sonraki;
9    }
10   if (ilk == null) return null;   // yok
11   buyukluk--;
▶    if (onceki != null)       // ortadan/sondan
13     onceki.sonraki = ilk.sonraki;
14   else                      // baştan
15     kovalar[kovaIndisi] = ilk.sonraki;
16   return ilk.deger;
17 }
```



`onceki == null` → silinen düğüm zincirin başı: `else` dalı.

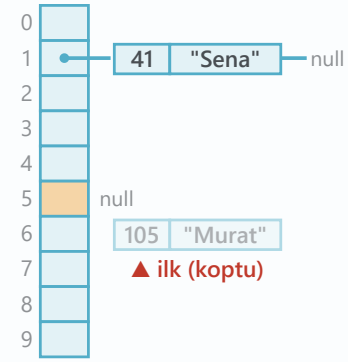
Adım 7 / 14

DEĞİŞKENLER

`anahtar = 105` `kovaIndisi = 5` `ilk = düğüm(105)` `onceki = null` `buyukluk = 1`

Kod İzleme: `sil(105)` (baştan), `sil(88)` (yok)

```
1  public String sil(Integer anahtar) {
2    int kovaIndisi = getKovaIndisi(anahtar);
3    HashDugumu ilk = kovalar[kovaIndisi];
4    HashDugumu onceki = null;
5    while (ilk != null &&
6           !ilk.anahtar.equals(anahtar)) {
7      onceki = ilk;
8      ilk = ilk.sonraki;
9    }
10   if (ilk == null) return null;   // yok
11   buyukluk--;
12   if (onceki != null)       // ortadan/sondan
13     onceki.sonraki = ilk.sonraki;
14   else                      // baştan
▶      kovalar[kovaIndisi] = ilk.sonraki;
16   return ilk.deger;
17 }
```



`kovalar[5] = ilk.sonraki` (null): kova artık 105'i göstermez.

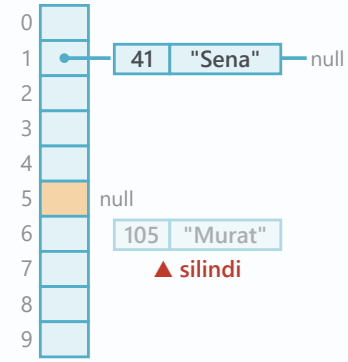
Adım 8 / 14

DEĞİŞKENLER

`anahtar = 105` `kovaIndisi = 5` `ilk = düğüm(105)` `onceki = null` `buyukluk = 1`

Kod İzleme: `sil(105)` (baştan), `sil(88)` (yok)

```
1  public String sil(Integer anahtar) {
2    int kovaIndisi = getKovaIndisi(anahtar);
3    HashDugumu ilk = kovalar[kovaIndisi];
4    HashDugumu onceki = null;
5    while (ilk != null &&
6           !ilk.anahtar.equals(anahtar)) {
7      onceki = ilk;
8      ilk = ilk.sonraki;
9    }
10   if (ilk == null) return null;   // yok
11   buyukluk--;
12   if (onceki != null)       // ortadan/sondan
13     onceki.sonraki = ilk.sonraki;
14   else                      // baştan
15     kovalar[kovaIndisi] = ilk.sonraki;
▶    return ilk.deger;
17 }
```



`return "Murat"`. Kopan düğümü çöp toplayıcı temizler.

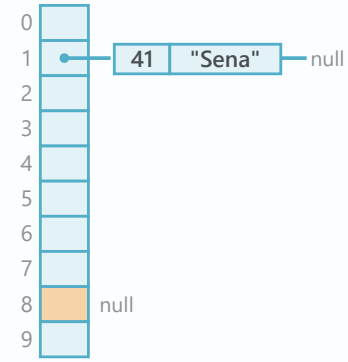
Adım 9 / 14

DEĞİŞKENLER

`anahtar = 105` `kovaIndisi = 5` `ilk = düğüm(105)` `onceki = null` `buyukluk = 1`

Kod İzleme: `sil(105)` (baştan), `sil(88)` (yok)

```
1  public String sil(Integer anahtar) {
2    int kovaIndisi = getKovaIndisi(anahtar);
3    HashDugumu ilk = kovalar[kovaIndisi];
4    HashDugumu onceki = null;
5    while (ilk != null &&
6           !ilk.anahtar.equals(anahtar)) {
7      onceki = ilk;
8      ilk = ilk.sonraki;
9    }
10   if (ilk == null) return null;   // yok
11   buyukluk--;
12   if (onceki != null)       // ortadan/sondan
13     onceki.sonraki = ilk.sonraki;
14   else                      // baştan
15     kovalar[kovaIndisi] = ilk.sonraki;
16   return ilk.deger;
17 }
```



`sil(88)` : h(88) = 88 mod 10 = **8**.

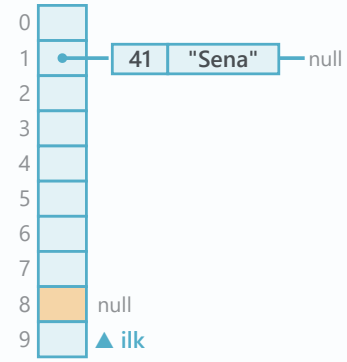
Adım 10 / 14

DEĞİŞKENLER

`anahtar = 88` `kovaIndisi = 8` `ilk = –` `onceki = –` `buyukluk = 1`

Kod İzleme: `sil(105)` (baştan), `sil(88)` (yok)

```
1  public String sil(Integer anahtar) {
2    int kovaIndisi = getKovaIndisi(anahtar);
▶    HashDugumu ilk = kovalar[kovaIndisi];
4    HashDugumu onceki = null;
5    while (ilk != null &&
6           !ilk.anahtar.equals(anahtar)) {
7      onceki = ilk;
8      ilk = ilk.sonraki;
9    }
10   if (ilk == null) return null;   // yok
11   buyukluk--;
12   if (onceki != null)       // ortadan/sondan
13     onceki.sonraki = ilk.sonraki;
14   else                      // baştan
15     kovalar[kovaIndisi] = ilk.sonraki;
16   return ilk.deger;
17 }
```



Kova 8 boş: `ilk = null`.

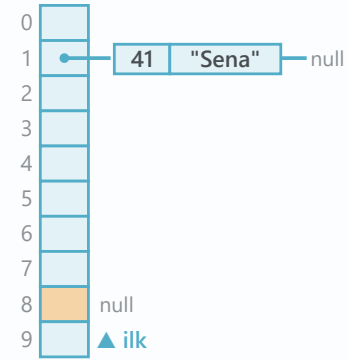
Adım 11 / 14

DEĞİŞKENLER

`anahtar = 88` `kovaIndisi = 8` `ilk = null` `onceki = –` `buyukluk = 1`

Kod İzleme: `sil(105)` (baştan), `sil(88)` (yok)

```
1  public String sil(Integer anahtar) {
2    int kovaIndisi = getKovaIndisi(anahtar);
3    HashDugumu ilk = kovalar[kovaIndisi];
▶    HashDugumu onceki = null;
5    while (ilk != null &&
6           !ilk.anahtar.equals(anahtar)) {
7      onceki = ilk;
8      ilk = ilk.sonraki;
9    }
10   if (ilk == null) return null;   // yok
11   buyukluk--;
12   if (onceki != null)       // ortadan/sondan
13     onceki.sonraki = ilk.sonraki;
14   else                      // baştan
15     kovalar[kovaIndisi] = ilk.sonraki;
16   return ilk.deger;
17 }
```



`onceki = null` : tek yönlü listede düğümü koparmak için bir önceki düğüm gerekir.

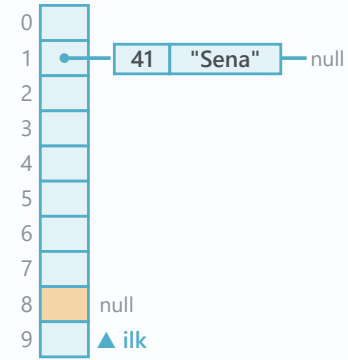
Adım 12 / 14

DEĞİŞKENLER

`anahtar = 88` `kovaIndisi = 8` `ilk = null` `onceki = null` `buyukluk = 1`

Kod İzleme: `sil(105)` (baştan), `sil(88)` (yok)

```
1  public String sil(Integer anahtar) {
2    int kovaIndisi = getKovaIndisi(anahtar);
3    HashDugumu ilk = kovalar[kovaIndisi];
4    HashDugumu onceki = null;
▶    while (ilk != null &&
▶           !ilk.anahtar.equals(anahtar)) {
7      onceki = ilk;
8      ilk = ilk.sonraki;
9    }
10   if (ilk == null) return null;   // yok
11   buyukluk--;
12   if (onceki != null)       // ortadan/sondan
13     onceki.sonraki = ilk.sonraki;
14   else                      // baştan
15     kovalar[kovaIndisi] = ilk.sonraki;
16   return ilk.deger;
17 }
```



`ilk == null` → döngü hemen biter.

Adım 13 / 14

DEĞİŞKENLER

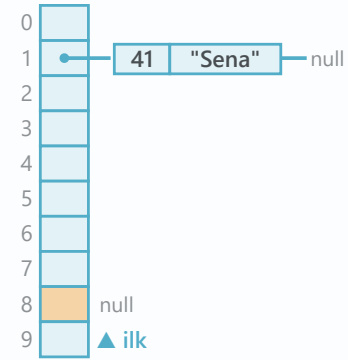
`anahtar = 88` `kovaIndisi = 8`

`ilk = null` `onceki = null`

`buyukluk = 1`

Kod İzleme: `sil(105)` (baştan), `sil(88)` (yok)

```
1  public String sil(Integer anahtar) {
2    int kovaIndisi = getKovaIndisi(anahtar);
3    HashDugumu ilk = kovalar[kovaIndisi];
4    HashDugumu onceki = null;
5    while (ilk != null &&
6           !ilk.anahtar.equals(anahtar)) {
7      onceki = ilk;
8      ilk = ilk.sonraki;
9    }
▶    if (ilk == null) return null;   // yok
11   buyukluk--;
12   if (onceki != null)       // ortadan/sondan
13     onceki.sonraki = ilk.sonraki;
14   else                      // baştan
15     kovalar[kovaIndisi] = ilk.sonraki;
16   return ilk.deger;
17 }
```



`return null` : 88 yok; **buyukluk** değişmez.

Adım 14 / 14

DEĞİŞKENLER

`anahtar = 88` `kovaIndisi = 8` `ilk = null` `onceki = null` `buyukluk = 1`

Açık Adresleme (Open Addressing)

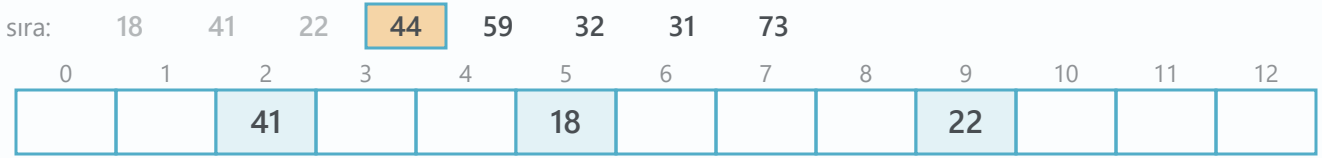
- Tüm çiftler dizinin kendisinde saklanır; yuva başına en fazla bir çift ($n \le N$).
- Çakışmada bir **yoklama dizisi** (probe sequence) ile boş yuva aranır: `h(k, i) = (h(k) + f(i)) mod N`, $i = 0, 1, 2, \ldots$

Yöntem	f(i)	Yoklama dizisi (h(k) = 5, N = 13)	Zayıf yönü
Doğrusal (linear probing)	i	5, 6, 7, 8, 9, …	Birincil kümelenme
Karesel (quadratic probing)	i^2	5, 6, 9, 14→1, 21→8, …	İkincil kümelenme; her yuvayı denemeyebilir
Çift hash (double hashing)	$i \cdot h_2(k)$	5, 5+d, 5+2d, … ($d = h_2(k)$)	$h_2(k) \neq 0$ ve adım ile N aralarında asal olmalı

Zincirlemeye göre: ek düğüm ve referans yok, bellek ardışık (önbellek dostu). **Ama:** yük faktörü düşük tutulmalı ve silme özel işlem ister. N asal ise 1 … N−1 arasındaki her adım N ile aralarında asaldır.

AÇIK ADRESLEME · KODU ADIM ADIM ÇALIŞTIR

Doğrusal Yoklama (Linear Probing)



18 → 5, 41 → 2, 22 → 9: üçü de ilk denemede boş yuvaya yerleşti. Sırada 44 var.

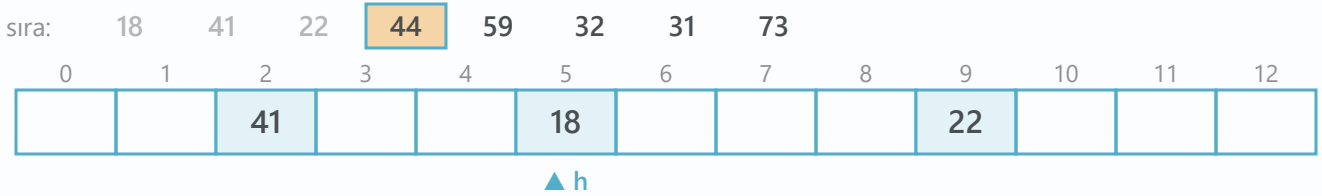
Adım 1 / 13

```
// yerlestir(k): doğrusal yoklama, N = 13
int h = k % 13;                 // ev yuvası
for (int i = 0; i < 13; i++) {
  int j = (h + i) % 13;           // h, h+1, h+2…
  if (tablo[j] == null) {       // boş mu?
    tablo[j] = k;
    break;
  }                             // dolu: i++
}
```

DEĞİŞKENLER

`N = 13`

Doğrusal Yoklama (Linear Probing)



Sıradaki anahtar 44: **h = 44 mod 13 = 5** (ev yuvası).

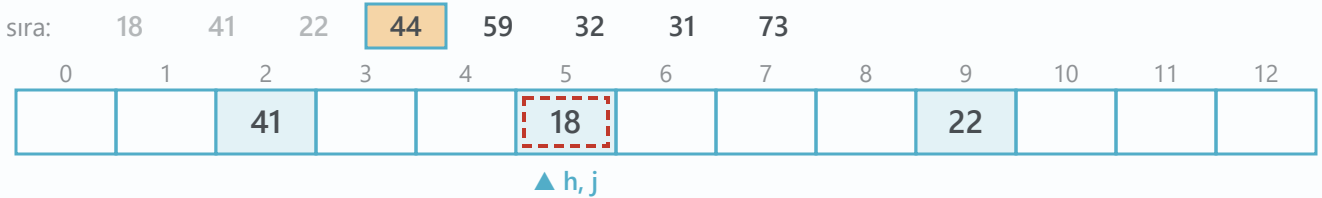
Adım 2 / 13

```
1 // yerlestir(k): doğrusal yoklama, N = 13
▶ int h = k % 13;                  // ev yuvası
3 for (int i = 0; i < 13; i++) {
4   int j = (h + i) % 13;            // h, h+1, h+2…
5   if (tablo[j] == null) {        // boş mu?
6     tablo[j] = k;
7     break;
8   }                              // dolu: i++
9 }
```

DEĞİŞKENLER

k = 44 h = 5

Doğrusal Yoklama (Linear Probing)



i = 0: j = (5 + 0) mod 13 = **5** → yuva dolu (18), bir sonraki i.

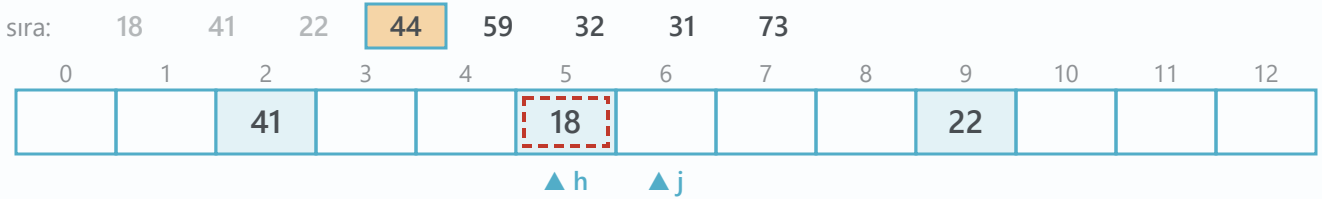
Adım 3 / 13

```
1 // yerlestir(k): doğrusal yoklama, N = 13
2 int h = k % 13;                  // ev yuvası
3 for (int i = 0; i < 13; i++) {
▶   int j = (h + i) % 13;            // h, h+1, h+2…
▶   if (tablo[j] == null) {        // boş mu?
6     tablo[j] = k;
7     break;
8   }                              // dolu: i++
9 }
```

DEĞİŞKENLER

`k = 44` `h = 5` `i = 0` `j = 5` `tablo[j] = 18`

Doğrusal Yoklama (Linear Probing)



i = 1: j = (5 + 1) mod 13 = **6** → yuva **boş**.

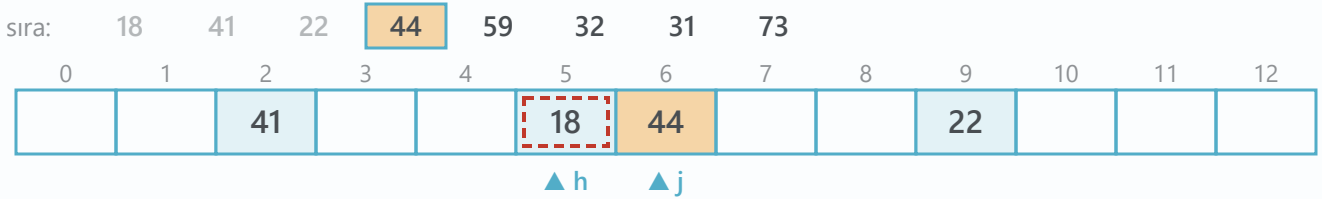

```
1 // yerlestir(k): doğrusal yoklama, N = 13
2 int h = k % 13;                  // ev yuvası
3 for (int i = 0; i < 13; i++) {
▶   int j = (h + i) % 13;           // h, h+1, h+2…
▶   if (tablo[j] == null) {        // boş mu?
6     tablo[j] = k;
7     break;
8   }                              // dolu: i++
9 }
```

DEĞİŞKENLER

`k = 44` `h = 5` `i = 1` `j = 6`

`tablo[j] = null`

Doğrusal Yoklama (Linear Probing)



`tablo[6] = 44` , `break` (2 yoklama).

Adım 5 / 13

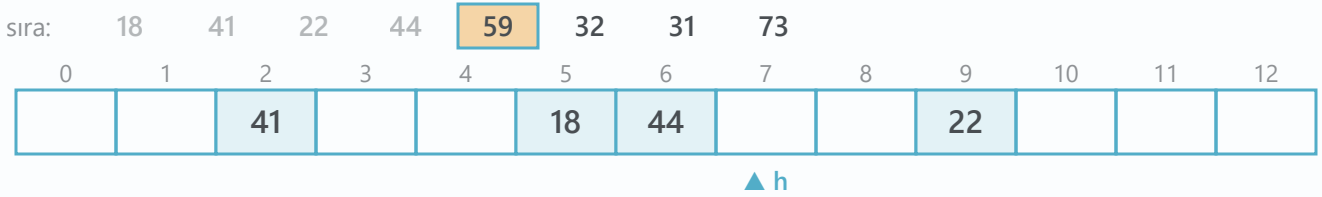
```
1 // yerlestir(k): doğrusal yoklama, N = 13
2 int h = k % 13;                  // ev yuvası
3 for (int i = 0; i < 13; i++) {
4   int j = (h + i) % 13;            // h, h+1, h+2…
5   if (tablo[j] == null) {        // boş mu?
▶     tablo[j] = k;
▶     break;
8   }                              // dolu: i++
9 }
```

DEĞİŞKENLER

`k = 44` `h = 5` `i = 1` `j = 6`

`tablo[j] = 44`

Doğrusal Yoklama (Linear Probing)



Sıradaki anahtar 59: **h = 59 mod 13 = 7** (ev yuvası).

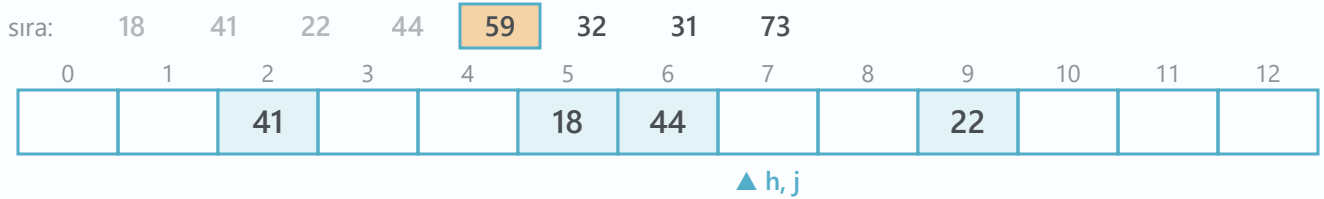
Adım 6 / 13

```
1 // yerlestir(k): doğrusal yoklama, N = 13
▶ int h = k % 13;                  // ev yuvası
3 for (int i = 0; i < 13; i++) {
4   int j = (h + i) % 13;            // h, h+1, h+2…
5   if (tablo[j] == null) {        // boş mu?
6     tablo[j] = k;
7     break;
8   }                              // dolu: i++
9 }
```

DEĞİŞKENLER

`k = 59` `h = 7`

Doğrusal Yoklama (Linear Probing)



i = 0: j = (7 + 0) mod 13 = **7** → yuva **boş**.

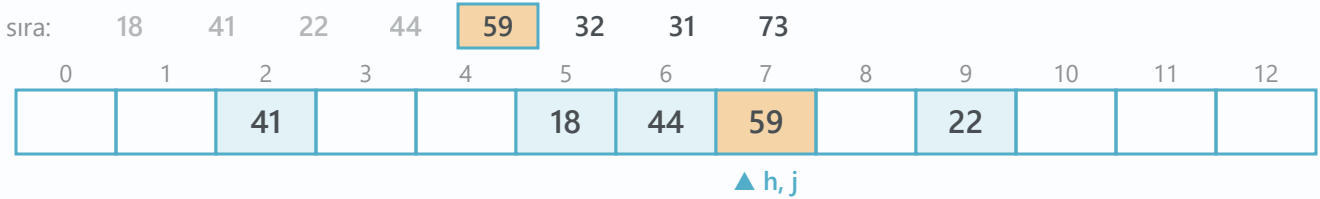
Adım 7 / 13

```
1 // yerlestir(k): doğrusal yoklama, N = 13
2 int h = k % 13;                  // ev yuvası
3 for (int i = 0; i < 13; i++) {
▶   int j = (h + i) % 13;            // h, h+1, h+2…
▶   if (tablo[j] == null) {        // boş mu?
6     tablo[j] = k;
7     break;
8   }                              // dolu: i++
9 }
```

DEĞİŞKENLER

`k = 59` `h = 7` `i = 0` `j = 7` `tablo[j] = null`

Doğrusal Yoklama (Linear Probing)



`tablo[7] = 59` , `break` (1 yoklama).

Adım 8 / 13

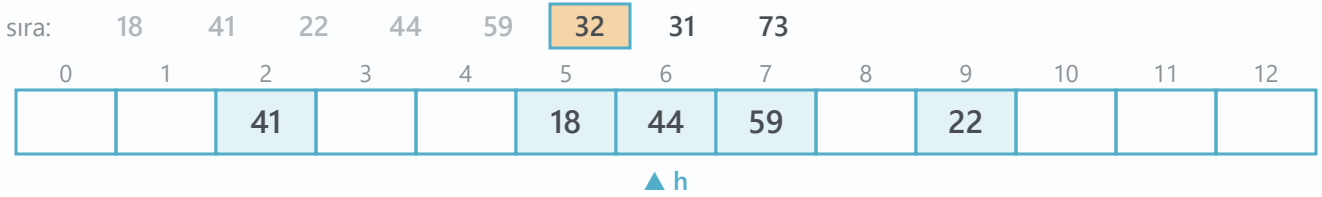
```
1 // yerlestir(k): doğrusal yoklama, N = 13
2 int h = k % 13;                  // ev yuvası
3 for (int i = 0; i < 13; i++) {
4   int j = (h + i) % 13;            // h, h+1, h+2…
5   if (tablo[j] == null) {        // boş mu?
▶     tablo[j] = k;
▶     break;
8   }                              // dolu: i++
9 }
```

DEĞİŞKENLER

`k = 59` `h = 7` `i = 0` `j = 7`

`tablo[j] = 59`

Doğrusal Yoklama (Linear Probing)



Sıradaki anahtar 32: **h = 32 mod 13 = 6** (ev yuvası).

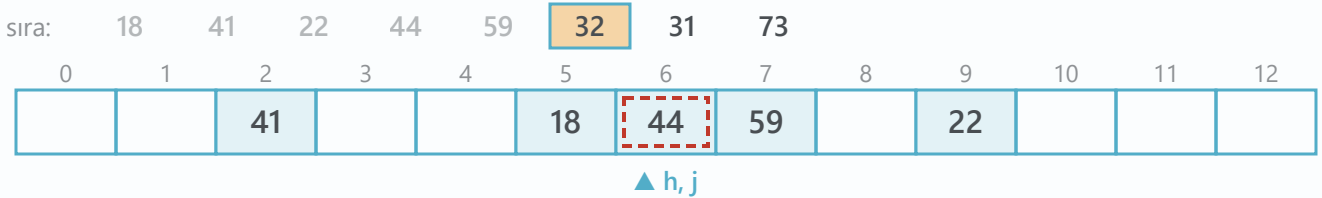
Adım 9 / 13

```
1 // yerlestir(k): doğrusal yoklama, N = 13
▶ int h = k % 13;                  // ev yuvası
3 for (int i = 0; i < 13; i++) {
4   int j = (h + i) % 13;            // h, h+1, h+2…
5   if (tablo[j] == null) {        // boş mu?
6     tablo[j] = k;
7     break;
8   }                              // dolu: i++
9 }
```

DEĞİŞKENLER

k = 32 h = 6

Doğrusal Yoklama (Linear Probing)



i = 0: j = (6 + 0) mod 13 = **6** → yuva dolu (44), bir sonraki i.

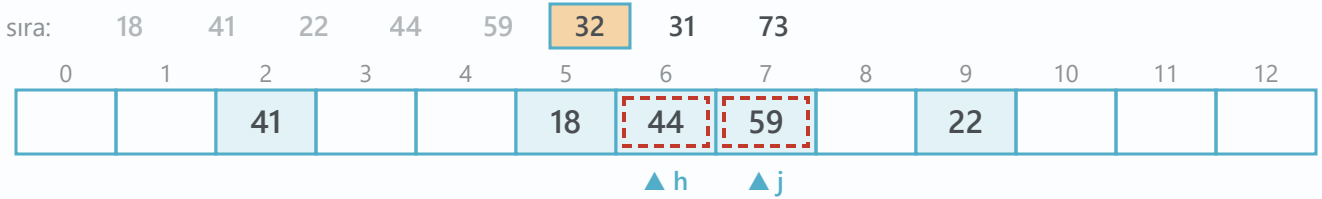
Adım 10 / 13

```
1 // yerlestir(k): doğrusal yoklama, N = 13
2 int h = k % 13;                  // ev yuvası
3 for (int i = 0; i < 13; i++) {
▶   int j = (h + i) % 13;            // h, h+1, h+2…
▶   if (tablo[j] == null) {        // boş mu?
6     tablo[j] = k;
7     break;
8   }                              // dolu: i++
9 }
```

DEĞİŞKENLER

`k = 32` `h = 6` `i = 0` `j = 6` `tablo[j] = 44`

Doğrusal Yoklama (Linear Probing)



i = 1: j = (6 + 1) mod 13 = **7** → yuva dolu (59), bir sonraki i.

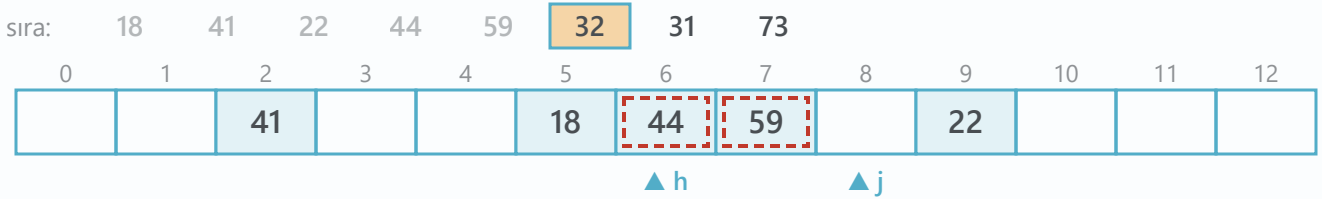

```
1 // yerlestir(k): doğrusal yoklama, N = 13
2 int h = k % 13;                  // ev yuvası
3 for (int i = 0; i < 13; i++) {
▶   int j = (h + i) % 13;           // h, h+1, h+2…
▶   if (tablo[j] == null) {        // boş mu?
6     tablo[j] = k;
7     break;
8   }                              // dolu: i++
9 }
```

DEĞİŞKENLER

`k = 32` `h = 6` `i = 1` `j = 7`

`tablo[j] = 59`

Doğrusal Yoklama (Linear Probing)



i = 2: j = (6 + 2) mod 13 = **8** → yuva **boş**.

Adım 12 / 13

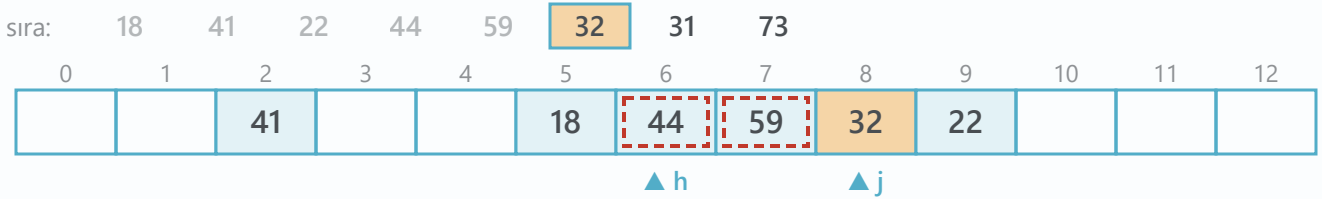
```
1 // yerlestir(k): doğrusal yoklama, N = 13
2 int h = k % 13;                  // ev yuvası
3 for (int i = 0; i < 13; i++) {
▶   int j = (h + i) % 13;           // h, h+1, h+2…
▶   if (tablo[j] == null) {        // boş mu?
6     tablo[j] = k;
7     break;
8   }                              // dolu: i++
9 }
```

DEĞİŞKENLER

`k = 32` `h = 6` `i = 2` `j = 8`

`tablo[j] = null`

Doğrusal Yoklama (Linear Probing)



`tablo[8] = 32` , `break` (3 yoklama).

Adım 13 / 13

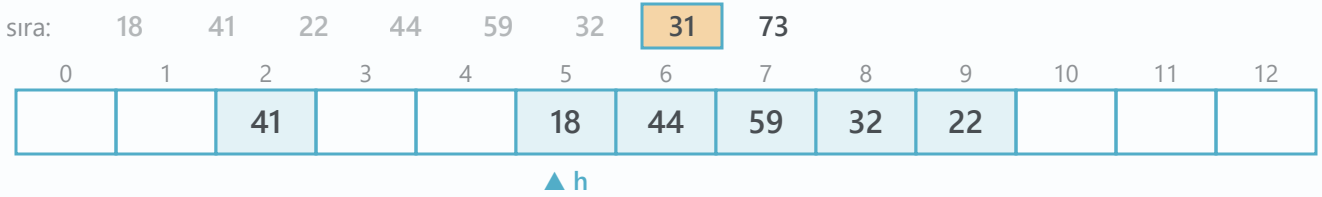
```
1 // yerlestir(k): doğrusal yoklama, N = 13
2 int h = k % 13;                  // ev yuvası
3 for (int i = 0; i < 13; i++) {
4   int j = (h + i) % 13;            // h, h+1, h+2…
5   if (tablo[j] == null) {        // boş mu?
▶     tablo[j] = k;
▶     break;
8   }                              // dolu: i++
9 }
```

DEĞİŞKENLER

`k = 32` `h = 6` `i = 2` `j = 8`

`tablo[j] = 32`

Doğrusal Yoklama (devam): 31 ve 73



Sıradaki anahtar 31: **h = 31 mod 13 = 5** (ev yuvası).

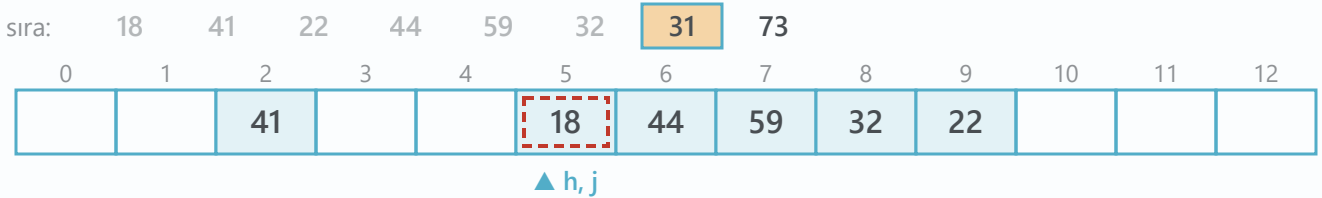
Adım 1 / 14

```
1 // yerlestir(k): doğrusal yoklama, N = 13
▶ int h = k % 13;                  // ev yuvası
3 for (int i = 0; i < 13; i++) {
4   int j = (h + i) % 13;            // h, h+1, h+2…
5   if (tablo[j] == null) {        // boş mu?
6     tablo[j] = k;
7     break;
8   }                              // dolu: i++
9 }
```

DEĞİŞKENLER

k = **31** h = **5**

Doğrusal Yoklama (devam): 31 ve 73



i = 0: j = (5 + 0) mod 13 = **5** → yuva dolu (18), bir sonraki i.

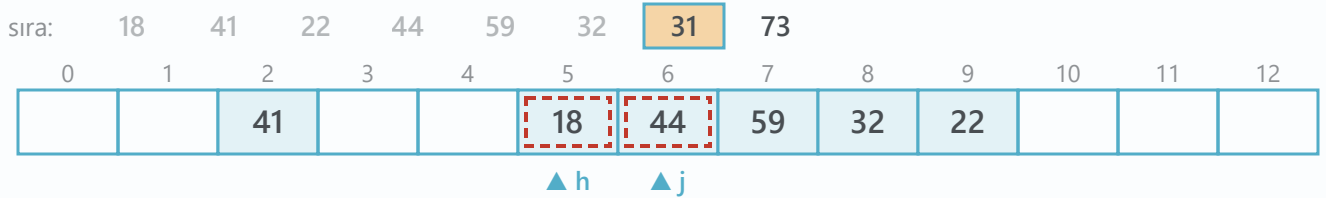
Adım 2 / 14

```
1 // yerlestir(k): doğrusal yoklama, N = 13
2 int h = k % 13;                  // ev yuvası
3 for (int i = 0; i < 13; i++) {
▶   int j = (h + i) % 13;           // h, h+1, h+2…
▶   if (tablo[j] == null) {        // boş mu?
6     tablo[j] = k;
7     break;
8   }                              // dolu: i++
9 }
```

DEĞİŞKENLER

`k = 31` `h = 5` `i = 0` `j = 5` `tablo[j] = 18`

Doğrusal Yoklama (devam): 31 ve 73



i = 1: j = (5 + 1) mod 13 = **6** → yuva dolu (44), bir sonraki i.

Adım 3 / 14

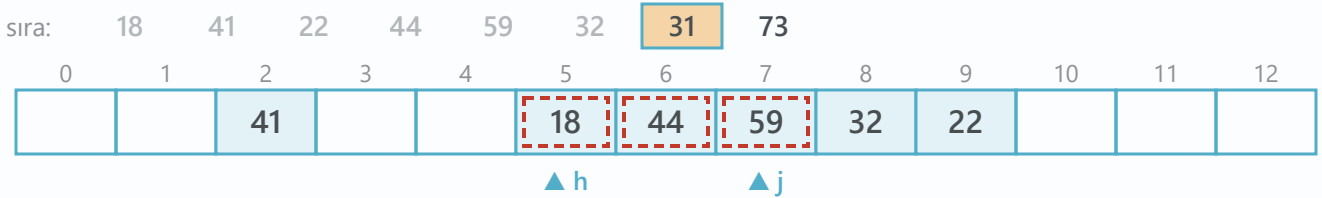
```
1 // yerlestir(k): doğrusal yoklama, N = 13
2 int h = k % 13;                  // ev yuvası
3 for (int i = 0; i < 13; i++) {
▶   int j = (h + i) % 13;            // h, h+1, h+2…
▶   if (tablo[j] == null) {        // boş mu?
6     tablo[j] = k;
7     break;
8   }                              // dolu: i++
9 }
```

DEĞİŞKENLER

`k = 31` `h = 5` `i = 1` `j = 6`

`tablo[j] = 44`

Doğrusal Yoklama (devam): 31 ve 73



i = 2: j = (5 + 2) mod 13 = **7** → yuva dolu (59), bir sonraki i.

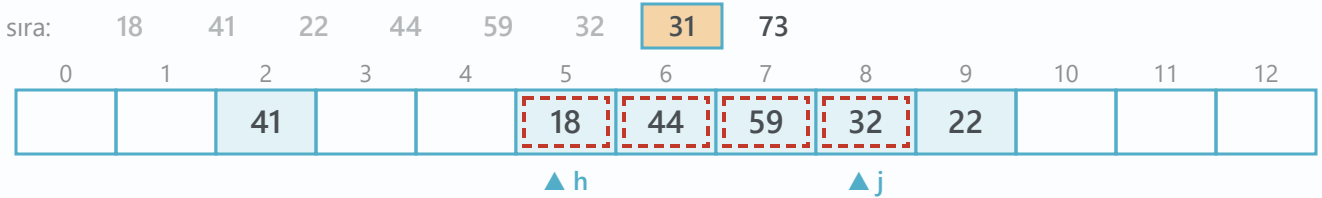
Adım 4 / 14

```
1 // yerlestir(k): doğrusal yoklama, N = 13
2 int h = k % 13;                  // ev yuvası
3 for (int i = 0; i < 13; i++) {
▶   int j = (h + i) % 13;           // h, h+1, h+2…
▶   if (tablo[j] == null) {        // boş mu?
6     tablo[j] = k;
7     break;
8   }                              // dolu: i++
9 }
```

DEĞİŞKENLER

`k = 31` `h = 5` `i = 2` `j = 7` `tablo[j] = 59`

Doğrusal Yoklama (devam): 31 ve 73



i = 3: j = (5 + 3) mod 13 = **8** → yuva dolu (32), bir sonraki i.

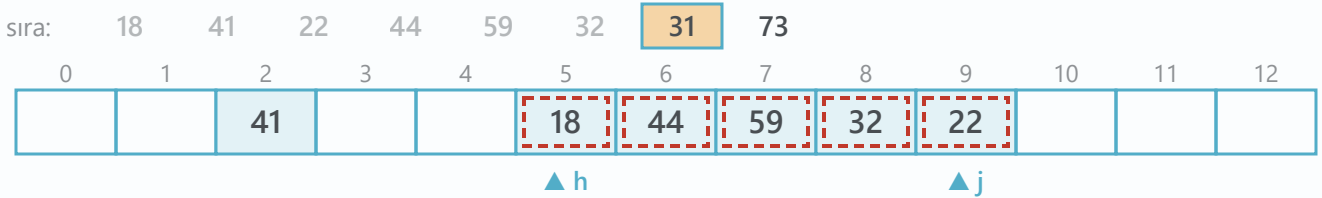
Adım 5 / 14

```
1 // yerlestir(k): doğrusal yoklama, N = 13
2 int h = k % 13;                  // ev yuvası
3 for (int i = 0; i < 13; i++) {
▶   int j = (h + i) % 13;           // h, h+1, h+2…
▶   if (tablo[j] == null) {        // boş mu?
6     tablo[j] = k;
7     break;
8   }                              // dolu: i++
9 }
```

DEĞİŞKENLER

`k = 31` `h = 5` `i = 3` `j = 8` `tablo[j] = 32`

Doğrusal Yoklama (devam): 31 ve 73



i = 4: j = (5 + 4) mod 13 = **9** → yuva dolu (22), bir sonraki i.

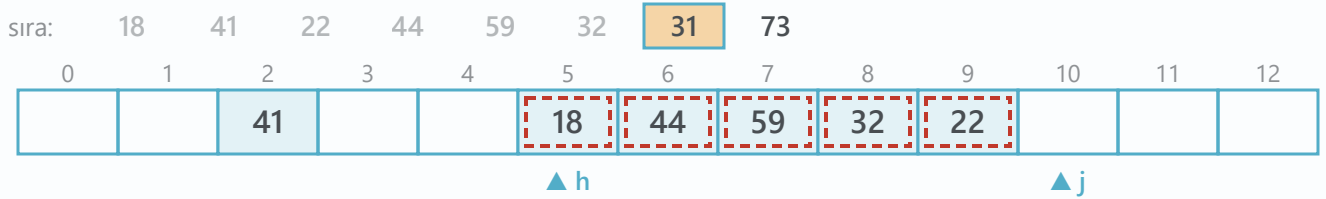
Adım 6 / 14

```
1 // yerlestir(k): doğrusal yoklama, N = 13
2 int h = k % 13;                  // ev yuvası
3 for (int i = 0; i < 13; i++) {
▶   int j = (h + i) % 13;           // h, h+1, h+2…
▶   if (tablo[j] == null) {        // boş mu?
6     tablo[j] = k;
7     break;
8   }                              // dolu: i++
9 }
```

DEĞİŞKENLER

`k = 31` `h = 5` `i = 4` `j = 9` `tablo[j] = 22`

Doğrusal Yoklama (devam): 31 ve 73



i = 5: j = (5 + 5) mod 13 = **10** → yuva **boş**.

Adım 7 / 14

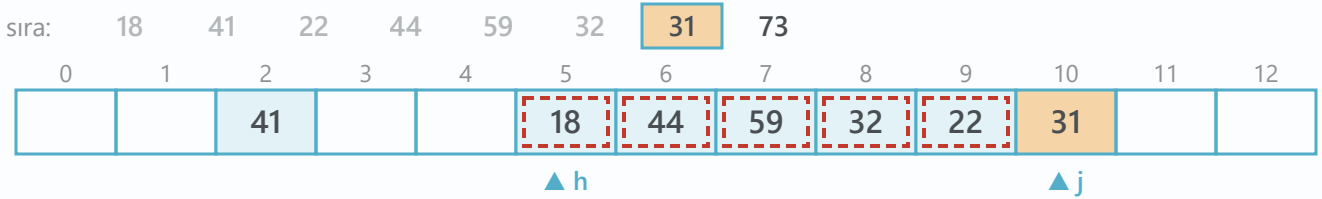
```
1 // yerlestir(k): doğrusal yoklama, N = 13
2 int h = k % 13;                  // ev yuvası
3 for (int i = 0; i < 13; i++) {
▶   int j = (h + i) % 13;           // h, h+1, h+2…
▶   if (tablo[j] == null) {        // boş mu?
6     tablo[j] = k;
7     break;
8   }                              // dolu: i++
9 }
```

DEĞİŞKENLER

`k = 31` `h = 5` `i = 5` `j = 10`

`tablo[j] = null`

Doğrusal Yoklama (devam): 31 ve 73



`tablo[10] = 31`, `break` (6 yoklama).

Adım 8 / 14

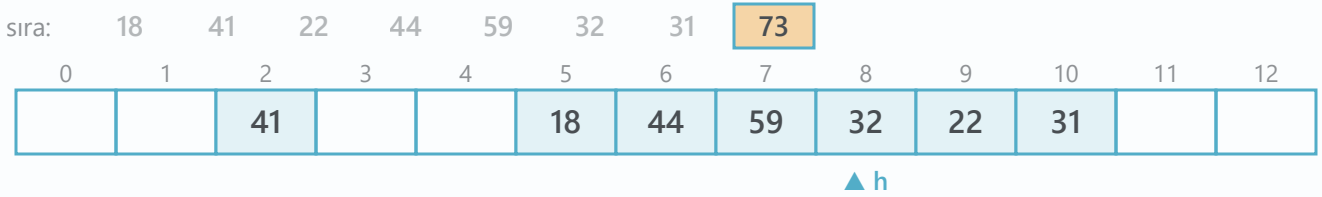
```
1 // yerlestir(k): doğrusal yoklama, N = 13
2 int h = k % 13;                  // ev yuvası
3 for (int i = 0; i < 13; i++) {
4   int j = (h + i) % 13;            // h, h+1, h+2…
5   if (tablo[j] == null) {        // boş mu?
▶     tablo[j] = k;
▶     break;
8   }                              // dolu: i++
9 }
```

DEĞİŞKENLER

`k = 31` `h = 5` `i = 5` `j = 10`

`tablo[j] = 31`

Doğrusal Yoklama (devam): 31 ve 73



Sıradaki anahtar 73: **h = 73 mod 13 = 8** (ev yuvası).

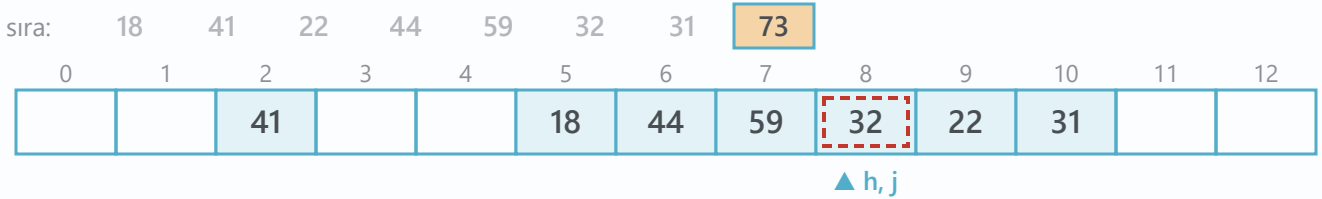
Adım 9 / 14

```
1 // yerlestir(k): doğrusal yoklama, N = 13
▶ int h = k % 13;                  // ev yuvası
3 for (int i = 0; i < 13; i++) {
4   int j = (h + i) % 13;            // h, h+1, h+2…
5   if (tablo[j] == null) {        // boş mu?
6     tablo[j] = k;
7     break;
8   }                              // dolu: i++
9 }
```

DEĞİŞKENLER

k = 73 h = 8

Doğrusal Yoklama (devam): 31 ve 73



i = 0: j = (8 + 0) mod 13 = **8** → yuva dolu (32), bir sonraki i.

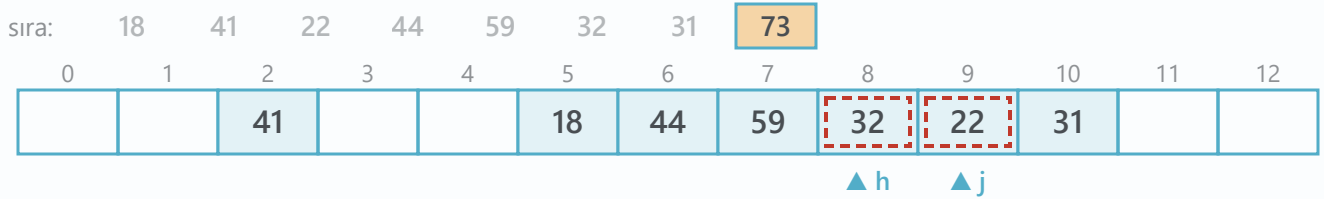
Adım 10 / 14

```
1 // yerlestir(k): doğrusal yoklama, N = 13
2 int h = k % 13;                  // ev yuvası
3 for (int i = 0; i < 13; i++) {
▶   int j = (h + i) % 13;            // h, h+1, h+2…
▶   if (tablo[j] == null) {        // boş mu?
6     tablo[j] = k;
7     break;
8   }                              // dolu: i++
9 }
```

DEĞİŞKENLER

`k = 73` `h = 8` `i = 0` `j = 8` `tablo[j] = 32`

Doğrusal Yoklama (devam): 31 ve 73



i = 1: j = (8 + 1) mod 13 = **9** → yuva dolu (22), bir sonraki i.

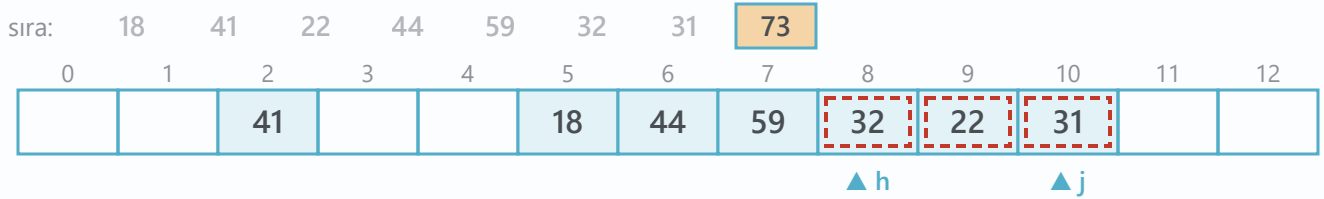
Adım 11 / 14

```
1 // yerlestir(k): doğrusal yoklama, N = 13
2 int h = k % 13;                  // ev yuvası
3 for (int i = 0; i < 13; i++) {
▶   int j = (h + i) % 13;           // h, h+1, h+2…
▶   if (tablo[j] == null) {        // boş mu?
6     tablo[j] = k;
7     break;
8   }                              // dolu: i++
9 }
```

DEĞİŞKENLER

`k = 73` `h = 8` `i = 1` `j = 9` `tablo[j] = 22`

Doğrusal Yoklama (devam): 31 ve 73



i = 2: j = (8 + 2) mod 13 = **10** → yuva dolu (31), bir sonraki i.

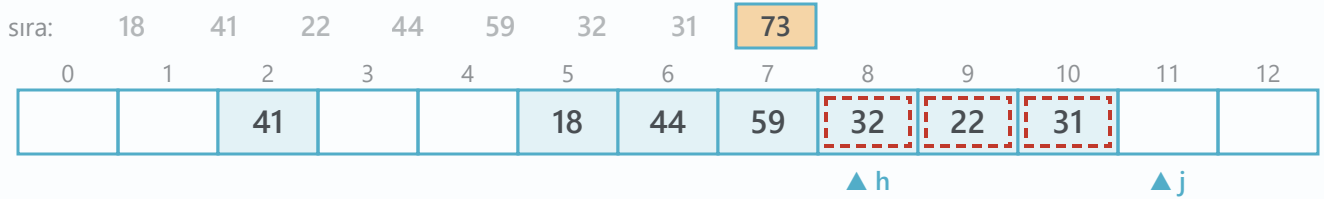
Adım 12 / 14

```
1 // yerlestir(k): doğrusal yoklama, N = 13
2 int h = k % 13;                  // ev yuvası
3 for (int i = 0; i < 13; i++) {
▶   int j = (h + i) % 13;            // h, h+1, h+2…
▶   if (tablo[j] == null) {        // boş mu?
6     tablo[j] = k;
7     break;
8   }                              // dolu: i++
9 }
```

DEĞİŞKENLER

`k = 73` `h = 8` `i = 2` `j = 10` `tablo[j] = 31`

Doğrusal Yoklama (devam): 31 ve 73



i = 3: j = (8 + 3) mod 13 = **11** → yuva **boş**.

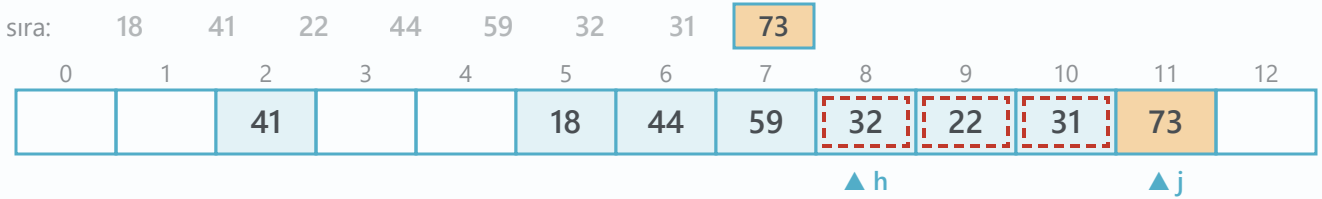

```
1 // yerlestir(k): doğrusal yoklama, N = 13
2 int h = k % 13;                  // ev yuvası
3 for (int i = 0; i < 13; i++) {
▶   int j = (h + i) % 13;           // h, h+1, h+2…
▶   if (tablo[j] == null) {        // boş mu?
6     tablo[j] = k;
7     break;
8   }                              // dolu: i++
9 }
```

DEĞİŞKENLER

`k = 73` `h = 8` `i = 3` `j = 11`

`tablo[j] = null`

Doğrusal Yoklama (devam): 31 ve 73



`tablo[11] = 73`, `break` (4 yoklama). 5–11 tek bir büyük küme: **birincil kümelenme**.

Adım 14 / 14

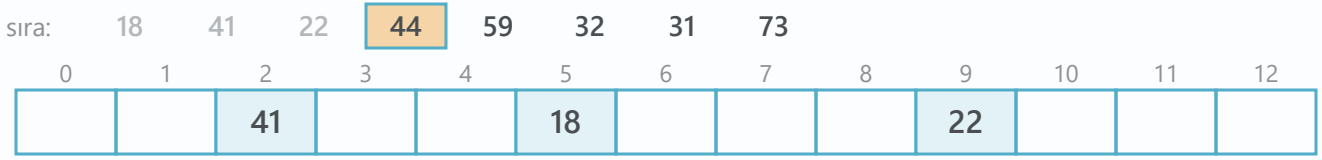
```
1 // yerlestir(k): doğrusal yoklama, N = 13
2 int h = k % 13;                  // ev yuvası
3 for (int i = 0; i < 13; i++) {
4   int j = (h + i) % 13;            // h, h+1, h+2…
5   if (tablo[j] == null) {        // boş mu?
▶     tablo[j] = k;
▶     break;
8   }                              // dolu: i++
9 }
```

DEĞİŞKENLER

`k = 73` `h = 8` `i = 3` `j = 11`

`tablo[j] = 73`

Karesel Yoklama (Quadratic Probing)



18 → 5, 41 → 2, 22 → 9: üçü de ilk denemede boş yuvaya yerleşti. Sırada 44 var.

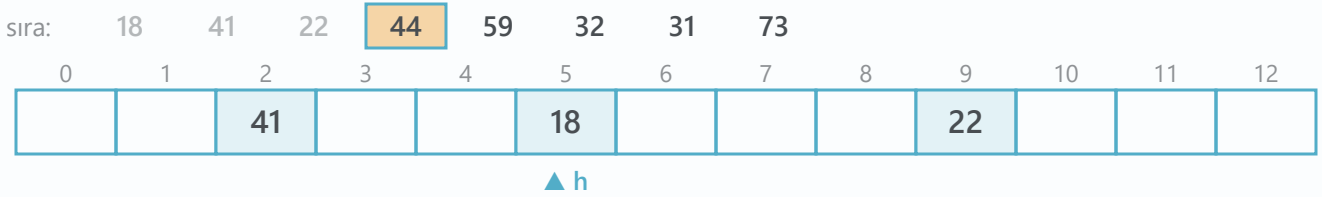
Adım 1 / 13

```
1 // yerlestir(k): karesel yoklama, N = 13
2 int h = k % 13;                  // ev yuvası
3 for (int i = 0; i < 13; i++) {
4   int j = (h + i * i) % 13;       // h, h+1, h+4…
5   if (tablo[j] == null) {        // boş mu?
6     tablo[j] = k;
7     break;
8   }                              // dolu: i++
9 }
```

DEĞİŞKENLER

`N = 13`

Karesel Yoklama (Quadratic Probing)



Sıradaki anahtar 44: **h = 44 mod 13 = 5** (ev yuvası).

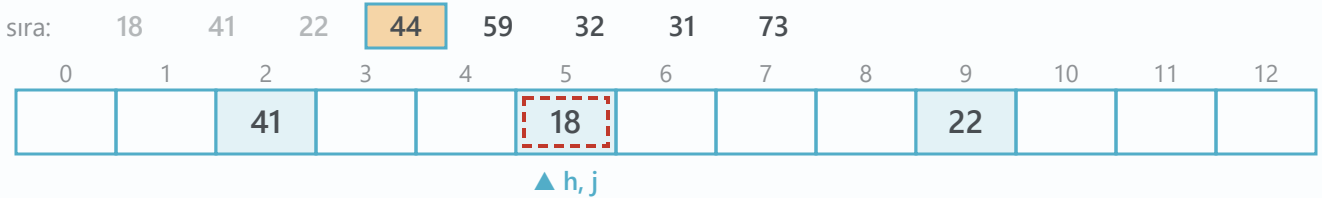
Adım 2 / 13

```
1 // yerlestir(k): karesel yoklama, N = 13
▶ int h = k % 13;                  // ev yuvası
3 for (int i = 0; i < 13; i++) {
4   int j = (h + i * i) % 13;       // h, h+1, h+4…
5   if (tablo[j] == null) {        // boş mu?
6     tablo[j] = k;
7     break;
8   }                              // dolu: i++
9 }
```

DEĞİŞKENLER

k = 44 h = 5

Karesel Yoklama (Quadratic Probing)



i = 0: j = (5 + 0^2) mod 13 = **5** → yuva dolu (18), bir sonraki i.

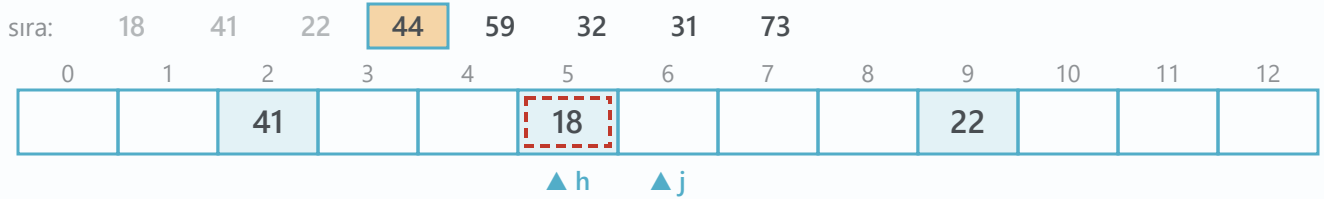
Adım 3 / 13

```
1 // yerlestir(k): karesel yoklama, N = 13
2 int h = k % 13;                  // ev yuvası
3 for (int i = 0; i < 13; i++) {
▶   int j = (h + i * i) % 13;       // h, h+1, h+4…
▶   if (tablo[j] == null) {        // boş mu?
6     tablo[j] = k;
7     break;
8   }                              // dolu: i++
9 }
```

DEĞİŞKENLER

`k = 44` `h = 5` `i = 0` `j = 5` `tablo[j] = 18`

Karesel Yoklama (Quadratic Probing)



i = 1: j = (5 + 1^2) mod 13 = **6** → yuva **boş**.

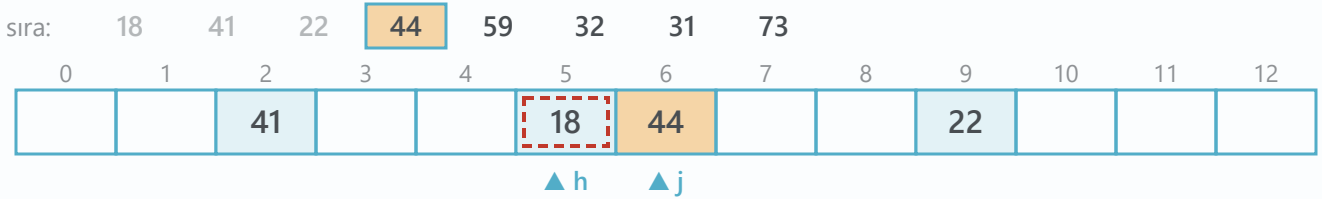

```
1 // yerlestir(k): karesel yoklama, N = 13
2 int h = k % 13;                  // ev yuvası
3 for (int i = 0; i < 13; i++) {
▶   int j = (h + i * i) % 13;       // h, h+1, h+4…
▶   if (tablo[j] == null) {        // boş mu?
6     tablo[j] = k;
7     break;
8   }                              // dolu: i++
9 }
```

DEĞİŞKENLER

`k = 44` `h = 5` `i = 1` `j = 6`

`tablo[j] = null`

Karesel Yoklama (Quadratic Probing)



`tablo[6] = 44` , `break` (2 yoklama).

Adım 5 / 13

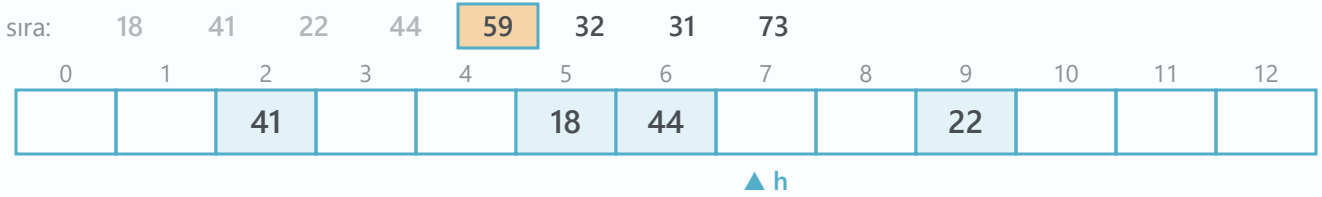
```
1 // yerlestir(k): karesel yoklama, N = 13
2 int h = k % 13;                  // ev yuvası
3 for (int i = 0; i < 13; i++) {
4   int j = (h + i * i) % 13;       // h, h+1, h+4…
5   if (tablo[j] == null) {        // boş mu?
▶     tablo[j] = k;
▶     break;
8   }                              // dolu: i++
9 }
```

DEĞİŞKENLER

`k = 44` `h = 5` `i = 1` `j = 6`

`tablo[j] = 44`

Karesel Yoklama (Quadratic Probing)



Sıradaki anahtar 59: **h = 59 mod 13 = 7** (ev yuvası).

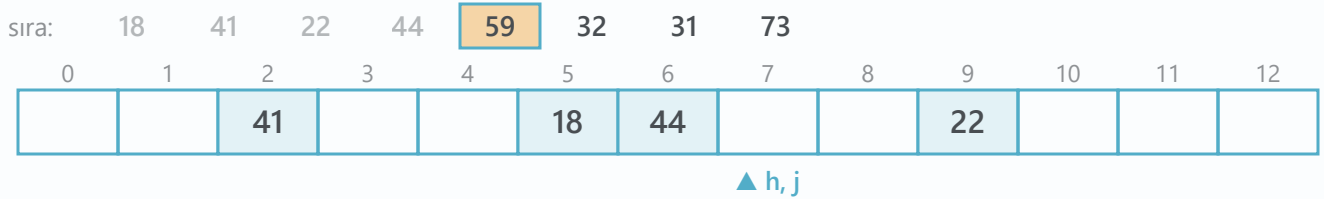
Adım 6 / 13

```
1 // yerlestir(k): karesel yoklama, N = 13
▶ int h = k % 13;                  // ev yuvası
3 for (int i = 0; i < 13; i++) {
4   int j = (h + i * i) % 13;       // h, h+1, h+4…
5   if (tablo[j] == null) {        // boş mu?
6     tablo[j] = k;
7     break;
8   }                              // dolu: i++
9 }
```

DEĞİŞKENLER

`k = 59` `h = 7`

Karesel Yoklama (Quadratic Probing)



i = 0: j = (7 + 0^2) mod 13 = **7** → yuva **boş**.

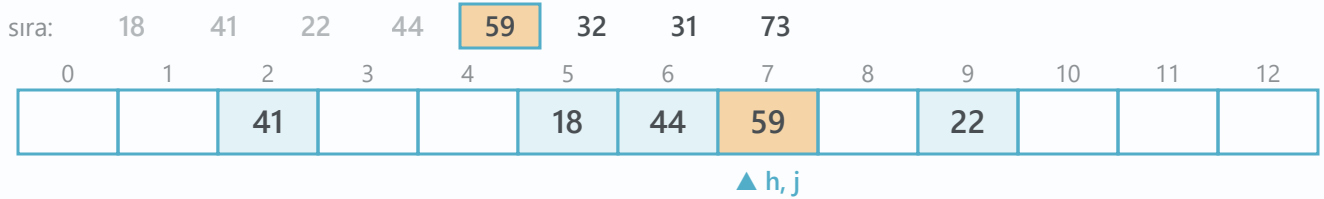
Adım 7 / 13

```
1 // yerlestir(k): karesel yoklama, N = 13
2 int h = k % 13;                  // ev yuvası
3 for (int i = 0; i < 13; i++) {
▶   int j = (h + i * i) % 13;       // h, h+1, h+4…
▶   if (tablo[j] == null) {        // boş mu?
6     tablo[j] = k;
7     break;
8   }                              // dolu: i++
9 }
```

DEĞİŞKENLER

`k = 59` `h = 7` `i = 0` `j = 7` `tablo[j] = null`

Karesel Yoklama (Quadratic Probing)



`tablo[7] = 59` , `break` (1 yoklama).

Adım 8 / 13

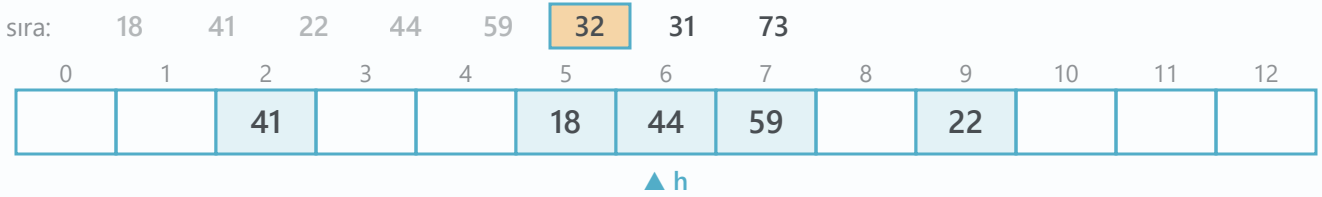
```
1 // yerlestir(k): karesel yoklama, N = 13
2 int h = k % 13;                  // ev yuvası
3 for (int i = 0; i < 13; i++) {
4   int j = (h + i * i) % 13;       // h, h+1, h+4…
5   if (tablo[j] == null) {         // boş mu?
▶     tablo[j] = k;
▶     break;
8   }                              // dolu: i++
9 }
```

DEĞİŞKENLER

`k = 59` `h = 7` `i = 0` `j = 7`

`tablo[j] = 59`

Karesel Yoklama (Quadratic Probing)



Sıradaki anahtar 32: **h = 32 mod 13 = 6** (ev yuvası).

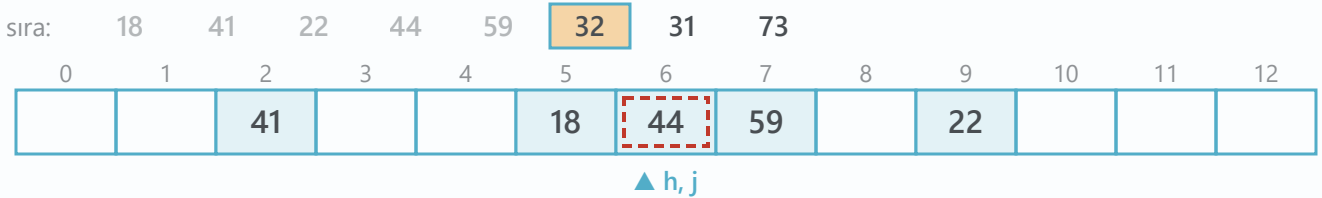
Adım 9 / 13

```
1 // yerlestir(k): karesel yoklama, N = 13
▶ int h = k % 13;                  // ev yuvası
3 for (int i = 0; i < 13; i++) {
4   int j = (h + i * i) % 13;       // h, h+1, h+4…
5   if (tablo[j] == null) {        // boş mu?
6     tablo[j] = k;
7     break;
8   }                              // dolu: i++
9 }
```

DEĞİŞKENLER

k = 32 h = 6

Karesel Yoklama (Quadratic Probing)



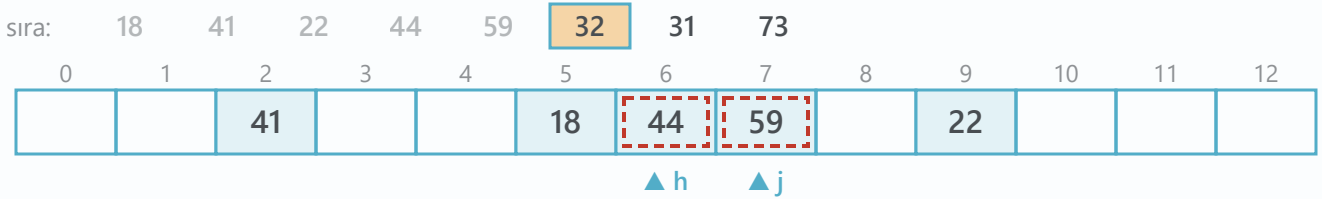
i = 0: j = (6 + 0^2) mod 13 = **6** → yuva dolu (44), bir sonraki i.


```
1 // yerlestir(k): karesel yoklama, N = 13
2 int h = k % 13;                  // ev yuvası
3 for (int i = 0; i < 13; i++) {
▶   int j = (h + i * i) % 13;       // h, h+1, h+4…
▶   if (tablo[j] == null) {        // boş mu?
6     tablo[j] = k;
7     break;
8   }                              // dolu: i++
9 }
```

DEĞİŞKENLER

`k = 32` `h = 6` `i = 0` `j = 6` `tablo[j] = 44`

Karesel Yoklama (Quadratic Probing)



i = 1: j = (6 + 1^2) mod 13 = **7** → yuva dolu (59), bir sonraki i.

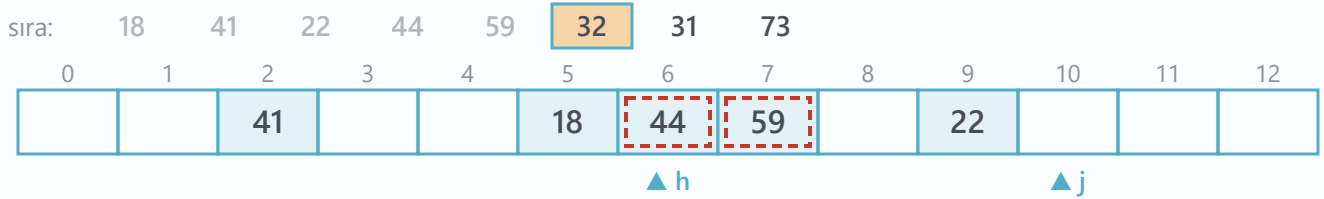

```
1 // yerlestir(k): karesel yoklama, N = 13
2 int h = k % 13;                  // ev yuvası
3 for (int i = 0; i < 13; i++) {
▶   int j = (h + i * i) % 13;       // h, h+1, h+4…
▶   if (tablo[j] == null) {        // boş mu?
6     tablo[j] = k;
7     break;
8   }                              // dolu: i++
9 }
```

DEĞİŞKENLER

`k = 32` `h = 6` `i = 1` `j = 7`

`tablo[j] = 59`

Karesel Yoklama (Quadratic Probing)



i = 2: j = (6 + 2^2) mod 13 = **10** → yuva **boş**.

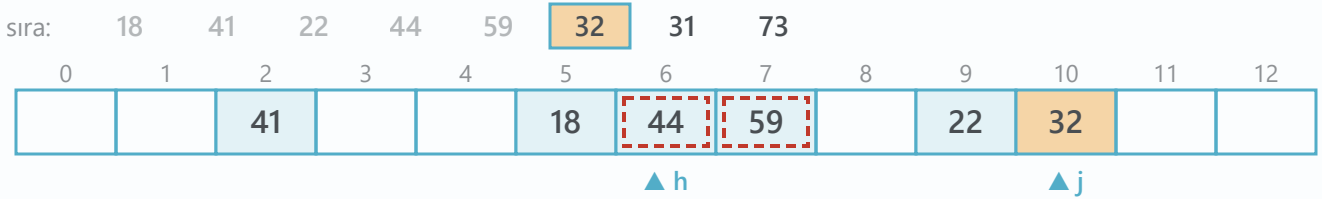

```
1 // yerlestir(k): karesel yoklama, N = 13
2 int h = k % 13;                  // ev yuvası
3 for (int i = 0; i < 13; i++) {
▶   int j = (h + i * i) % 13;       // h, h+1, h+4…
▶   if (tablo[j] == null) {        // boş mu?
6     tablo[j] = k;
7     break;
8   }                              // dolu: i++
9 }
```

DEĞİŞKENLER

`k = 32` `h = 6` `i = 2` `j = 10`

`tablo[j] = null`

Karesel Yoklama (Quadratic Probing)



`tablo[10] = 32` , `break` (3 yoklama).

Adım 13 / 13

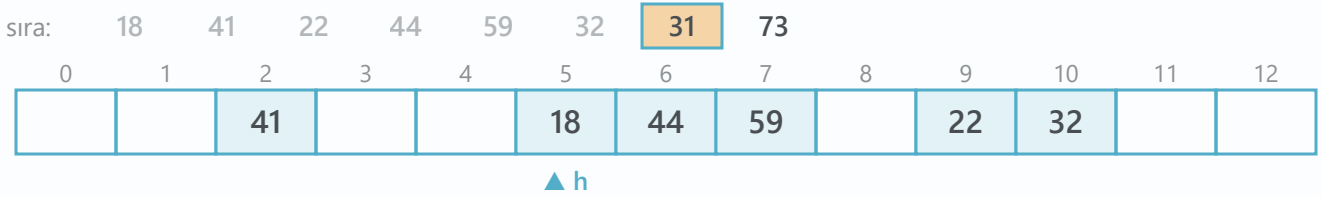
```
1 // yerlestir(k): karesel yoklama, N = 13
2 int h = k % 13;                  // ev yuvası
3 for (int i = 0; i < 13; i++) {
4   int j = (h + i * i) % 13;       // h, h+1, h+4…
5   if (tablo[j] == null) {         // boş mu?
▶     tablo[j] = k;
▶     break;
8   }                              // dolu: i++
9 }
```

DEĞİŞKENLER

`k = 32` `h = 6` `i = 2` `j = 10`

`tablo[j] = 32`

Karesel Yoklama (devam): 31 ve 73



Sıradaki anahtar 31: **h = 31 mod 13 = 5** (ev yuvası).

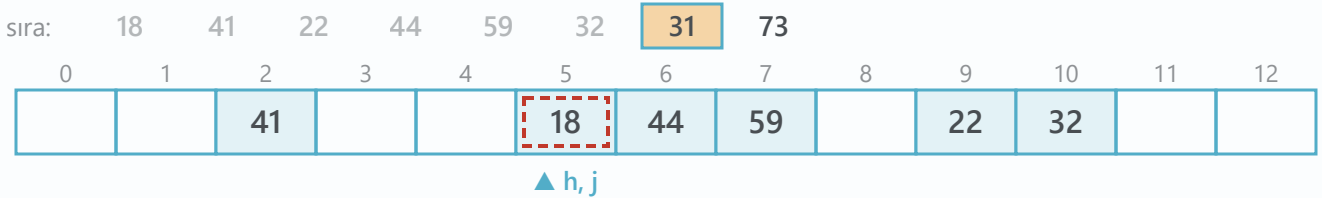
Adım 1 / 9

```
1 // yerlestir(k): karesel yoklama, N = 13
▶ int h = k % 13;                  // ev yuvası
3 for (int i = 0; i < 13; i++) {
4   int j = (h + i * i) % 13;       // h, h+1, h+4…
5   if (tablo[j] == null) {        // boş mu?
6     tablo[j] = k;
7     break;
8   }                              // dolu: i++
9 }
```

DEĞİŞKENLER

k = **31** h = **5**

Karesel Yoklama (devam): 31 ve 73



i = 0: j = (5 + 0^2) mod 13 = **5** → yuva dolu (18), bir sonraki i.

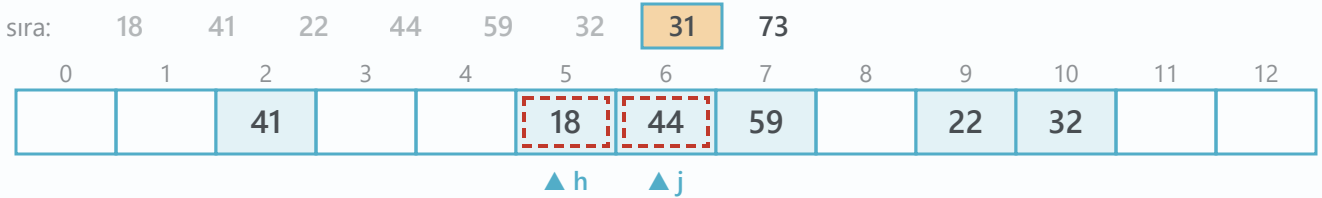
Adım 2 / 9

```
1 // yerlestir(k): karesel yoklama, N = 13
2 int h = k % 13;                  // ev yuvası
3 for (int i = 0; i < 13; i++) {
▶   int j = (h + i * i) % 13;       // h, h+1, h+4…
▶   if (tablo[j] == null) {        // boş mu?
6     tablo[j] = k;
7     break;
8   }                              // dolu: i++
9 }
```

DEĞİŞKENLER

`k = 31` `h = 5` `i = 0` `j = 5` `tablo[j] = 18`

Karesel Yoklama (devam): 31 ve 73



i = 1: j = (5 + 1^2) mod 13 = **6** → yuva dolu (44), bir sonraki i.

Adım 3 / 9

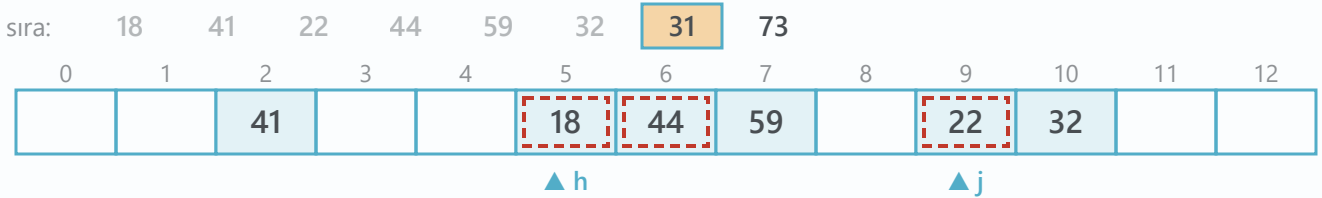
```
1 // yerlestir(k): karesel yoklama, N = 13
2 int h = k % 13;                  // ev yuvası
3 for (int i = 0; i < 13; i++) {
▶   int j = (h + i * i) % 13;       // h, h+1, h+4…
▶   if (tablo[j] == null) {        // boş mu?
6     tablo[j] = k;
7     break;
8   }                              // dolu: i++
9 }
```

DEĞİŞKENLER

`k = 31` `h = 5` `i = 1` `j = 6`

`tablo[j] = 44`

Karesel Yoklama (devam): 31 ve 73



i = 2: j = (5 + 2^2) mod 13 = **9** → yuva dolu (22), bir sonraki i.

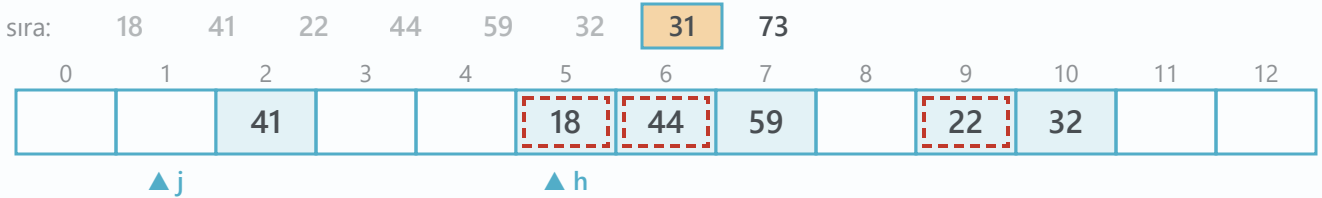
Adım 4 / 9

```
1 // yerlestir(k): karesel yoklama, N = 13
2 int h = k % 13;                  // ev yuvası
3 for (int i = 0; i < 13; i++) {
▶   int j = (h + i * i) % 13;       // h, h+1, h+4…
▶   if (tablo[j] == null) {        // boş mu?
6     tablo[j] = k;
7     break;
8   }                              // dolu: i++
9 }
```

DEĞİŞKENLER

`k = 31` `h = 5` `i = 2` `j = 9` `tablo[j] = 22`

Karesel Yoklama (devam): 31 ve 73



i = 3: j = (5 + 3^2) mod 13 = **1** → yuva **boş**.

Adım 5 / 9

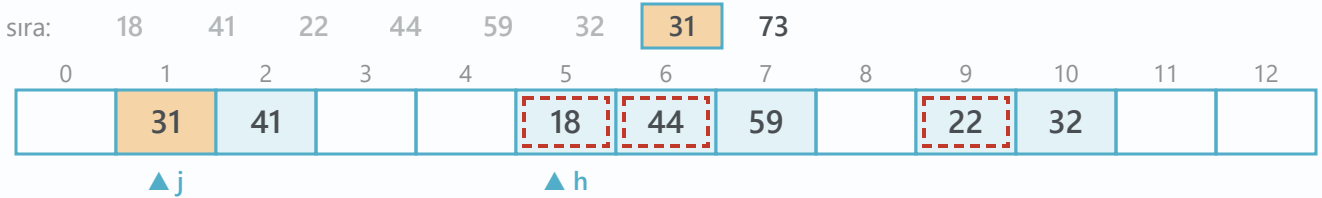
```
1 // yerlestir(k): karesel yoklama, N = 13
2 int h = k % 13;                  // ev yuvası
3 for (int i = 0; i < 13; i++) {
▶   int j = (h + i * i) % 13;       // h, h+1, h+4…
▶   if (tablo[j] == null) {        // boş mu?
6     tablo[j] = k;
7     break;
8   }                              // dolu: i++
9 }
```

DEĞİŞKENLER

`k = 31` `h = 5` `i = 3` `j = 1`

`tablo[j] = null`

Karesel Yoklama (devam): 31 ve 73



`tablo[1] = 31`, `break` (4 yoklama). Sıçramalar büyüdüğü için küme dışına çabuk çıkıldı.

Adım 6 / 9

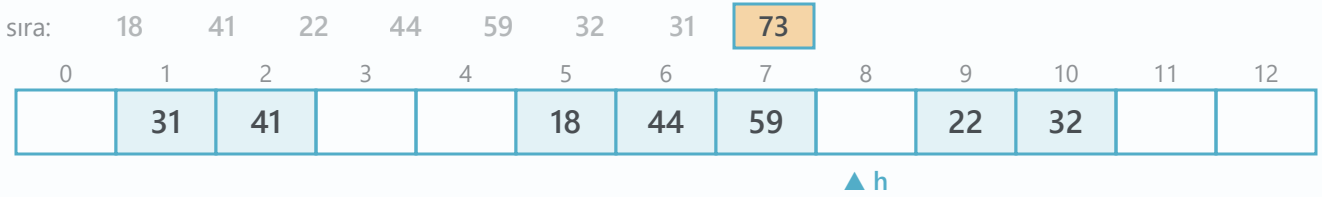
```
1 // yerlestir(k): karesel yoklama, N = 13
2 int h = k % 13;                  // ev yuvası
3 for (int i = 0; i < 13; i++) {
4   int j = (h + i * i) % 13;       // h, h+1, h+4…
5   if (tablo[j] == null) {        // boş mu?
▶     tablo[j] = k;
▶     break;
8   }                              // dolu: i++
9 }
```

DEĞİŞKENLER

`k = 31` `h = 5` `i = 3` `j = 1`

`tablo[j] = 31`

Karesel Yoklama (devam): 31 ve 73



Sıradaki anahtar 73: **h = 73 mod 13 = 8** (ev yuvası).

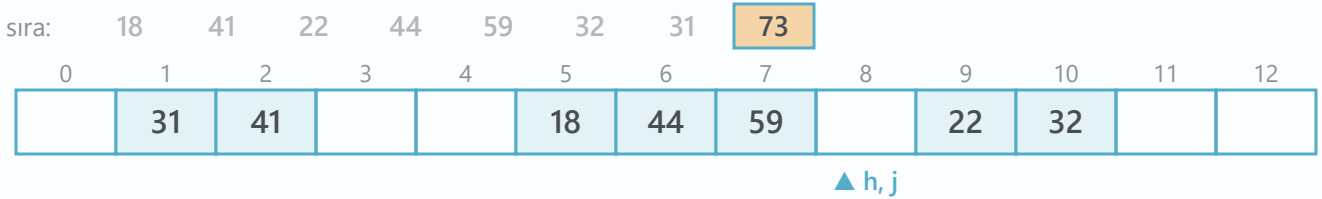
Adım 7 / 9

```
1 // yerlestir(k): karesel yoklama, N = 13
▶ int h = k % 13;                  // ev yuvası
3 for (int i = 0; i < 13; i++) {
4   int j = (h + i * i) % 13;       // h, h+1, h+4…
5   if (tablo[j] == null) {        // boş mu?
6     tablo[j] = k;
7     break;
8   }                              // dolu: i++
9 }
```

DEĞİŞKENLER

k = **73** h = **8**

Karesel Yoklama (devam): 31 ve 73



i = 0: j = (8 + 0^2) mod 13 = **8** → yuva **boş**.

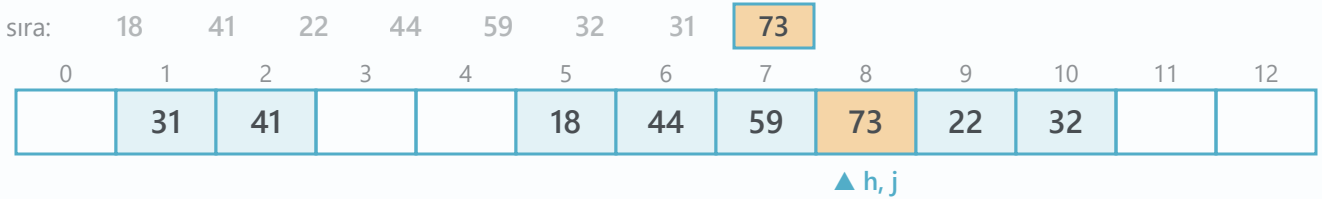

```
1 // yerlestir(k): karesel yoklama, N = 13
2 int h = k % 13;                  // ev yuvası
3 for (int i = 0; i < 13; i++) {
▶   int j = (h + i * i) % 13;       // h, h+1, h+4…
▶   if (tablo[j] == null) {        // boş mu?
6     tablo[j] = k;
7     break;
8   }                              // dolu: i++
9 }
```

DEĞİŞKENLER

`k = 73` `h = 8` `i = 0` `j = 8`

`tablo[j] = null`

Karesel Yoklama (devam): 31 ve 73



`tablo[8] = 73` , `break` (1 yoklama).

Adım 9 / 9

```
1 // yerlestir(k): karesel yoklama, N = 13
2 int h = k % 13;                  // ev yuvası
3 for (int i = 0; i < 13; i++) {
4   int j = (h + i * i) % 13;       // h, h+1, h+4…
5   if (tablo[j] == null) {        // boş mu?
▶     tablo[j] = k;
▶     break;
8   }                              // dolu: i++
9 }
```

DEĞİŞKENLER

`k = 73` `h = 8` `i = 0` `j = 8`

`tablo[j] = 73`

AÇIK ADRESLEME

Çift Hash (Double Hashing)

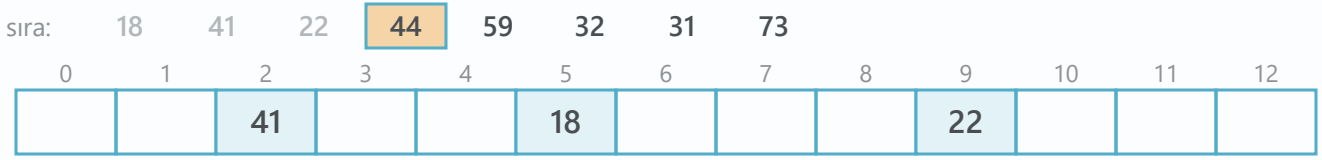
`h(k, i) = (h(k) + i · h₂(k)) mod 13` , `h₂(k) = 7 − (k mod 7)` → adım 1…7 arası, asla 0 değil.

k	18	41	22	44	59	32	31	73
h(k) = k mod 13	5	2	9	5	7	6	5	8
$h_2(k)$ (çakışmada)	—	—	—	7−2 = 5	—	—	7−3 = 4	—
Denenen yuvalar	5	2	9	5, 10	7	6	5, 9, 0	8



44: 5 dolu → 5+5 = **10**. 31: 5 dolu → 5+4 = 9 dolu → 5+8 = 13 mod 13 = **0**. Aynı ev yuvasına düşen anahtarlar **farklı adımlarla** ilerler; kümelenme azalır.

Kod İzleme: Çift Hash



18 → 5, 41 → 2, 22 → 9: üçü de ilk denemede boş yuvaya yerleşti. Sırada 44 var.

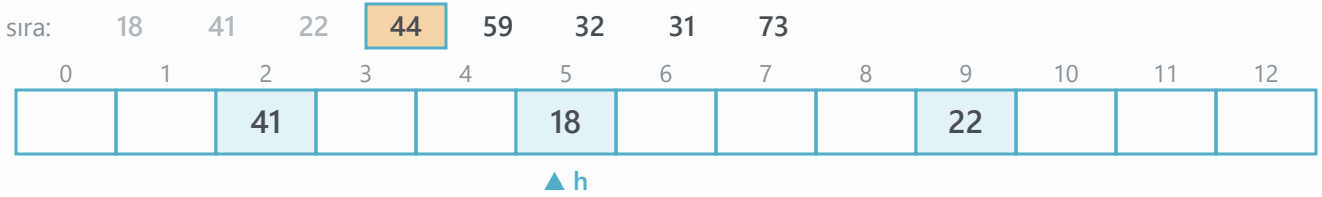
Adım 1 / 19

```
1  // yerlestir(k): çift hash, N = 13
2  int h = k % 13;
3  int d = 7 - k % 7;              // adım: 1…7
4  for (int i = 0; i < 13; i++) {
5    int j = (h + i * d) % 13;
6    if (tablo[j] == null) {       // boş mu?
7      tablo[j] = k;
8      break;
9    }                             // dolu: i++
10 }
```

DEĞİŞKENLER

`N = 13`

Kod İzleme: Çift Hash



h = 44 mod 13 = 5 (ev yuvası), **d = 7 − 2 = 5** (adım; asla 0 olmaz).

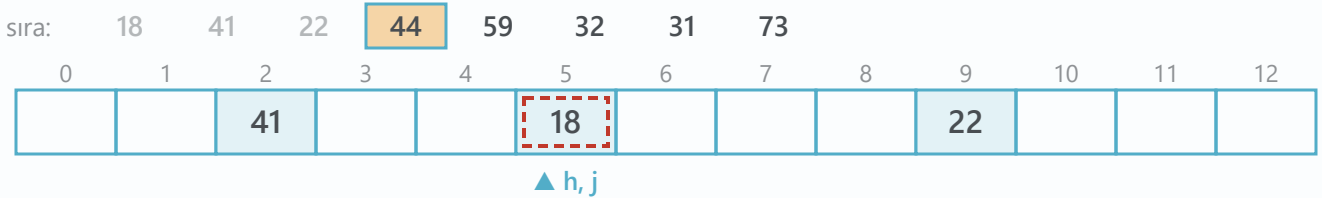
Adım 2 / 19

```
1  // yerlestir(k): çift hash, N = 13
▶ int h = k % 13;
▶ int d = 7 - k % 7;              // adım: 1…7
4  for (int i = 0; i < 13; i++) {
5    int j = (h + i * d) % 13;
6    if (tablo[j] == null) {      // boş mu?
7      tablo[j] = k;
8      break;
9    }                            // dolu: i++
10 }
```

DEĞİŞKENLER

k = 44 | h = 5 | d = 5

Kod İzleme: Çift Hash



i = 0: j = (5 + 0·5) mod 13 = **5** → yuva dolu (18), bir sonraki i.

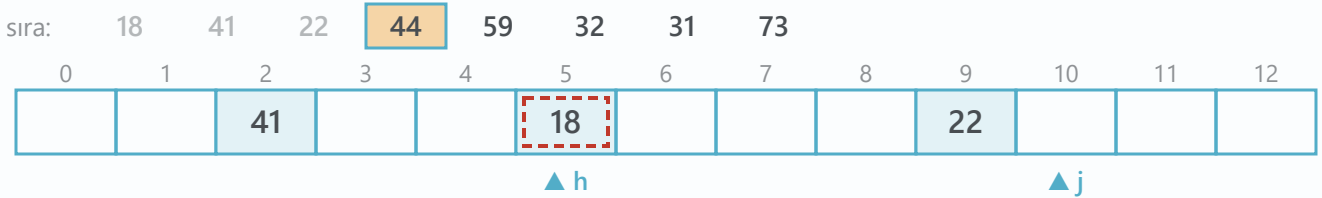
Adım 3 / 19

```
1  // yerlestir(k): çift hash, N = 13
2  int h = k % 13;
3  int d = 7 - k % 7;              // adım: 1…7
4  for (int i = 0; i < 13; i++) {
▶    int j = (h + i * d) % 13;
▶    if (tablo[j] == null) {       // boş mu?
7      tablo[j] = k;
8      break;
9    }                             // dolu: i++
10 }
```

DEĞİŞKENLER

`k = 44` `h = 5` `d = 5` `i = 0` `j = 5` `tablo[j] = 18`

Kod İzleme: Çift Hash



i = 1: j = (5 + 1·5) mod 13 = **10** → yuva **boş**.

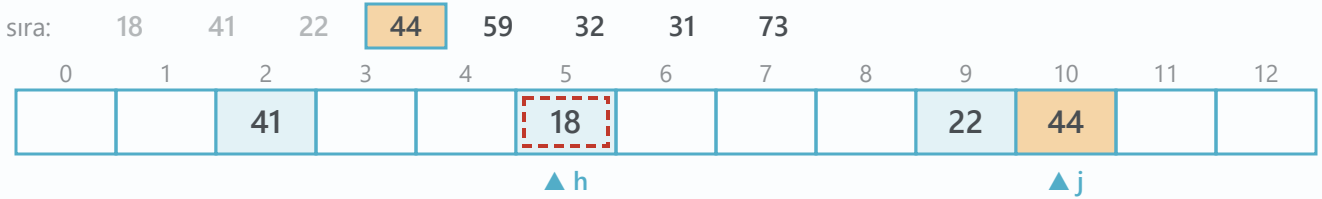
Adım 4 / 19

```
1  // yerlestir(k): çift hash, N = 13
2  int h = k % 13;
3  int d = 7 - k % 7;              // adım: 1…7
4  for (int i = 0; i < 13; i++) {
▶    int j = (h + i * d) % 13;
▶    if (tablo[j] == null) {       // boş mu?
7      tablo[j] = k;
8      break;
9    }                             // dolu: i++
10 }
```

DEĞİŞKENLER

`k = 44` `h = 5` `d = 5` `i = 1` `j = 10` `tablo[j] = null`

Kod İzleme: Çift Hash



`tablo[10] = 44`, `break` (2 yoklama).

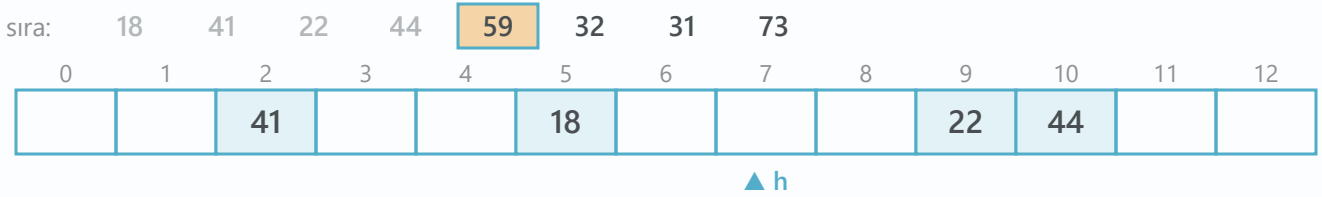
Adım 5 / 19

```
1  // yerlestir(k): çift hash, N = 13
2  int h = k % 13;
3  int d = 7 - k % 7;              // adım: 1…7
4  for (int i = 0; i < 13; i++) {
5    int j = (h + i * d) % 13;
6    if (tablo[j] == null) {       // boş mu?
▶      tablo[j] = k;
▶      break;
9    }                             // dolu: i++
10 }
```

DEĞİŞKENLER

`k = 44` `h = 5` `d = 5` `i = 1` `j = 10` `tablo[j] = 44`

Kod İzleme: Çift Hash



h = 59 mod 13 = 7 (ev yuvası), **d = 7 − 3 = 4** (adım; asla 0 olmaz).

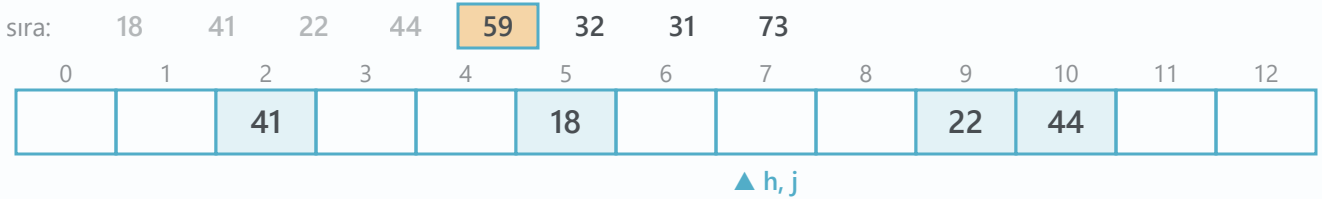
Adım 6 / 19

```
1  // yerlestir(k): çift hash, N = 13
▶ int h = k % 13;
▶ int d = 7 - k % 7;              // adım: 1…7
4  for (int i = 0; i < 13; i++) {
5    int j = (h + i * d) % 13;
6    if (tablo[j] == null) {      // boş mu?
7      tablo[j] = k;
8      break;
9    }                            // dolu: i++
10 }
```

DEĞİŞKENLER

k = 59 | h = 7 | d = 4

Kod İzleme: Çift Hash



i = 0: j = (7 + 0·4) mod 13 = **7** → yuva **boş**.

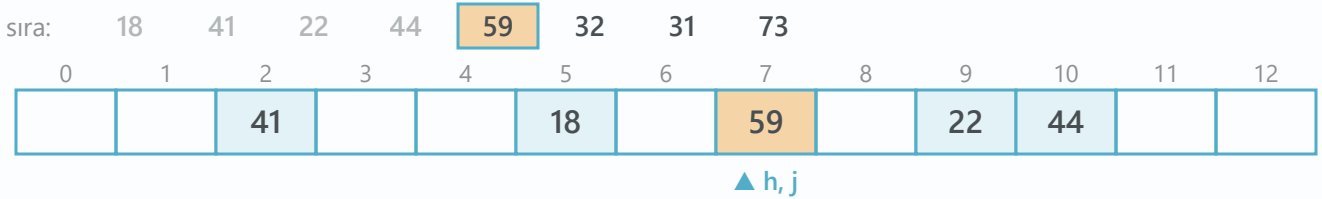
Adım 7 / 19

```
1  // yerlestir(k): çift hash, N = 13
2  int h = k % 13;
3  int d = 7 - k % 7;              // adım: 1…7
4  for (int i = 0; i < 13; i++) {
▶    int j = (h + i * d) % 13;
▶    if (tablo[j] == null) {       // boş mu?
7      tablo[j] = k;
8      break;
9    }                             // dolu: i++
10 }
```

DEĞİŞKENLER

`k = 59` `h = 7` `d = 4` `i = 0` `j = 7` `tablo[j] = null`

Kod İzleme: Çift Hash



`tablo[7] = 59` , `break` (1 yoklama).

Adım 8 / 19

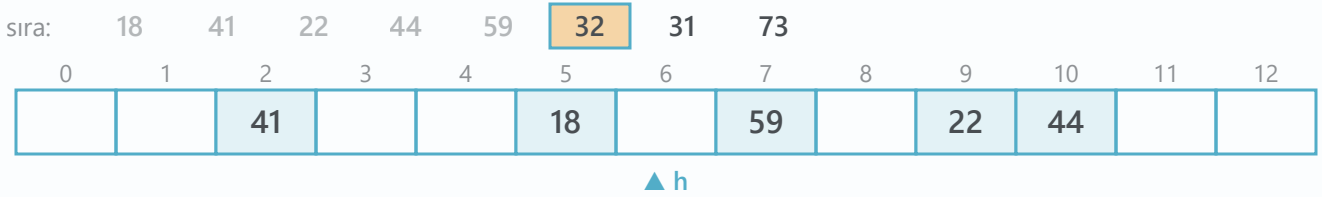
```
1  // yerlestir(k): çift hash, N = 13
2  int h = k % 13;
3  int d = 7 - k % 7;              // adım: 1…7
4  for (int i = 0; i < 13; i++) {
5    int j = (h + i * d) % 13;
6    if (tablo[j] == null) {       // boş mu?
▶      tablo[j] = k;
▶      break;
9    }                             // dolu: i++
10 }
```

DEĞİŞKENLER

`k = 59` `h = 7` `d = 4` `i = 0`

`j = 7` `tablo[j] = 59`

Kod İzleme: Çift Hash



h = 32 mod 13 = 6 (ev yuvası), **d = 7 − 4 = 3** (adım; asla 0 olmaz).

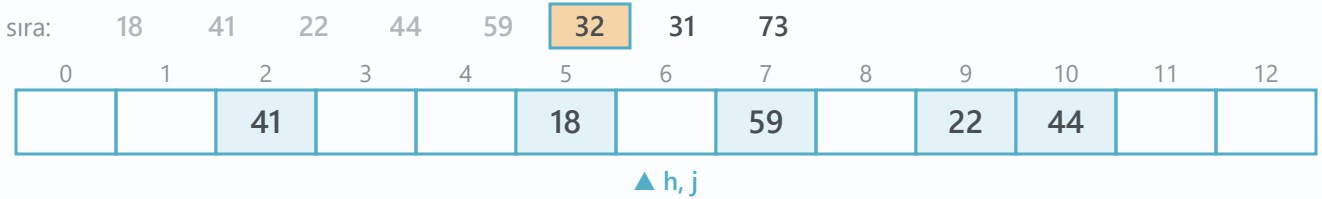
Adım 9 / 19

```
1  // yerlestir(k): çift hash, N = 13
▶ int h = k % 13;
▶ int d = 7 - k % 7;              // adım: 1…7
4  for (int i = 0; i < 13; i++) {
5    int j = (h + i * d) % 13;
6    if (tablo[j] == null) {      // boş mu?
7      tablo[j] = k;
8      break;
9    }                            // dolu: i++
10 }
```

DEĞİŞKENLER

k = 32 | h = 6 | d = 3

Kod İzleme: Çift Hash



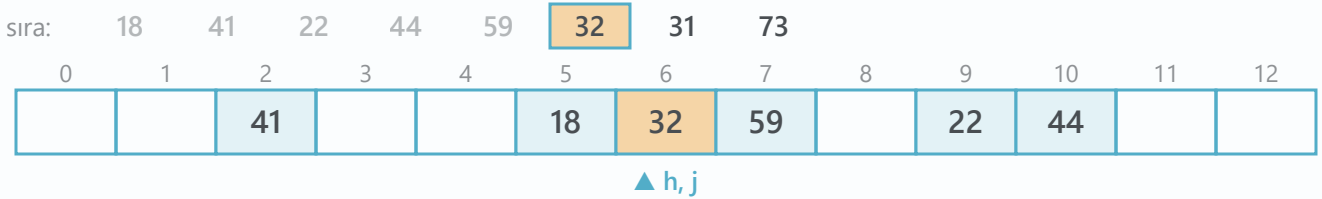
i = 0: j = (6 + 0·3) mod 13 = **6** → yuva **boş**.


```
1  // yerlestir(k): çift hash, N = 13
2  int h = k % 13;
3  int d = 7 - k % 7;              // adım: 1…7
4  for (int i = 0; i < 13; i++) {
▶    int j = (h + i * d) % 13;
▶    if (tablo[j] == null) {       // boş mu?
7      tablo[j] = k;
8      break;
9    }                             // dolu: i++
10 }
```

DEĞİŞKENLER

`k = 32` `h = 6` `d = 3` `i = 0` `j = 6` `tablo[j] = null`

Kod İzleme: Çift Hash



`tablo[6] = 32` , `break` (1 yoklama).

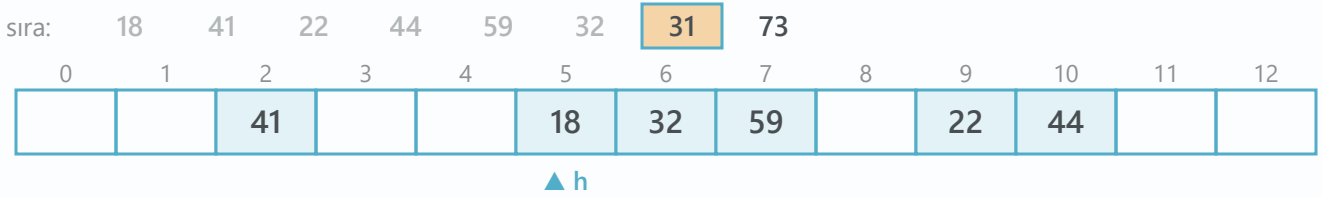
Adım 11 / 19

```
1  // yerlestir(k): çift hash, N = 13
2  int h = k % 13;
3  int d = 7 - k % 7;              // adım: 1…7
4  for (int i = 0; i < 13; i++) {
5    int j = (h + i * d) % 13;
6    if (tablo[j] == null) {       // boş mu?
▶      tablo[j] = k;
▶      break;
9    }                             // dolu: i++
10 }
```

DEĞİŞKENLER

`k = 32` `h = 6` `d = 3` `i = 0` `j = 6` `tablo[j] = 32`

Kod İzleme: Çift Hash



h = 31 mod 13 = 5 (ev yuvası), **d = 7 − 3 = 4** (adım; asla 0 olmaz).

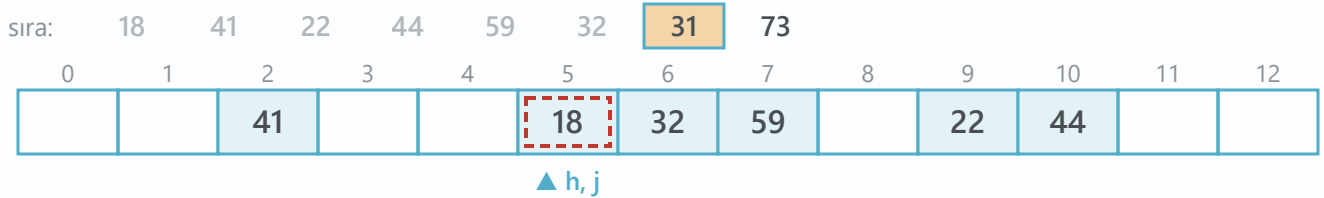
Adım 12 / 19

```
1  // yerlestir(k): çift hash, N = 13
▶ int h = k % 13;
▶ int d = 7 - k % 7;              // adım: 1…7
4  for (int i = 0; i < 13; i++) {
5    int j = (h + i * d) % 13;
6    if (tablo[j] == null) {      // boş mu?
7      tablo[j] = k;
8      break;
9    }                            // dolu: i++
10 }
```

DEĞİŞKENLER

k = 31 | h = 5 | d = 4

Kod İzleme: Çift Hash



i = 0: j = (5 + 0·4) mod 13 = **5** → yuva dolu (18), bir sonraki i.

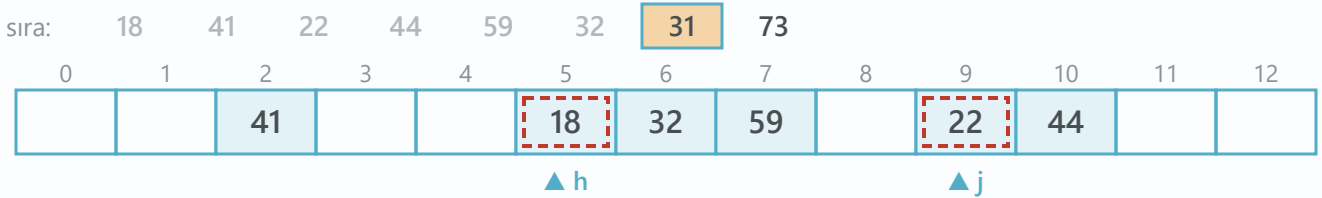
Adım 13 / 19

```
1  // yerlestir(k): çift hash, N = 13
2  int h = k % 13;
3  int d = 7 - k % 7;              // adım: 1…7
4  for (int i = 0; i < 13; i++) {
▶    int j = (h + i * d) % 13;
▶    if (tablo[j] == null) {       // boş mu?
7      tablo[j] = k;
8      break;
9    }                             // dolu: i++
10 }
```

DEĞİŞKENLER

`k = 31` `h = 5` `d = 4` `i = 0` `j = 5` `tablo[j] = 18`

Kod İzleme: Çift Hash



i = 1: j = (5 + 1·4) mod 13 = **9** → yuva dolu (22), bir sonraki i.

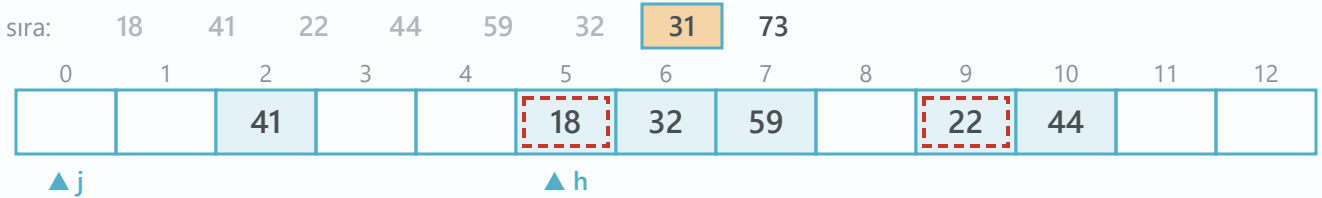
Adım 14 / 19

```
1  // yerlestir(k): çift hash, N = 13
2  int h = k % 13;
3  int d = 7 - k % 7;              // adım: 1…7
4  for (int i = 0; i < 13; i++) {
▶    int j = (h + i * d) % 13;
▶    if (tablo[j] == null) {       // boş mu?
7      tablo[j] = k;
8      break;
9    }                             // dolu: i++
10 }
```

DEĞİŞKENLER

`k = 31` `h = 5` `d = 4` `i = 1` `j = 9` `tablo[j] = 22`

Kod İzleme: Çift Hash



i = 2: j = (5 + 2·4) mod 13 = **0** → yuva **boş**.

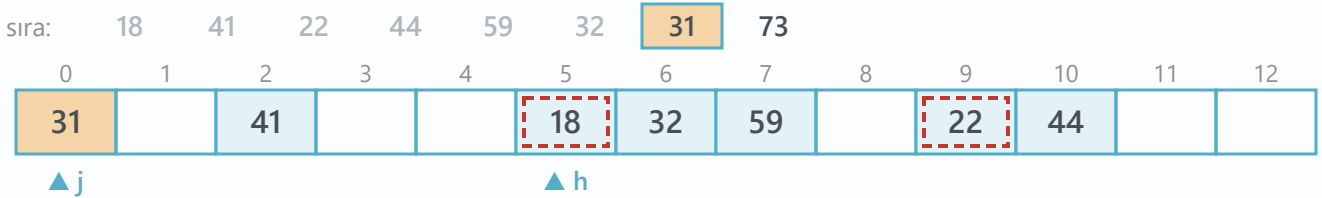
Adım 15 / 19

```
1  // yerlestir(k): çift hash, N = 13
2  int h = k % 13;
3  int d = 7 - k % 7;              // adım: 1…7
4  for (int i = 0; i < 13; i++) {
▶    int j = (h + i * d) % 13;
▶    if (tablo[j] == null) {       // boş mu?
7      tablo[j] = k;
8      break;
9    }                             // dolu: i++
10 }
```

DEĞİŞKENLER

`k = 31` `h = 5` `d = 4` `i = 2` `j = 0` `tablo[j] = null`

Kod İzleme: Çift Hash



`tablo[0] = 31` , `break` (3 yoklama).

Adım 16 / 19

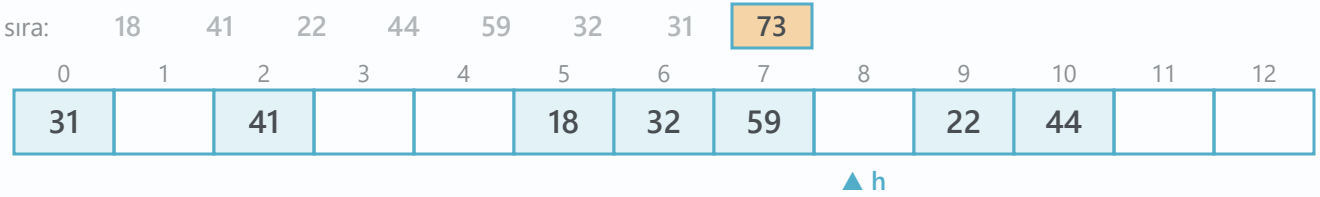
```
1  // yerlestir(k): çift hash, N = 13
2  int h = k % 13;
3  int d = 7 - k % 7;              // adım: 1…7
4  for (int i = 0; i < 13; i++) {
5    int j = (h + i * d) % 13;
6    if (tablo[j] == null) {       // boş mu?
▶      tablo[j] = k;
▶      break;
9    }                             // dolu: i++
10 }
```

DEĞİŞKENLER

`k = 31` `h = 5` `d = 4` `i = 2`

`j = 0` `tablo[j] = 31`

Kod İzleme: Çift Hash



h = 73 mod 13 = 8 (ev yuvası), **d = 7 − 3 = 4** (adım; asla 0 olmaz).

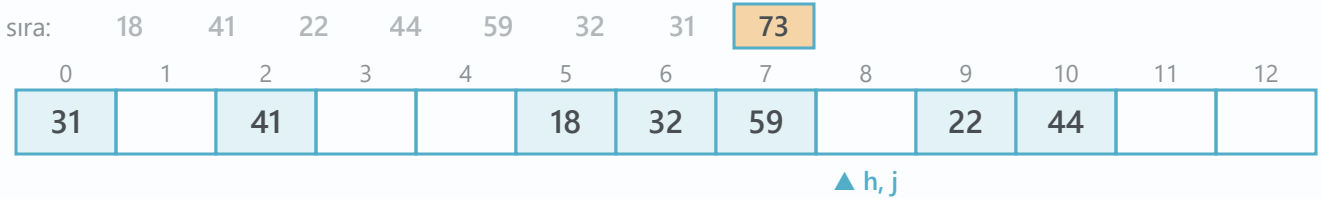
Adım 17 / 19

```
1  // yerlestir(k): çift hash, N = 13
▶ int h = k % 13;
▶ int d = 7 - k % 7;              // adım: 1…7
4  for (int i = 0; i < 13; i++) {
5    int j = (h + i * d) % 13;
6    if (tablo[j] == null) {      // boş mu?
7      tablo[j] = k;
8      break;
9    }                            // dolu: i++
10 }
```

DEĞİŞKENLER

k = 73 | h = 8 | d = 4

Kod İzleme: Çift Hash



i = 0: j = (8 + 0·4) mod 13 = **8** → yuva **boş**.

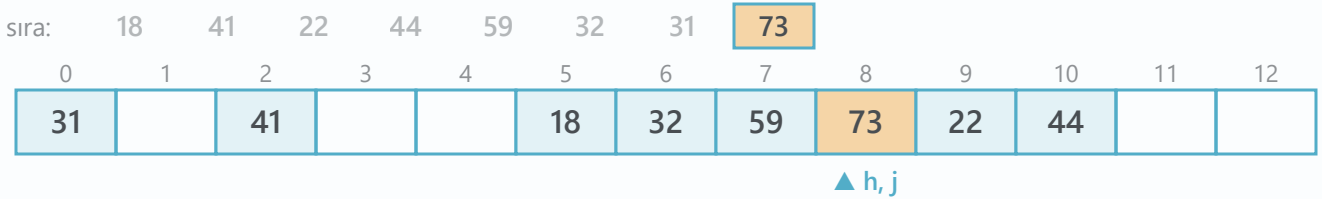
Adım 18 / 19

```
1  // yerlestir(k): çift hash, N = 13
2  int h = k % 13;
3  int d = 7 - k % 7;              // adım: 1…7
4  for (int i = 0; i < 13; i++) {
▶    int j = (h + i * d) % 13;
▶    if (tablo[j] == null) {       // boş mu?
7      tablo[j] = k;
8      break;
9    }                             // dolu: i++
10 }
```

DEĞİŞKENLER

`k = 73` `h = 8` `d = 4` `i = 0` `j = 8` `tablo[j] = null`

Kod İzleme: Çift Hash



`tablo[8] = 73` , `break` (1 yoklama).

Adım 19 / 19

```
1  // yerlestir(k): çift hash, N = 13
2  int h = k % 13;
3  int d = 7 - k % 7;              // adım: 1…7
4  for (int i = 0; i < 13; i++) {
5    int j = (h + i * d) % 13;
6    if (tablo[j] == null) {       // boş mu?
▶      tablo[j] = k;
▶      break;
9    }                             // dolu: i++
10 }
```

DEĞİŞKENLER

`k = 73` `h = 8` `d = 4` `i = 0` `j = 8` `tablo[j] = 73`

Açık Adreslemede Silme: "Silindi" İşareti

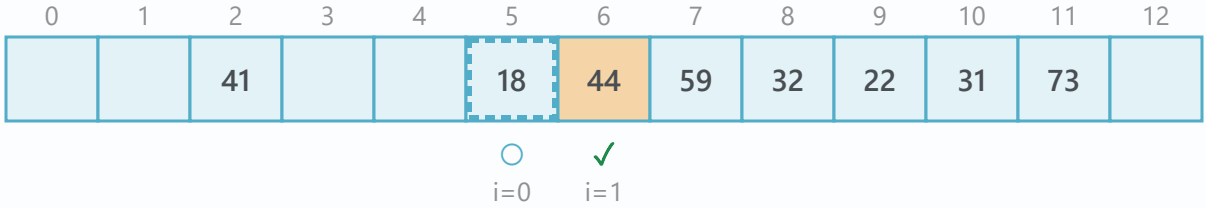


Doğrusal yoklama sonrası tablo. 44 silinmek isteniyor.

Adım 1 / 6

╳ = silindi (DEFUNCT). Arama bu yuvayı dolu gibi atlar; ekleme ise onu boş gibi yeniden kullanabilir. ○ = yoklanan yuva, ✓ = bulundu.

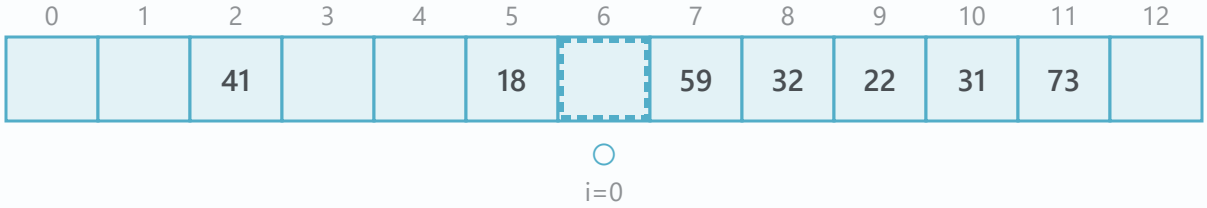
Açık Adreslemede Silme: "Silindi" İşareti



sil(44): h(44) = **5** → 18 ≠ 44 → yuva 6: 44 bulundu. Yuvayı `null` yapsak ne olur?

✗ = silindi (DEFUNCT). Arama bu yuvayı dolu gibi atlar; ekleme ise onu boş gibi yeniden kullanabilir. ○ = yoklanan yuva, ✓ = bulundu.

Açık Adreslemede Silme: "Silindi" İşareti

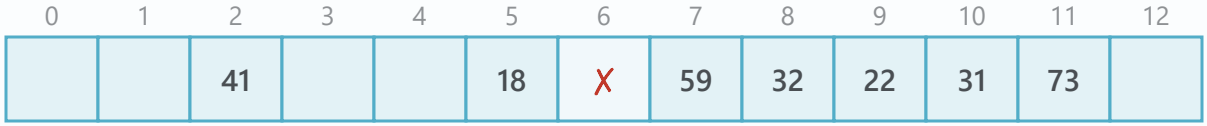


Yanlış yol: yuva 6 boşaltıldı. getir(32): h(32) = **6** → boş yuva → "yok" der. Oysa 32 yuva 8'de!

Adım 3 / 6

✗ = silindi (DEFUNCT). Arama bu yuvayı dolu gibi atlar; ekleme ise onu boş gibi yeniden kullanabilir. ○ = yoklanan yuva, ✓ = bulundu.

Açık Adreslemede Silme: "Silindi" İşareti

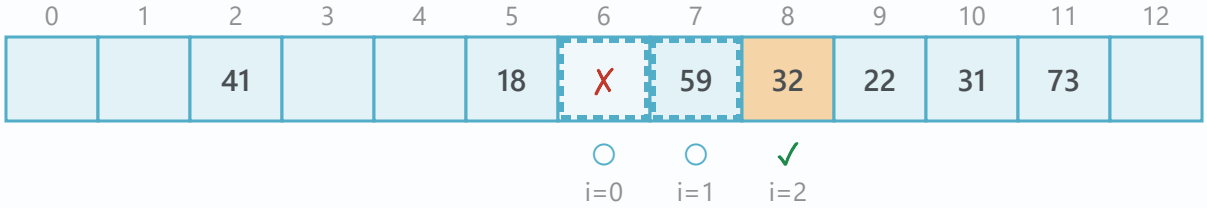


Doğru yol: yuvaya özel bir **silindi** işareti (DEFUNCT, tombstone ✗) konur.

Adım 4 / 6

✗ = silindi (DEFUNCT). Arama bu yuvayı dolu gibi atlar; ekleme ise onu boş gibi yeniden kullanabilir. ○ = yoklanan yuva, ✓ = bulundu.

Açık Adreslemede Silme: "Silindi" İşareti

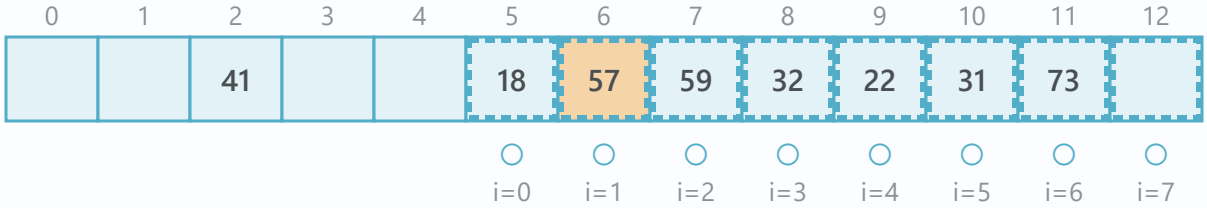


getir(32): h = **6** → X silindi, aramaya devam → 7: 59 ≠ 32 → 8: **32 bulundu**.

Adım 5 / 6

X = silindi (DEFUNCT). Arama bu yuvayı dolu gibi atlar; ekleme ise onu boş gibi yeniden kullanabilir. ○ = yoklanan yuva, ✓ = bulundu.

Açık Adreslemede Silme: "Silindi" İşareti



yerlestir(57): h = **5** → 5, 6(X), 7 … 11 dolu, 12 boş → 57 yok. İlk uygun yuva olan **6**'ya yerleşir.

Adım 6 / 6

X = silindi (DEFUNCT). Arama bu yuvayı dolu gibi atlar; ekleme ise onu boş gibi yeniden kullanabilir. ○ = yoklanan yuva, ✓ = bulundu.

Yuva Bulma: Arama ve Eklemenin Ortak Adımı

```
private int yuvaBul(int k) {       // findSlot
  int h = Math.floorMod(k, N), uygun = -1;
  for (int i = 0; i < N; i++) {
    int j = (h + i) % N;           // doğrusal
    if (tablo[j] == null ||
        tablo[j] == SILINDI) {
      if (uygun == -1) uygun = j;
      if (tablo[j] == null) break;
    } else if (tablo[j].anahtar == k)
      return j;                    // bulundu
  }
  return -(uygun + 1);         // yok: yer kodu
}
```

getir(k)

`j = yuvaBul(k)` ; j ≥ 0 ise `tablo[j].deger` , değilse `null` .

yerlestir(k, v)

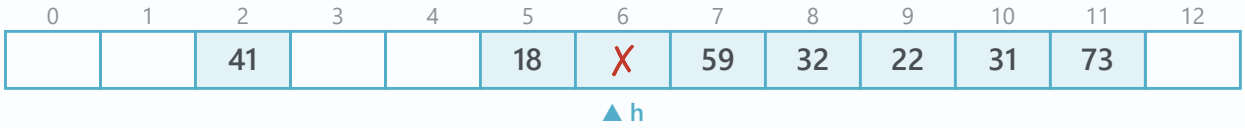
j ≥ 0 ise güncelle; değilse `tablo[-(j+1)]` yuvasına yeni çift.

Diğer yoklamalar

Karesel: `(h + i*i) % N`
Çift hash: `(h + i*h2) % N`

Tablo hiçbir zaman tamamen dolmaz (λ < 1 tutulur, gerekince büyütülür); bu yüzden `uygun` her zaman bulunur.

Kod İzleme: `yuvaBul(32)`, 44 silinmiş (✗)



`yuvaBul(32)` : h = 32 mod 13 = **6**; henüz uygun yuva yok (−1).

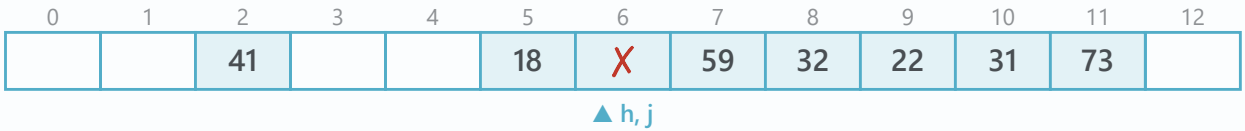
Adım 1 / 7

```
1  private int yuvaBul(int k) {        // findSlot
▶    int h = Math.floorMod(k, N), uygun = -1;
3    for (int i = 0; i < N; i++) {
4      int j = (h + i) % N;            // doğrusal
5      if (tablo[j] == null ||
6          tablo[j] == SILINDI) {
7        if (uygun == -1) uygun = j;
8        if (tablo[j] == null) break;
9      } else if (tablo[j].anahtar == k)
10       return j;                     // bulundu
11   }
12   return -(uygun + 1);          // yok: yer kodu
13 }
```

DEĞİŞKENLER

`k = 32` `h = 6` `uygun = −1`

Kod İzleme: `yuvaBul(32)`, 44 silinmiş (X)



i = 0: j = **6** → yuva **SILINDI** (X).

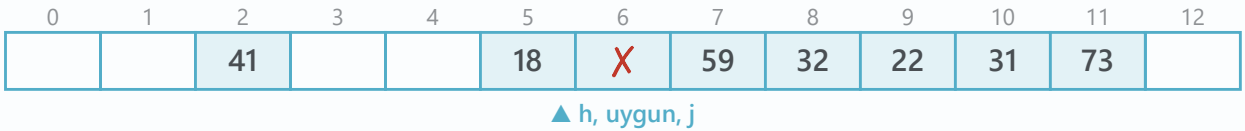
Adım 2 / 7

```
1  private int yuvaBul(int k) {        // findSlot
2    int h = Math.floorMod(k, N), uygun = -1;
3    for (int i = 0; i < N; i++) {
▶      int j = (h + i) % N;            // doğrusal
▶      if (tablo[j] == null ||
▶          tablo[j] == SILINDI) {
7        if (uygun == -1) uygun = j;
8        if (tablo[j] == null) break;
9      } else if (tablo[j].anahtar == k)
10       return j;                     // bulundu
11   }
12   return -(uygun + 1);          // yok: yer kodu
13 }
```

DEĞİŞKENLER

`k = 32` `h = 6` `uygun = -1` `i = 0` `j = 6` `tablo[j] = SILINDI`

Kod İzleme: `yuvaBul(32)`, 44 silinmiş (✗)



İlk boş/silindi yuva: **uygun = 6** (ekleme gerekirse buraya).

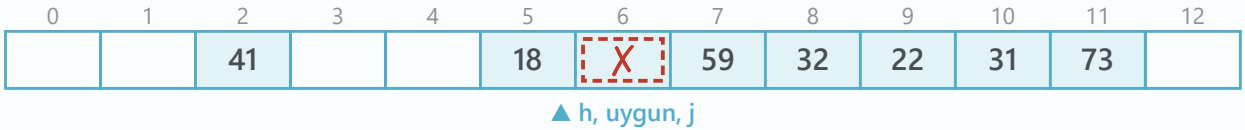
Adım 3 / 7

```
1  private int yuvaBul(int k) {        // findSlot
2    int h = Math.floorMod(k, N), uygun = -1;
3    for (int i = 0; i < N; i++) {
4      int j = (h + i) % N;            // doğrusal
5      if (tablo[j] == null ||
6          tablo[j] == SILINDI) {
▶        if (uygun == -1) uygun = j;
8        if (tablo[j] == null) break;
9      } else if (tablo[j].anahtar == k)
10       return j;                     // bulundu
11   }
12   return -(uygun + 1);          // yok: yer kodu
13 }
```

DEĞİŞKENLER

`k = 32` `h = 6` `uygun = 6` `i = 0` `j = 6` `tablo[j] = SILINDI`

Kod İzleme: `yuvaBul(32)`, 44 silinmiş (X)



Silindi ≠ boş: **aramaya devam** (ötesinde 32 olabilir).

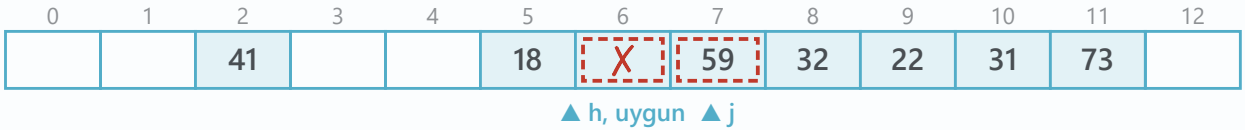
Adım 4 / 7

```
1  private int yuvaBul(int k) {        // findSlot
2    int h = Math.floorMod(k, N), uygun = -1;
3    for (int i = 0; i < N; i++) {
4      int j = (h + i) % N;            // doğrusal
5      if (tablo[j] == null ||
6          tablo[j] == SILINDI) {
7        if (uygun == -1) uygun = j;
▶        if (tablo[j] == null) break;
9      } else if (tablo[j].anahtar == k)
10       return j;                     // bulundu
11   }
12   return -(uygun + 1);          // yok: yer kodu
13 }
```

DEĞİŞKENLER

`k = 32` `h = 6` `uygun = 6` `i = 0` `j = 6` `tablo[j] = SILINDI`

Kod İzleme: `yuvaBul(32)`, 44 silinmiş (X)



i = 1: j = **7** → dolu (59 ≠ 32), sonraki yuva.

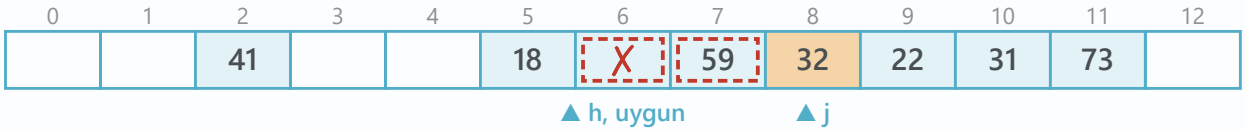
Adım 5 / 7

```
1  private int yuvaBul(int k) {        // findSlot
2    int h = Math.floorMod(k, N), uygun = -1;
3    for (int i = 0; i < N; i++) {
▶      int j = (h + i) % N;            // doğrusal
▶      if (tablo[j] == null ||
6          tablo[j] == SILINDI) {
7        if (uygun == -1) uygun = j;
8        if (tablo[j] == null) break;
▶      } else if (tablo[j].anahtar == k)
10       return j;                     // bulundu
11   }
12   return -(uygun + 1);          // yok: yer kodu
13 }
```

DEĞİŞKENLER

`k = 32` `h = 6` `uygun = 6` `i = 1` `j = 7` `tablo[j] = 59`

Kod İzleme: `yuvaBul(32)`, 44 silinmiş (X)



i = 2: j = **8** → dolu, tablo[8].anahtar = 32 == 32.

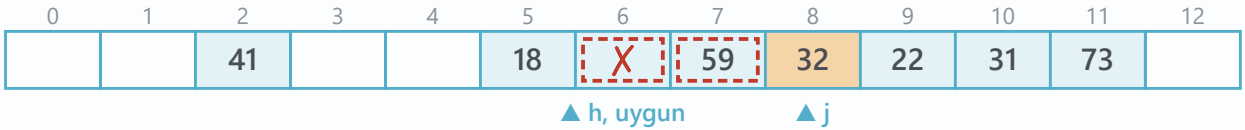
Adım 6 / 7

```
1  private int yuvaBul(int k) {        // findSlot
2    int h = Math.floorMod(k, N), uygun = -1;
3    for (int i = 0; i < N; i++) {
▶      int j = (h + i) % N;            // doğrusal
▶      if (tablo[j] == null ||
6          tablo[j] == SILINDI) {
7        if (uygun == -1) uygun = j;
8        if (tablo[j] == null) break;
▶      } else if (tablo[j].anahtar == k)
10       return j;                     // bulundu
11   }
12   return -(uygun + 1);          // yok: yer kodu
13 }
```

DEĞİŞKENLER

`k = 32` `h = 6` `uygun = 6` `i = 2` `j = 8` `tablo[j] = 32`

Kod İzleme: `yuvaBul(32)`, 44 silinmiş (X)



`return 8` (≥ 0 = bulundu). `getir` tablo[8].deger döndürür; X arama zincirini koparmadı.

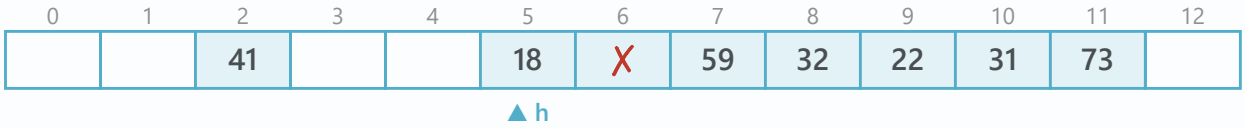
Adım 7 / 7

```
1  private int yuvaBul(int k) {        // findSlot
2    int h = Math.floorMod(k, N), uygun = -1;
3    for (int i = 0; i < N; i++) {
4      int j = (h + i) % N;            // doğrusal
5      if (tablo[j] == null ||
6          tablo[j] == SILINDI) {
7        if (uygun == -1) uygun = j;
8        if (tablo[j] == null) break;
9      } else if (tablo[j].anahtar == k)
▶        return j;                     // bulundu
11   }
12   return -(uygun + 1);          // yok: yer kodu
13 }
```

DEĞİŞKENLER

`k = 32` `h = 6` `uygun = 6` `i = 2` `j = 8` `tablo[j] = 32`

Kod İzleme: `yuvaBul(57)` ve ekleme



`yuvaBul(57)`: h = 57 mod 13 = **5**; henüz uygun yuva yok (−1).

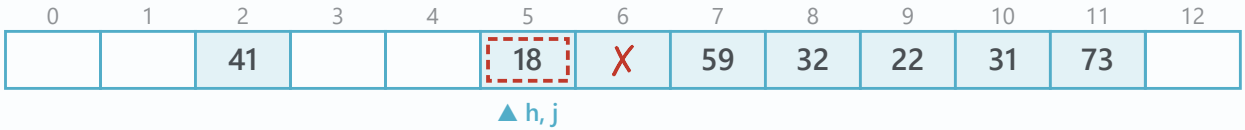
Adım 1 / 15

```
1  private int yuvaBul(int k) {        // findSlot
▶    int h = Math.floorMod(k, N), uygun = -1;
3    for (int i = 0; i < N; i++) {
4      int j = (h + i) % N;            // doğrusal
5      if (tablo[j] == null ||
6          tablo[j] == SILINDI) {
7        if (uygun == -1) uygun = j;
8        if (tablo[j] == null) break;
9      } else if (tablo[j].anahtar == k)
10       return j;                     // bulundu
11   }
12   return -(uygun + 1);          // yok: yer kodu
13 }
```

DEĞİŞKENLER

`k = 57` `h = 5` `uygun = −1`

Kod İzleme: `yuvaBul(57)` ve ekleme



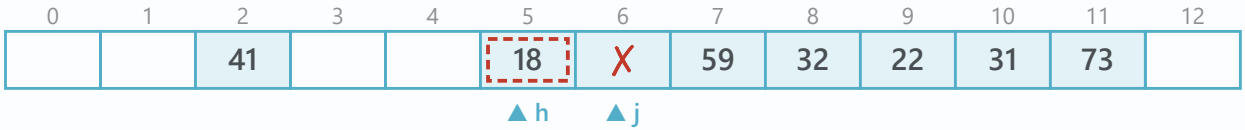
i = 0: j = **5** → dolu (18 ≠ 57), sonraki yuva.


```
1  private int yuvaBul(int k) {        // findSlot
2    int h = Math.floorMod(k, N), uygun = -1;
3    for (int i = 0; i < N; i++) {
▶      int j = (h + i) % N;            // doğrusal
▶      if (tablo[j] == null ||
6          tablo[j] == SILINDI) {
7        if (uygun == -1) uygun = j;
8        if (tablo[j] == null) break;
▶      } else if (tablo[j].anahtar == k)
10       return j;                     // bulundu
11   }
12   return -(uygun + 1);          // yok: yer kodu
13 }
```

DEĞİŞKENLER

`k = 57` `h = 5` `uygun = -1` `i = 0` `j = 5` `tablo[j] = 18`

Kod İzleme: `yuvaBul(57)` ve ekleme



i = 1: j = **6** → yuva **SILINDI** (*X*).

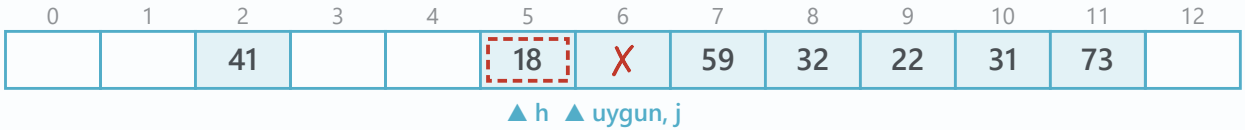
Adım 3 / 15

```
1  private int yuvaBul(int k) {        // findSlot
2    int h = Math.floorMod(k, N), uygun = -1;
3    for (int i = 0; i < N; i++) {
▶      int j = (h + i) % N;            // doğrusal
▶      if (tablo[j] == null ||
▶          tablo[j] == SILINDI) {
7        if (uygun == -1) uygun = j;
8        if (tablo[j] == null) break;
9      } else if (tablo[j].anahtar == k)
10       return j;                     // bulundu
11   }
12   return -(uygun + 1);          // yok: yer kodu
13 }
```

DEĞİŞKENLER

`k = 57` `h = 5` `uygun = -1` `i = 1` `j = 6` `tablo[j] = SILINDI`

Kod İzleme: `yuvaBul(57)` ve ekleme



İlk boş/silindi yuva: **uygun = 6** (ekleme gerekirse buraya).

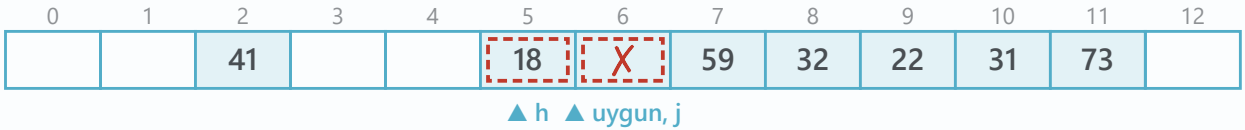
Adım 4 / 15

```
1  private int yuvaBul(int k) {        // findSlot
2    int h = Math.floorMod(k, N), uygun = -1;
3    for (int i = 0; i < N; i++) {
4      int j = (h + i) % N;            // doğrusal
5      if (tablo[j] == null ||
6          tablo[j] == SILINDI) {
▶        if (uygun == -1) uygun = j;
8        if (tablo[j] == null) break;
9      } else if (tablo[j].anahtar == k)
10       return j;                     // bulundu
11   }
12   return -(uygun + 1);          // yok: yer kodu
13 }
```

DEĞİŞKENLER

`k = 57` `h = 5` `uygun = 6` `i = 1` `j = 6` `tablo[j] = SILINDI`

Kod İzleme: `yuvaBul(57)` ve ekleme



Silindi ≠ boş: **aramaya devam** (ötesinde 57 olabilir).

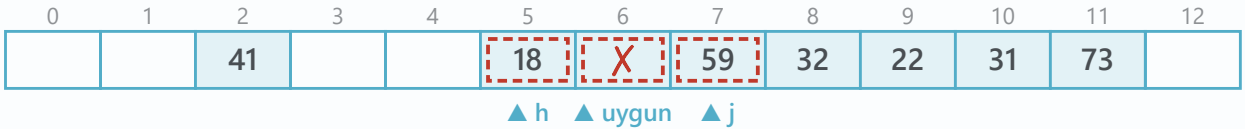
Adım 5 / 15

```
1  private int yuvaBul(int k) {        // findSlot
2    int h = Math.floorMod(k, N), uygun = -1;
3    for (int i = 0; i < N; i++) {
4      int j = (h + i) % N;            // doğrusal
5      if (tablo[j] == null ||
6          tablo[j] == SILINDI) {
7        if (uygun == -1) uygun = j;
▶        if (tablo[j] == null) break;
9      } else if (tablo[j].anahtar == k)
10       return j;                     // bulundu
11   }
12   return -(uygun + 1);          // yok: yer kodu
13 }
```

DEĞİŞKENLER

`k = 57` `h = 5` `uygun = 6` `i = 1` `j = 6` `tablo[j] = SILINDI`

Kod İzleme: `yuvaBul(57)` ve ekleme



i = 2: j = **7** → dolu (59 ≠ 57), sonraki yuva.

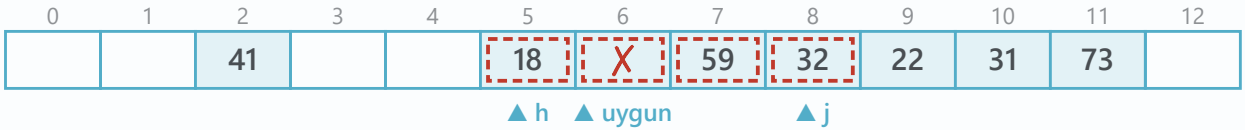
Adım 6 / 15

```
1  private int yuvaBul(int k) {        // findSlot
2    int h = Math.floorMod(k, N), uygun = -1;
3    for (int i = 0; i < N; i++) {
▶      int j = (h + i) % N;            // doğrusal
▶      if (tablo[j] == null ||
6          tablo[j] == SILINDI) {
7        if (uygun == -1) uygun = j;
8        if (tablo[j] == null) break;
▶      } else if (tablo[j].anahtar == k)
10       return j;                     // bulundu
11   }
12   return -(uygun + 1);          // yok: yer kodu
13 }
```

DEĞİŞKENLER

`k = 57` `h = 5` `uygun = 6` `i = 2` `j = 7` `tablo[j] = 59`

Kod İzleme: `yuvaBul(57)` ve ekleme



i = 3: j = **8** → dolu (32 ≠ 57), sonraki yuva.

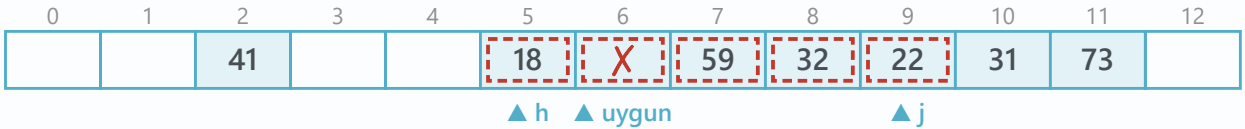
Adım 7 / 15

```
1  private int yuvaBul(int k) {        // findSlot
2    int h = Math.floorMod(k, N), uygun = -1;
3    for (int i = 0; i < N; i++) {
▶      int j = (h + i) % N;            // doğrusal
▶      if (tablo[j] == null ||
6          tablo[j] == SILINDI) {
7        if (uygun == -1) uygun = j;
8        if (tablo[j] == null) break;
▶      } else if (tablo[j].anahtar == k)
10       return j;                     // bulundu
11   }
12   return -(uygun + 1);          // yok: yer kodu
13 }
```

DEĞİŞKENLER

`k = 57` `h = 5` `uygun = 6` `i = 3` `j = 8` `tablo[j] = 32`

Kod İzleme: `yuvaBul(57)` ve ekleme



i = 4: j = **9** → dolu (22 ≠ 57), sonraki yuva.

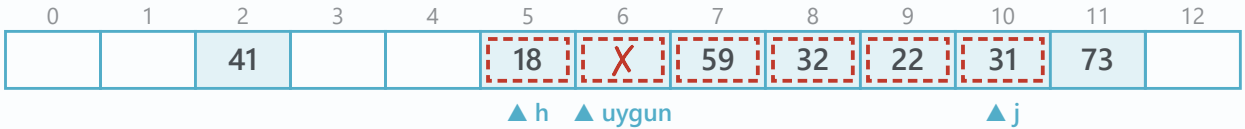
Adım 8 / 15

```
1  private int yuvaBul(int k) {        // findSlot
2    int h = Math.floorMod(k, N), uygun = -1;
3    for (int i = 0; i < N; i++) {
▶      int j = (h + i) % N;            // doğrusal
▶      if (tablo[j] == null ||
6          tablo[j] == SILINDI) {
7        if (uygun == -1) uygun = j;
8        if (tablo[j] == null) break;
▶      } else if (tablo[j].anahtar == k)
10       return j;                     // bulundu
11   }
12   return -(uygun + 1);          // yok: yer kodu
13 }
```

DEĞİŞKENLER

`k = 57` `h = 5` `uygun = 6` `i = 4` `j = 9` `tablo[j] = 22`

Kod İzleme: `yuvaBul(57)` ve ekleme



i = 5: j = **10** → dolu (31 ≠ 57), sonraki yuva.

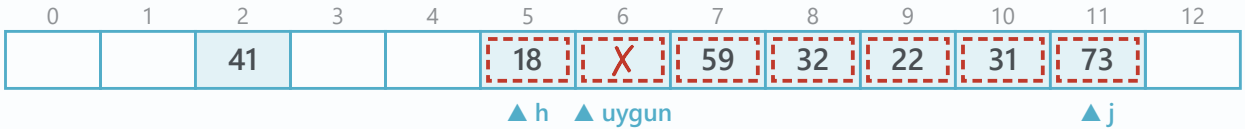
Adım 9 / 15

```
1  private int yuvaBul(int k) {        // findSlot
2    int h = Math.floorMod(k, N), uygun = -1;
3    for (int i = 0; i < N; i++) {
▶      int j = (h + i) % N;            // doğrusal
▶      if (tablo[j] == null ||
6          tablo[j] == SILINDI) {
7        if (uygun == -1) uygun = j;
8        if (tablo[j] == null) break;
▶      } else if (tablo[j].anahtar == k)
10       return j;                     // bulundu
11   }
12   return -(uygun + 1);          // yok: yer kodu
13 }
```

DEĞİŞKENLER

`k = 57` `h = 5` `uygun = 6` `i = 5` `j = 10` `tablo[j] = 31`

Kod İzleme: `yuvaBul(57)` ve ekleme



i = 6: j = **11** → dolu (73 ≠ 57), sonraki yuva.

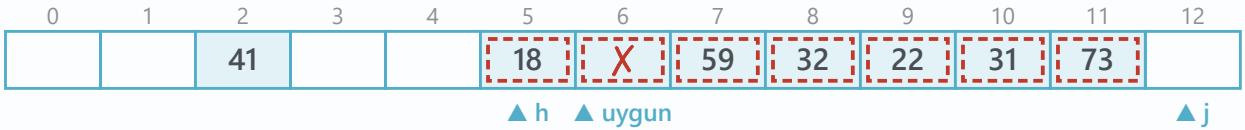
Adım 10 / 15

```
1  private int yuvaBul(int k) {        // findSlot
2    int h = Math.floorMod(k, N), uygun = -1;
3    for (int i = 0; i < N; i++) {
▶      int j = (h + i) % N;            // doğrusal
▶      if (tablo[j] == null ||
6          tablo[j] == SILINDI) {
7        if (uygun == -1) uygun = j;
8        if (tablo[j] == null) break;
▶      } else if (tablo[j].anahtar == k)
10       return j;                     // bulundu
11   }
12   return -(uygun + 1);          // yok: yer kodu
13 }
```

DEĞİŞKENLER

`k = 57` `h = 5` `uygun = 6` `i = 6` `j = 11` `tablo[j] = 73`

Kod İzleme: `yuvaBul(57)` ve ekleme



i = 7: j = **12** → yuva boş.

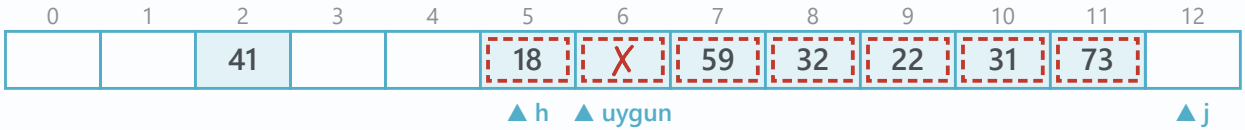
Adım 11 / 15

```
1  private int yuvaBul(int k) {        // findSlot
2    int h = Math.floorMod(k, N), uygun = -1;
3    for (int i = 0; i < N; i++) {
▶      int j = (h + i) % N;            // doğrusal
▶      if (tablo[j] == null ||
▶          tablo[j] == SILINDI) {
7        if (uygun == -1) uygun = j;
8        if (tablo[j] == null) break;
9      } else if (tablo[j].anahtar == k)
10       return j;                     // bulundu
11   }
12   return -(uygun + 1);          // yok: yer kodu
13 }
```

DEĞİŞKENLER

`k = 57` `h = 5` `uygun = 6` `i = 7` `j = 12` `tablo[j] = null`

Kod İzleme: `yuvaBul(57)` ve ekleme



uygun zaten 6: değişmez (ilk uygun yuva korunur).

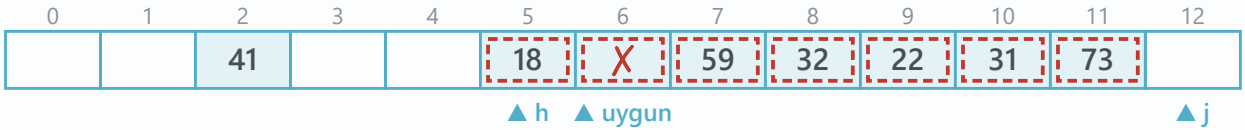
Adım 12 / 15

```
1  private int yuvaBul(int k) {        // findSlot
2    int h = Math.floorMod(k, N), uygun = -1;
3    for (int i = 0; i < N; i++) {
4      int j = (h + i) % N;            // doğrusal
5      if (tablo[j] == null ||
6          tablo[j] == SILINDI) {
▶        if (uygun == -1) uygun = j;
8        if (tablo[j] == null) break;
9      } else if (tablo[j].anahtar == k)
10       return j;                     // bulundu
11   }
12   return -(uygun + 1);          // yok: yer kodu
13 }
```

DEĞİŞKENLER

`k = 57` `h = 5` `uygun = 6` `i = 7` `j = 12` `tablo[j] = null`

Kod İzleme: `yuvaBul(57)` ve ekleme



Gerçekten **boş** yuva → 57 bundan ötede olamaz: `break`.

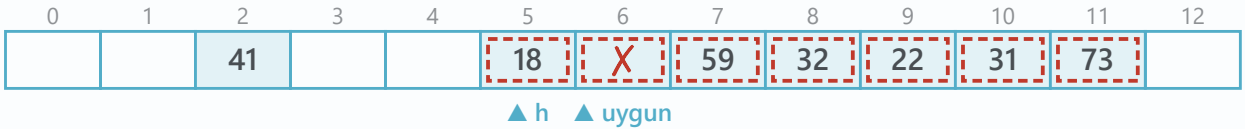
Adım 13 / 15

```
1  private int yuvaBul(int k) {        // findSlot
2    int h = Math.floorMod(k, N), uygun = -1;
3    for (int i = 0; i < N; i++) {
4      int j = (h + i) % N;            // doğrusal
5      if (tablo[j] == null ||
6          tablo[j] == SILINDI) {
7        if (uygun == -1) uygun = j;
▶        if (tablo[j] == null) break;
9      } else if (tablo[j].anahtar == k)
10       return j;                     // bulundu
11   }
12   return -(uygun + 1);          // yok: yer kodu
13 }
```

DEĞİŞKENLER

`k = 57` `h = 5` `uygun = 6` `i = 7` `j = 12` `tablo[j] = null`

Kod İzleme: `yuvaBul(57)` ve ekleme



57 yok: `return -(6 + 1) = -7` (negatif = "yok", içinde eklenecek yer).

Adım 14 / 15

```
1  private int yuvaBul(int k) {        // findSlot
2    int h = Math.floorMod(k, N), uygun = -1;
3    for (int i = 0; i < N; i++) {
4      int j = (h + i) % N;            // doğrusal
5      if (tablo[j] == null ||
6          tablo[j] == SILINDI) {
7        if (uygun == -1) uygun = j;
8        if (tablo[j] == null) break;
9      } else if (tablo[j].anahtar == k)
10       return j;                     // bulundu
11   }
▶    return -(uygun + 1);          // yok: yer kodu
13 }
```

DEĞİŞKENLER

`k = 57` `h = 5` `uygun = 6`

Kod İzleme: `yuvaBul(57)` ve ekleme



`yerlestir` : j = -7 < 0 → `tablo[-(j + 1)] = tablo[6] = 57` . Silindi yuvası yeniden kullanıldı.

Adım 15 / 15

```
1  private int yuvaBul(int k) {        // findSlot
2    int h = Math.floorMod(k, N), uygun = -1;
3    for (int i = 0; i < N; i++) {
4      int j = (h + i) % N;            // doğrusal
5      if (tablo[j] == null ||
6          tablo[j] == SILINDI) {
7        if (uygun == -1) uygun = j;
8        if (tablo[j] == null) break;
9      } else if (tablo[j].anahtar == k)
10       return j;                     // bulundu
11   }
▶    return -(uygun + 1);          // yok: yer kodu
13 }
```

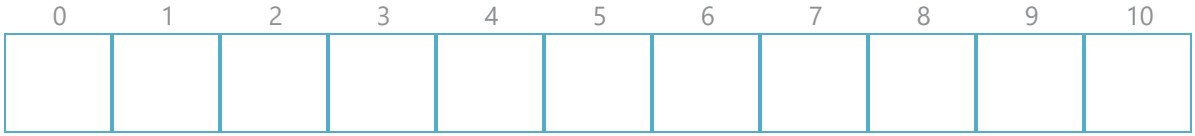
DEĞİŞKENLER

`k = 57` `h = 5` `uygun = 6`

Deneyin: Açık Adresleme (N = 11)

Yöntem doğrusal: h + i Anahtar 22

ekle ara sil örnek doldur sıfırla

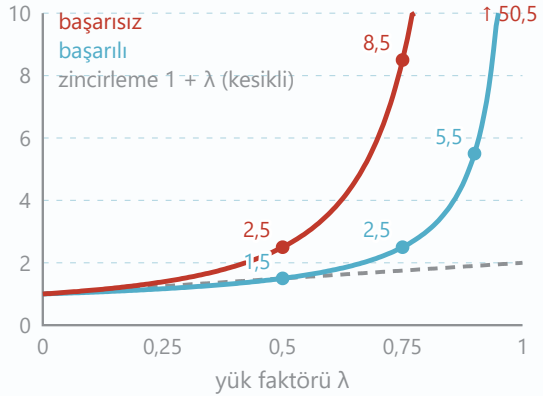


Bir anahtar girip **ekle**'ye basın ya da **örnek doldur** ile başlayın.

h(k) = k mod 11. Kesikli çerçeve: yoklanan dolu yuva · turuncu: sonuç yuvası · X: silindi (DEFUNCT). Enter tuşu = ekle.

Yük Faktörü (Load Factor)

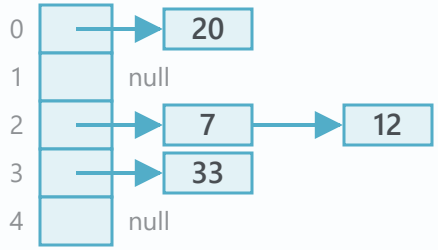
- **λ = n / N** (n: çift sayısı = buyukluk, N: kova sayısı).
- Zincirlemede λ, ortalama zincir uzunluğudur; λ > 1 olabilir ama işlemler yavaşlar.
- Açık adreslemede λ ≤ 1 zorunludur; λ → 1 iken yoklama sayısı patlar.
- Pratik eşikler: zincirlemede λ ≈ 0,75 (Java `HashMap`), açık adreslemede λ ≤ 0,5.
- Eşik aşılınca tablo yaklaşık 2 katına büyütülür.



Doğrusal yoklamada beklenen yoklama sayısı (Knuth):
başarılı $\tfrac{1}{2}(1 + 1/(1-\lambda))$ · başarısız $\tfrac{1}{2}(1 + 1/(1-\lambda)^2)$

Zincirlemede beklenen maliyet **O(1 + λ)**: λ sabit tutulursa (tablo n ile birlikte büyütülürse) ekleme, arama ve silme **ortalama O(1)** olur.

Yeniden Boyutlandırma (Rehashing)



N = 5, h(k) = k mod 5

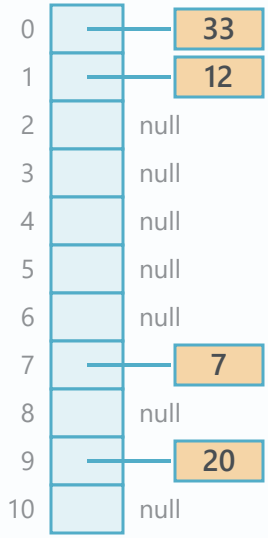
12 → 2, 7 → 2, 20 → 0, 33 → 3

λ = 4 / 5 = 0,80 > 0,75

→ tabloyu büyüt

N = 5 kovalı tabloya 12, 7, 20, 33 eklendi. λ = 0,80 ile 0,75 eşiği aşıldı → yeniden boyutlandırma (resizing).

Yeniden Boyutlandırma (Rehashing)



Yeni N = 11 (2·5'ten büyük ilk asal)

20 mod 11 = 9

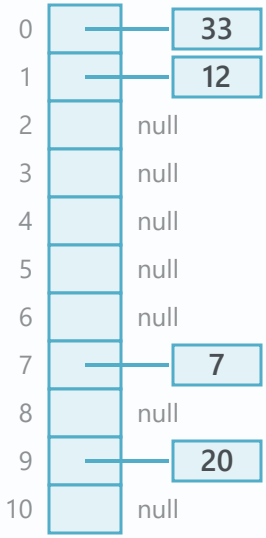
7 mod 11 = 7

12 mod 11 = 1

33 mod 11 = 0

Yeni dizi ayrılır ve **her anahtar yeniden hash'lenir** (rehashing): indisler değişir, eski konumlar geçersizdir.

Yeniden Boyutlandırma (Rehashing)



λ = 4 / 11 ≈ 0,36

çakışma kalmadı

maliyet: O(n) bir kez

amortize: ekleme başına O(1)

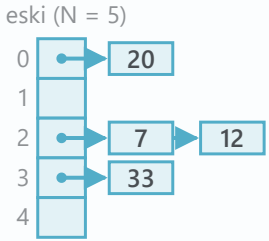
Büyütme tek seferde O(n) sürer ve pahalıdır; ama kapasite her seferinde ~2 katına çıktığından ekleme başına **amortize O(1)**.

Kod İzleme: `yenidenBoyutlandir()`

```
1  private void yenidenBoyutlandir() {
▶    HashDugumu[] eski = kovalar;
3    int yeniN = 2 * eski.length + 1;
4    kovalar = new HashDugumu[yeniN];
5    buyukluk = 0;  kovaSayisi = yeniN;
6    for (HashDugumu ilk : eski) {   // her kova
7      while (ilk != null) {
8        yerlestir(ilk.anahtar, ilk.deger);
9        ilk = ilk.sonraki;
10     }
11   }
12 }
```

DEĞİŞKENLER

`eski` = **`kovalar(5)`**



λ = 4/5 = 0,8 > 0,75 olduğu için çağrıldı.

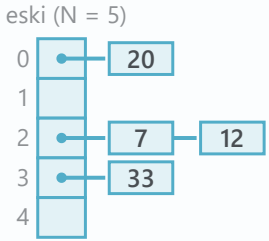
`eski` eski diziyi tutar.

Kod İzleme: `yenidenBoyutlandir()`

```
1  private void yenidenBoyutlandir() {
2    HashDugumu[] eski = kovalar;
▶    int yeniN = 2 * eski.length + 1;
4    kovalar = new HashDugumu[yeniN];
5    buyukluk = 0;  kovaSayisi = yeniN;
6    for (HashDugumu ilk : eski) {   // her kova
7      while (ilk != null) {
8        yerlestir(ilk.anahtar, ilk.deger);
9        ilk = ilk.sonraki;
10     }
11   }
12 }
```

DEĞİŞKENLER

`eski = kovalar(5)` `yeniN = 11`



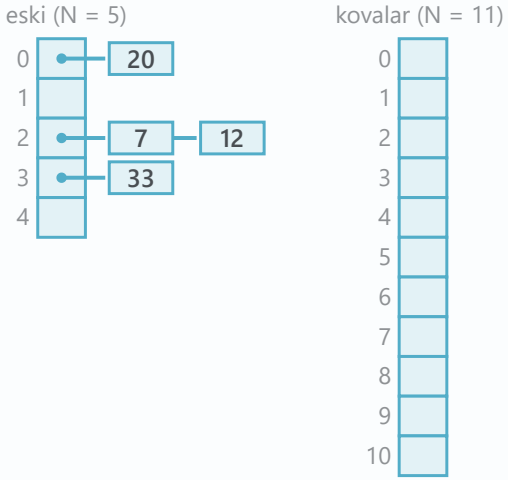
yeniN = 2·5 + 1 = 11: yaklaşık iki kat ve tek sayı (burada asal).

Kod İzleme: `yenidenBoyutlandir()`

```
1  private void yenidenBoyutlandir() {
2    HashDugumu[] eski = kovalar;
3    int yeniN = 2 * eski.length + 1;
▶    kovalar = new HashDugumu[yeniN];
5    buyukluk = 0;  kovaSayisi = yeniN;
6    for (HashDugumu ilk : eski) {   // her kova
7      while (ilk != null) {
8        yerlestir(ilk.anahtar, ilk.deger);
9        ilk = ilk.sonraki;
10     }
11   }
12 }
```

DEĞİŞKENLER

`eski = kovalar(5)` `yeniN = 11`



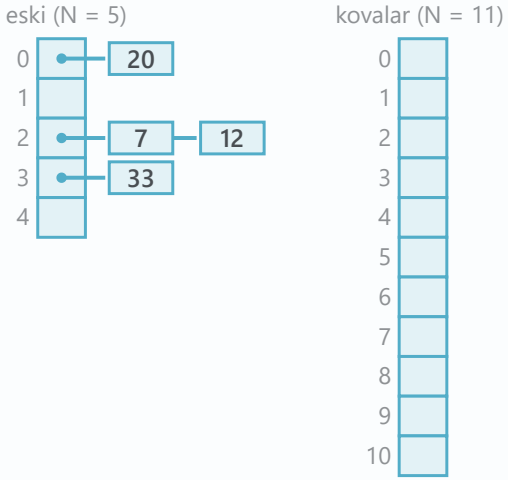
`kovalar` artık 11 boş kovalı yeni diziyi gösterir.

Kod İzleme: `yenidenBoyutlandir()`

```
1  private void yenidenBoyutlandir() {
2    HashDugumu[] eski = kovalar;
3    int yeniN = 2 * eski.length + 1;
4    kovalar = new HashDugumu[yeniN];
▶    buyukluk = 0;  kovaSayisi = yeniN;
6    for (HashDugumu ilk : eski) {   // her kova
7      while (ilk != null) {
8        yerlestir(ilk.anahtar, ilk.deger);
9        ilk = ilk.sonraki;
10     }
11   }
12 }
```

DEĞİŞKENLER

`eski = kovalar(5)` `yeniN = 11` `buyukluk = 0`



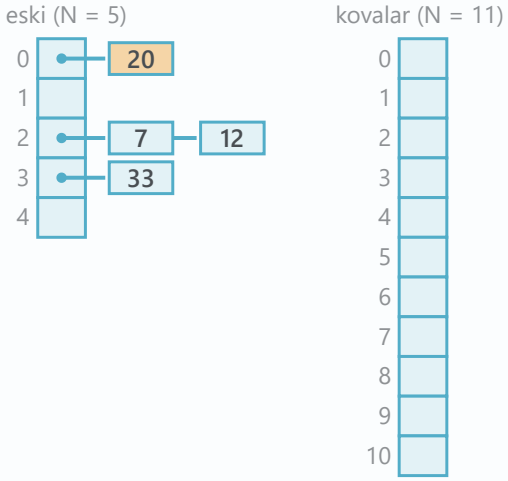
buyukluk = 0: `yerlestir` her çifti yeniden sayacak.

Kod İzleme: `yenidenBoyutlandir()`

```
1  private void yenidenBoyutlandir() {
2    HashDugumu[] eski = kovalar;
3    int yeniN = 2 * eski.length + 1;
4    kovalar = new HashDugumu[yeniN];
5    buyukluk = 0;  kovaSayisi = yeniN;
6    for (HashDugumu ilk : eski) {   // her kova
7      while (ilk != null) {
8        yerlestir(ilk.anahtar, ilk.deger);
9        ilk = ilk.sonraki;
10     }
11   }
12 }
```

DEĞİŞKENLER

`eski = kovalar(5)` `yeniN = 11` `buyukluk = 0` `ilk = düğüm(20)`



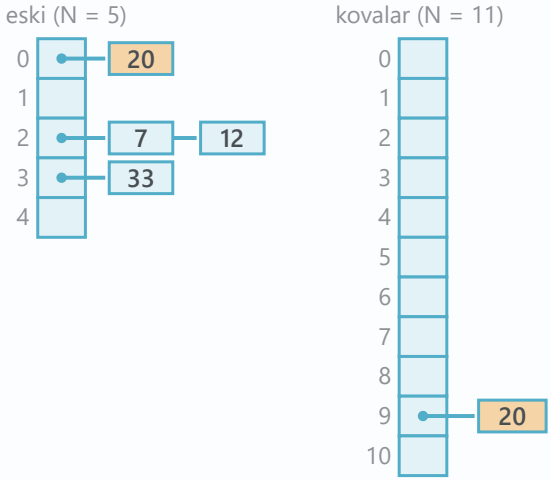
Eski kova 0: `ilk` = düğüm 20.

Kod İzleme: `yenidenBoyutlandir()`

```
1  private void yenidenBoyutlandir() {
2    HashDugumu[] eski = kovalar;
3    int yeniN = 2 * eski.length + 1;
4    kovalar = new HashDugumu[yeniN];
5    buyukluk = 0;  kovaSayisi = yeniN;
6    for (HashDugumu ilk : eski) {   // her kova
7      while (ilk != null) {
▶        yerlestir(ilk.anahtar, ilk.deger);
9        ilk = ilk.sonraki;
10     }
11   }
12 }
```

DEĞİŞKENLER

`eski = kovalar(5)` `yeniN = 11` `buyukluk = 1` `ilk = düğüm(20)`



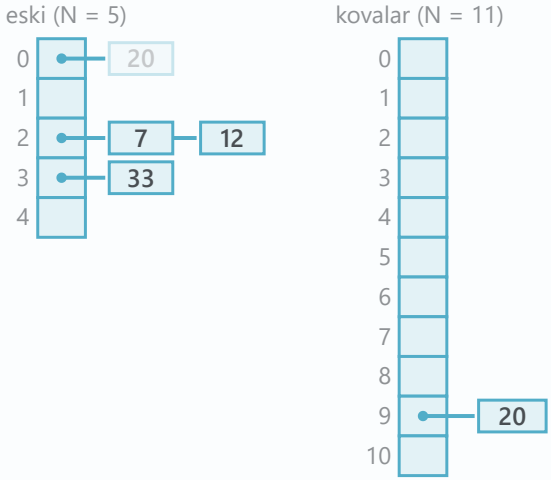
`yerlestir(20)`: eskiden 20 mod 5 = 0, şimdi **20 mod 11 = 9** → yeni kova 9.

Kod İzleme: `yenidenBoyutlandir()`

```
1  private void yenidenBoyutlandir() {
2    HashDugumu[] eski = kovalar;
3    int yeniN = 2 * eski.length + 1;
4    kovalar = new HashDugumu[yeniN];
5    buyukluk = 0;  kovaSayisi = yeniN;
6    for (HashDugumu ilk : eski) {   // her kova
7      while (ilk != null) {
8        yerlestir(ilk.anahtar, ilk.deger);
▶        ilk = ilk.sonraki;
10     }
11   }
12 }
```

DEĞİŞKENLER

`eski = kovalar(5)` `yeniN = 11` `buyukluk = 1` `ilk = null`



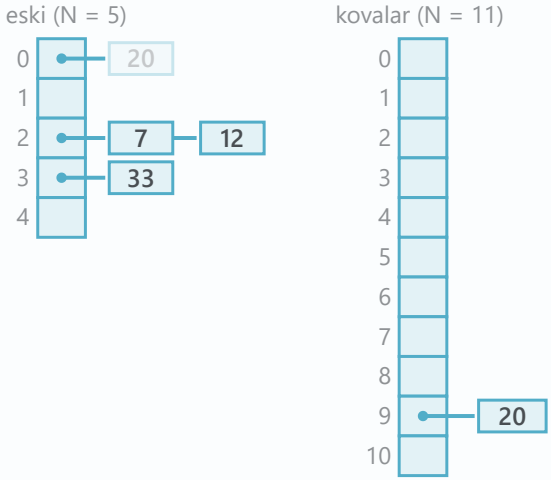
`ilk = null` : bu zincir bitti.

Kod İzleme: `yenidenBoyutlandir()`

```
1  private void yenidenBoyutlandir() {
2    HashDugumu[] eski = kovalar;
3    int yeniN = 2 * eski.length + 1;
4    kovalar = new HashDugumu[yeniN];
5    buyukluk = 0;  kovaSayisi = yeniN;
▶    for (HashDugumu ilk : eski) {   // her kova
▶      while (ilk != null) {
8        yerlestir(ilk.anahtar, ilk.deger);
9        ilk = ilk.sonraki;
10     }
11   }
12 }
```

DEĞİŞKENLER

`eski = kovalar(5)` `yeniN = 11` `buyukluk = 1` `ilk = null`



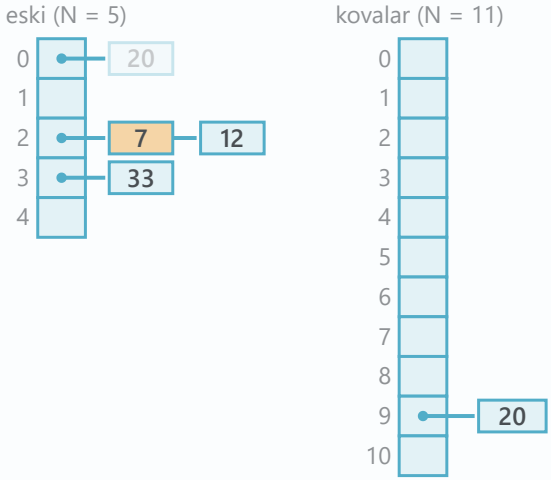
Eski kova 1 boş: `ilk = null`, while'a girilmez.

Kod İzleme: `yenidenBoyutlandir()`

```
1  private void yenidenBoyutlandir() {
2    HashDugumu[] eski = kovalar;
3    int yeniN = 2 * eski.length + 1;
4    kovalar = new HashDugumu[yeniN];
5    buyukluk = 0;  kovaSayisi = yeniN;
▶   for (HashDugumu ilk : eski) {   // her kova
7      while (ilk != null) {
8        yerlestir(ilk.anahtar, ilk.deger);
9        ilk = ilk.sonraki;
10     }
11   }
12 }
```

DEĞİŞKENLER

`eski = kovalar(5)` `yeniN = 11` `buyukluk = 1` `ilk = düğüm(7)`



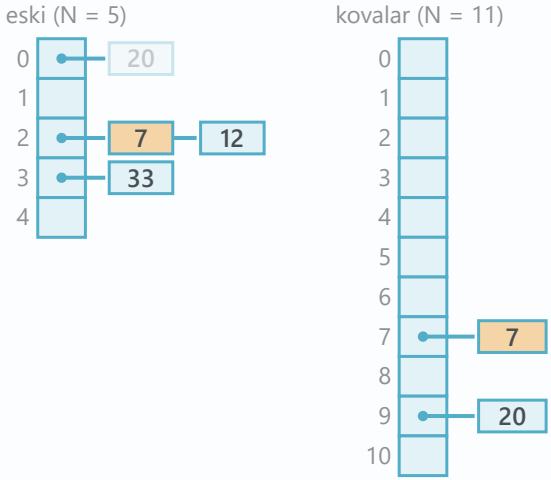
Eski kova 2: `ilk` = düğüm 7.

Kod İzleme: `yenidenBoyutlandir()`

```
1  private void yenidenBoyutlandir() {
2    HashDugumu[] eski = kovalar;
3    int yeniN = 2 * eski.length + 1;
4    kovalar = new HashDugumu[yeniN];
5    buyukluk = 0;  kovaSayisi = yeniN;
6    for (HashDugumu ilk : eski) {   // her kova
7      while (ilk != null) {
▶        yerlestir(ilk.anahtar, ilk.deger);
9        ilk = ilk.sonraki;
10     }
11   }
12 }
```

DEĞİŞKENLER

`eski = kovalar(5)` `yeniN = 11` `buyukluk = 2` `ilk = düğüm(7)`



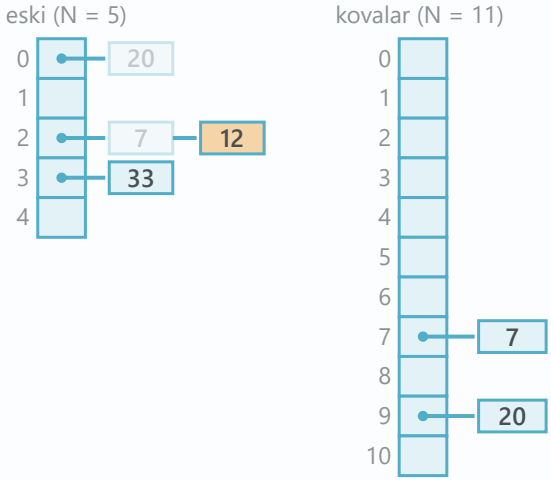
`yerlestir(7)` : eskiden 7 mod 5 = 2, şimdi **7 mod 11 = 7** → yeni kova 7.

Kod İzleme: `yenidenBoyutlandir()`

```
1  private void yenidenBoyutlandir() {
2    HashDugumu[] eski = kovalar;
3    int yeniN = 2 * eski.length + 1;
4    kovalar = new HashDugumu[yeniN];
5    buyukluk = 0;  kovaSayisi = yeniN;
6    for (HashDugumu ilk : eski) {   // her kova
7      while (ilk != null) {
8        yerlestir(ilk.anahtar, ilk.deger);
▶        ilk = ilk.sonraki;
10     }
11   }
12 }
```

DEĞİŞKENLER

`eski = kovalar(5)` `yeniN = 11` `buyukluk = 2` `ilk = düğüm(12)`



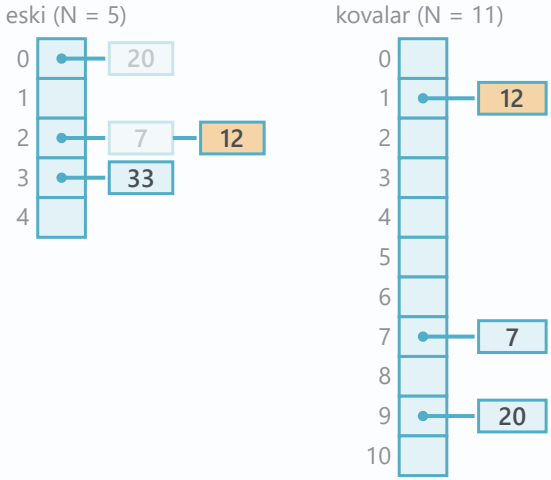
`ilk = ilk.sonraki` → düğüm 12 (aynı eski zincir).

Kod İzleme: `yenidenBoyutlandir()`

```
1  private void yenidenBoyutlandir() {
2    HashDugumu[] eski = kovalar;
3    int yeniN = 2 * eski.length + 1;
4    kovalar = new HashDugumu[yeniN];
5    buyukluk = 0;  kovaSayisi = yeniN;
6    for (HashDugumu ilk : eski) {   // her kova
7      while (ilk != null) {
▶        yerlestir(ilk.anahtar, ilk.deger);
9        ilk = ilk.sonraki;
10     }
11   }
12 }
```

DEĞİŞKENLER

`eski` = **kovalar(5)** | `yeniN` = **11** | `buyukluk` = **3** | `ilk` = **düğüm(12)**



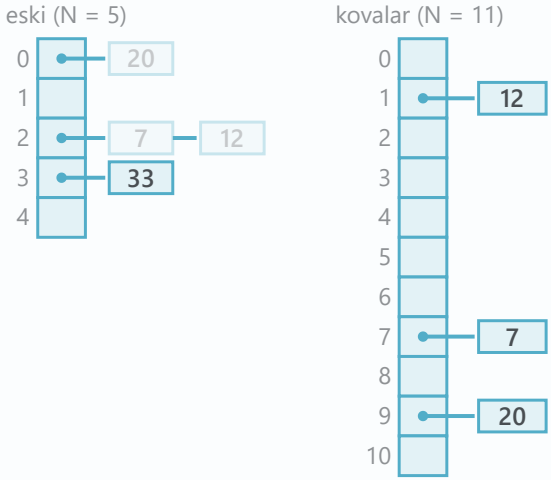
`yerlestir(12)` : eskiden 12 mod 5 = 2, şimdi **12 mod 11 = 1** → yeni kova 1.

Kod İzleme: `yenidenBoyutlandir()`

```
1  private void yenidenBoyutlandir() {
2    HashDugumu[] eski = kovalar;
3    int yeniN = 2 * eski.length + 1;
4    kovalar = new HashDugumu[yeniN];
5    buyukluk = 0;  kovaSayisi = yeniN;
6    for (HashDugumu ilk : eski) {   // her kova
7      while (ilk != null) {
8        yerlestir(ilk.anahtar, ilk.deger);
▶        ilk = ilk.sonraki;
10     }
11   }
12 }
```

DEĞİŞKENLER

`eski = kovalar(5)` `yeniN = 11` `buyukluk = 3` `ilk = null`



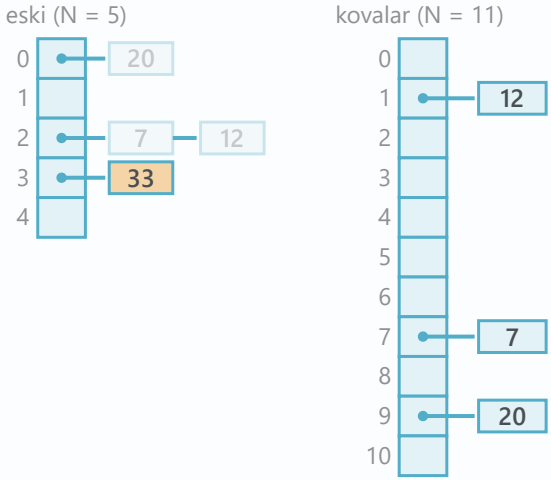
`ilk = null` : bu zincir bitti.

Kod İzleme: `yenidenBoyutlandir()`

```
1  private void yenidenBoyutlandir() {
2    HashDugumu[] eski = kovalar;
3    int yeniN = 2 * eski.length + 1;
4    kovalar = new HashDugumu[yeniN];
5    buyukluk = 0;  kovaSayisi = yeniN;
▶   for (HashDugumu ilk : eski) {   // her kova
7      while (ilk != null) {
8        yerlestir(ilk.anahtar, ilk.deger);
9        ilk = ilk.sonraki;
10     }
11   }
12 }
```

DEĞİŞKENLER

`eski = kovalar(5)` `yeniN = 11` `buyukluk = 3` `ilk = düğüm(33)`



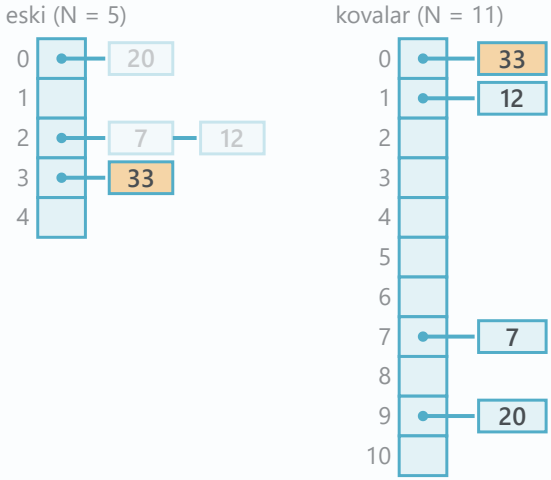
Eski kova 3: `ilk` = düğüm 33.

Kod İzleme: `yenidenBoyutlandir()`

```
1  private void yenidenBoyutlandir() {
2    HashDugumu[] eski = kovalar;
3    int yeniN = 2 * eski.length + 1;
4    kovalar = new HashDugumu[yeniN];
5    buyukluk = 0;  kovaSayisi = yeniN;
6    for (HashDugumu ilk : eski) {   // her kova
7      while (ilk != null) {
▶        yerlestir(ilk.anahtar, ilk.deger);
9        ilk = ilk.sonraki;
10     }
11   }
12 }
```

DEĞİŞKENLER

`eski = kovalar(5)` `yeniN = 11` `buyukluk = 4` `ilk = düğüm(33)`



`yerlestir(33)` : eskiden 33 mod 5 = 3, şimdi **33 mod 11 = 0** → yeni kova 0.

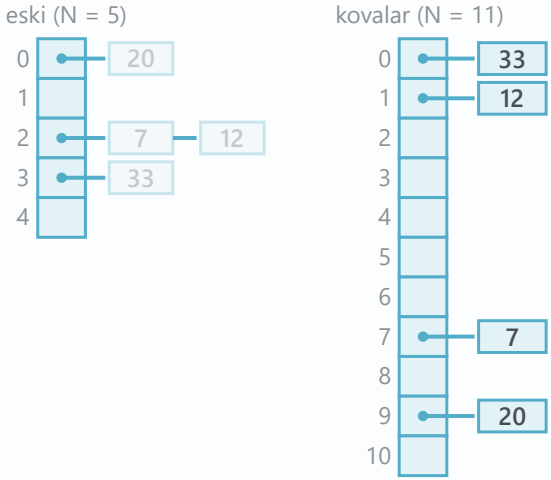
Kod İzleme: `yenidenBoyutlandir()`

```
1  private void yenidenBoyutlandir() {
2    HashDugumu[] eski = kovalar;
3    int yeniN = 2 * eski.length + 1;
4    kovalar = new HashDugumu[yeniN];
5    buyukluk = 0;  kovaSayisi = yeniN;
6    for (HashDugumu ilk : eski) {   // her kova
7      while (ilk != null) {
8        yerlestir(ilk.anahtar, ilk.deger);
▶        ilk = ilk.sonraki;
10     }
11   }
12 }
```

DEĞİŞKENLER

`eski = kovalar(5)` `yeniN = 11` `buyukluk = 4`

`ilk = null`



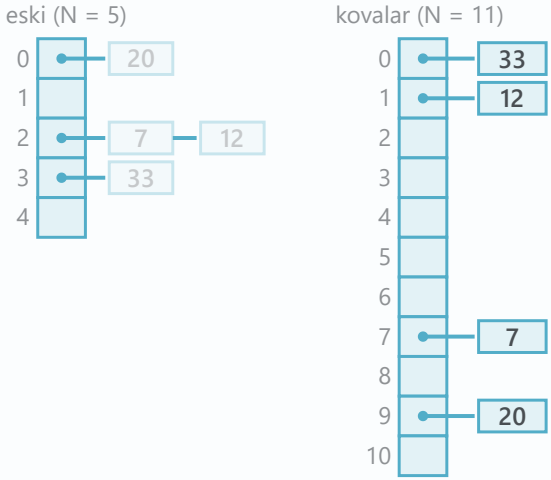
`ilk = null` : bu zincir bitti.

Kod İzleme: `yenidenBoyutlandir()`

```
1  private void yenidenBoyutlandir() {
2    HashDugumu[] eski = kovalar;
3    int yeniN = 2 * eski.length + 1;
4    kovalar = new HashDugumu[yeniN];
5    buyukluk = 0;  kovaSayisi = yeniN;
▶    for (HashDugumu ilk : eski) {   // her kova
▶      while (ilk != null) {
8        yerlestir(ilk.anahtar, ilk.deger);
9        ilk = ilk.sonraki;
10     }
11   }
12 }
```

DEĞİŞKENLER

`eski = kovalar(5)` `yeniN = 11` `buyukluk = 4` `ilk = null`



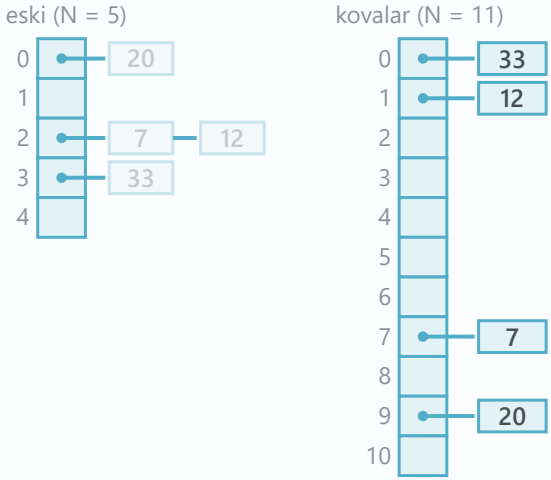
Eski kova 4 boş: `ilk = null`, while'a girilmez.

Kod İzleme: `yenidenBoyutlandir()`

```
1  private void yenidenBoyutlandir() {
2    HashDugumu[] eski = kovalar;
3    int yeniN = 2 * eski.length + 1;
4    kovalar = new HashDugumu[yeniN];
5    buyukluk = 0;  kovaSayisi = yeniN;
6    for (HashDugumu ilk : eski) {   // her kova
7      while (ilk != null) {
8        yerlestir(ilk.anahtar, ilk.deger);
9        ilk = ilk.sonraki;
10     }
11   }
▶ }
```

DEĞİŞKENLER

`yeniN = 11` `buyukluk = 4`



Bitti: 7 ile 12 artık ayrı kovalarda. λ = 4/11 ≈ 0,36. Tüm çiftler taşındığı için **O(n)**.

Karmaşıklık Karşılaştırması

Eşleme gerçeklemesi	getir	yerlestir	sil
Sırasız dizi / liste	O(n)	O(n)*	O(n)
Sıralı dizi (ikili arama)	O(log n)	O(n)	O(n)
Hash tablosu: ortalama	**O(1)**	**O(1)**	**O(1)**
Hash tablosu: en kötü	O(n)	O(n)	O(n)

* Anahtarın zaten var olup olmadığını kontrol etmek için tüm liste taranır.

En kötü durum

Tüm anahtarlar aynı kovaya düşerse tek bir uzun zincir (ya da küme) oluşur: O(n).

Alan

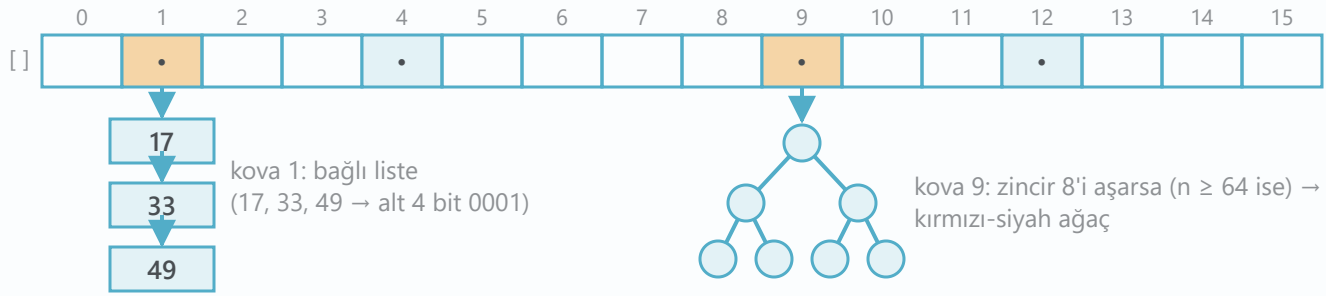
O(n + N): kova dizisi + çiftler. Zincirlemede düğüm başına bir referans fazlası.

Sıra yok

En küçük anahtar, sıralı gezinme gibi işlemler O(n): sıralı erişim için arama ağaçları.

PERFORMANS VE UYGULAMALAR

Java `HashMap` İçeride Nasıl Çalışır?



`indis = (n - 1) & (h ^ (h >>> 16)), n = 16 = 2⁴`

12 öğeyi aşınca (0,75·16) → n = 32

Ayrı zincirleme, n = 2^k

Kova sayısı hep 2'nin kuvveti (varsayılan 16); mod yerine hızlı `& (n - 1)` maskesi.

Bit karıştırma

Maske yalnız alt bitleri görür; `h ^ (h >>> 16)` üst bitleri de indise katar.

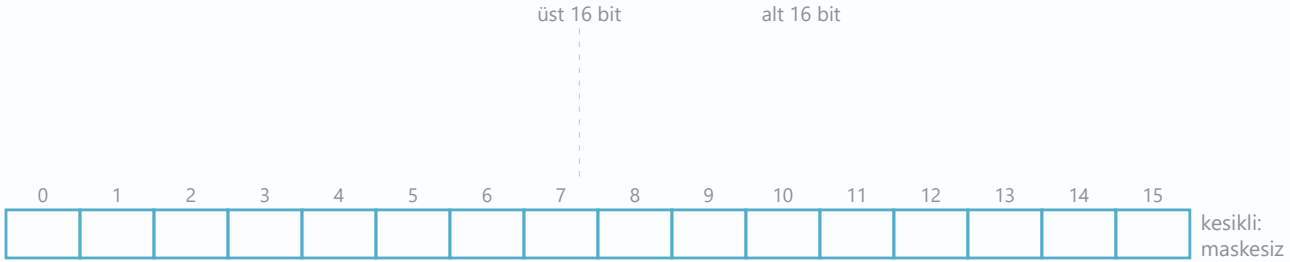
λ = 0,75 eşiği

Öğe sayısı 0,75·n'yi aşınca dizi **2 katına** büyür, düğümler yeni kovalara dağıtılır.

Ağaçlaşma (Java 8+)

Zincir 8 düğümü aşarsa ağaca çevrilir (n ≥ 64 ise; değilse tablo büyür): en kötü O(log n).

Kod İzleme: `indis(k, 16)` — karıştır, maskele



16 kovalı (n = 2^4) tablo. Üç anahtarın **alt 16 biti aynı** (...0001); yalnızca üst bitleri farklı. Maske `& 15` yalnız son 4 biti görür.

Adım 1 / 11

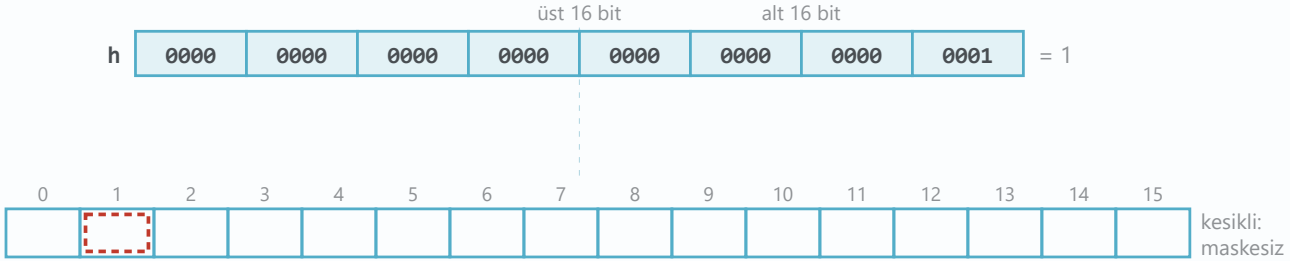
```
1 static int indis(Object key, int n) { // n = 2^k
2   int h = key.hashCode();
3   int m = h ^ (h >>> 16); // üst bitleri karıştır
4   return m & (n - 1); // alt k bit
5 }
6 // main:
▶ for (int k : new int[]{1, 65537, 131073})
8   System.out.println(k + " -> " + indis(k, 16));
```

DEĞİŞKENLER

`n = 16`

ÇIKTI

Kod İzleme: `indis(k, 16)` — karıştır, maskele



`Integer.hashCode()` değerin kendisidir: **h = 1** = 0x00000001. Maskesiz kova h & 15 = **1** olurdu (kesikli).

Adım 2 / 11

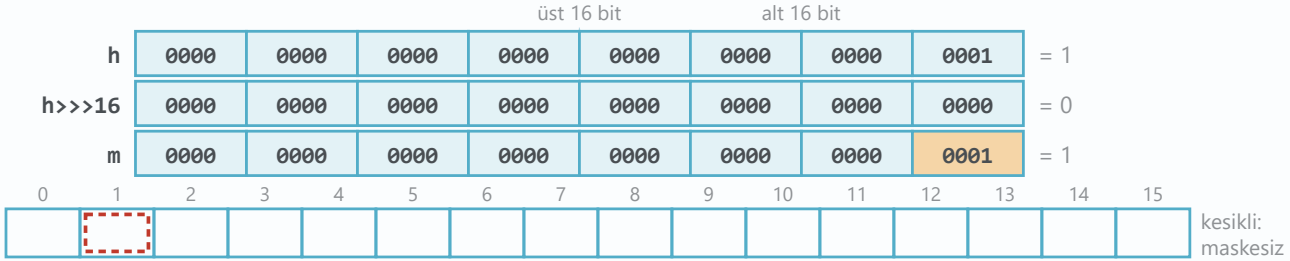
```
1 static int indis(Object key, int n) { // n = 2^k
▶   int h = key.hashCode();
3   int m = h ^ (h >>> 16); // üst bitleri karıştır
4   return m & (n - 1); // alt k bit
5 }
6 // main:
7 for (int k : new int[]{1, 65537, 131073})
8   System.out.println(k + " -> " + indis(k, 16));
```

DEĞİŞKENLER

`k = 1` `key = 1` `n = 16` `h = 1`

ÇIKTI

Kod İzleme: `indis(k, 16)` — karıştır, maskele



`h >>> 16` = 0: üst yarı alta kayar. XOR ile **m = 1**; artık alt 4 bit = 0001.

Adım 3 / 11

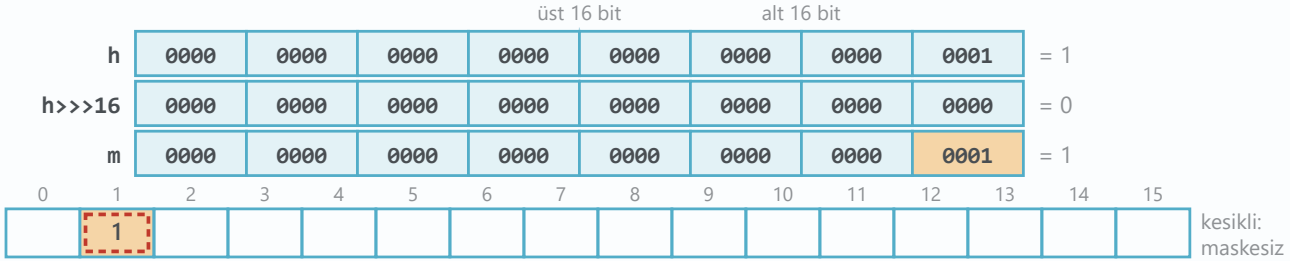
```
1  static int indis(Object key, int n) { // n = 2^k
2    int h = key.hashCode();
▶    int m = h ^ (h >>> 16); // üst bitleri karıştır
4    return m & (n - 1); // alt k bit
5  }
6  // main:
7  for (int k : new int[]{1, 65537, 131073})
8    System.out.println(k + " -> " + indis(k, 16));
```

DEĞİŞKENLER

`k = 1` `key = 1` `n = 16` `h = 1` `m = 1`

ÇIKTI

Kod İzleme: `indis(k, 16)` — karıştır, maskele



`m & 15` = **1** → anahtar 1 kova 1'e gider ve yazdırılır.

Adım 4 / 11

```
1  static int indis(Object key, int n) { // n = 2^k
2    int h = key.hashCode();
3    int m = h ^ (h >>> 16); // üst bitleri karıştır
▶    return m & (n - 1); // alt k bit
5  }
6  // main:
7  for (int k : new int[]{1, 65537, 131073})
▶    System.out.println(k + " -> " + indis(k, 16));
```

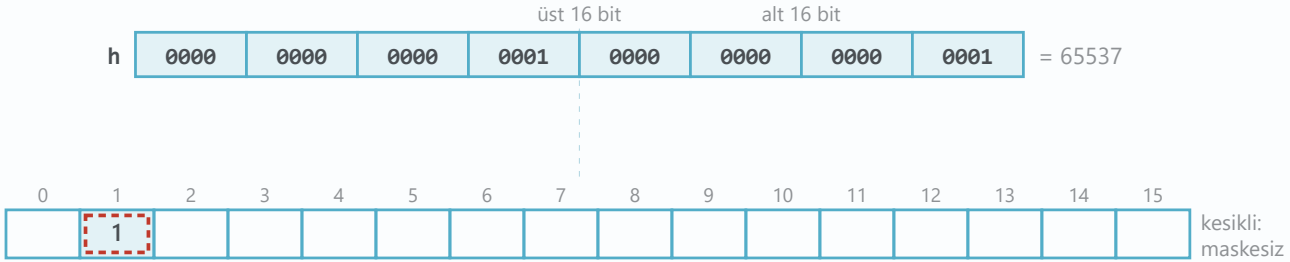
DEĞİŞKENLER

`k = 1` `key = 1` `n = 16` `h = 1` `m = 1`

ÇIKTI

```
1 -> 1
```

Kod İzleme: `indis(k, 16)` — karıştır, maskele



`Integer.hashCode()` değerin kendisidir: **h = 65537** = 0x00010001. Maskesiz kova h & 15 = **1** olurdu (kesikli).

Adım 5 / 11

```
1  static int indis(Object key, int n) { // n = 2^k
▶    int h = key.hashCode();
3    int m = h ^ (h >>> 16); // üst bitleri karıştır
4    return m & (n - 1); // alt k bit
5  }
6  // main:
7  for (int k : new int[]{1, 65537, 131073})
8    System.out.println(k + " -> " + indis(k, 16));
```

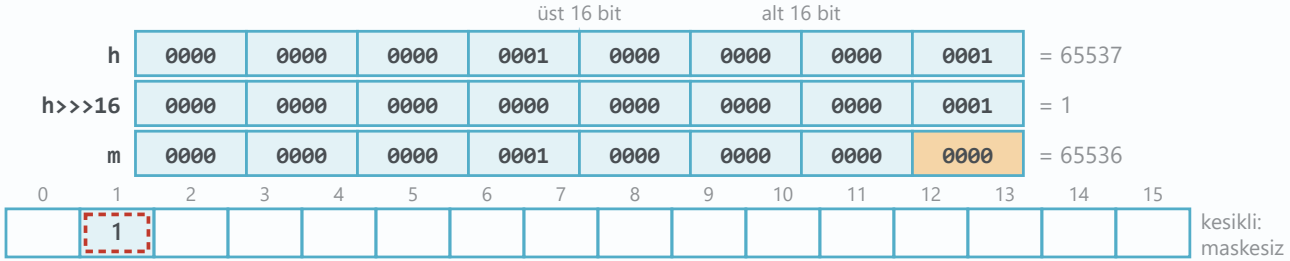
DEĞİŞKENLER

`k = 65537` `key = 65537` `n = 16` `h = 65537`

ÇIKTI

```
1 -> 1
```

Kod İzleme: `indis(k, 16)` — karıştır, maskele



`h >>> 16` = 1: üst yarı alta kayar. XOR ile **m = 65536**; artık alt 4 bit = 0000.

Adım 6 / 11

```
1  static int indis(Object key, int n) { // n = 2^k
2    int h = key.hashCode();
▶    int m = h ^ (h >>> 16); // üst bitleri karıştır
4    return m & (n - 1); // alt k bit
5  }
6  // main:
7  for (int k : new int[]{1, 65537, 131073})
8    System.out.println(k + " -> " + indis(k, 16));
```

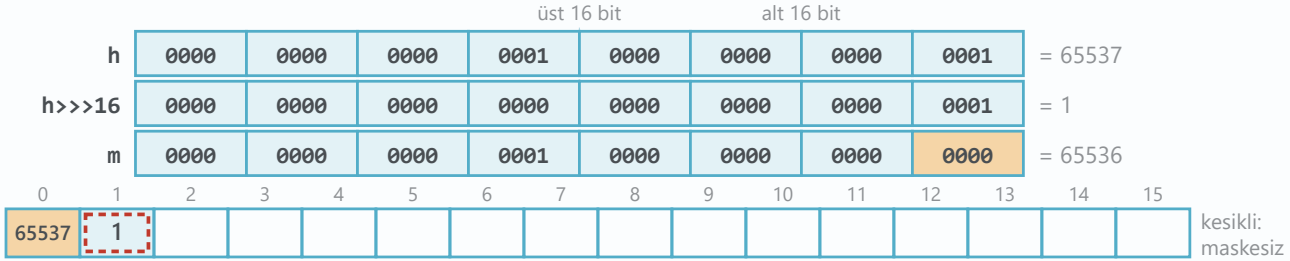
DEĞİŞKENLER

`k = 65537` `key = 65537` `n = 16` `h = 65537` `m = 65536`

ÇIKTI

```
1 -> 1
```

Kod İzleme: `indis(k, 16)` — karıştır, maskele



`m & 15` = **0** → anahtar 65537 kova 0'a gider ve yazdırılır.

Adım 7 / 11

```
1  static int indis(Object key, int n) { // n = 2^k
2    int h = key.hashCode();
3    int m = h ^ (h >>> 16); // üst bitleri karıştır
▶    return m & (n - 1); // alt k bit
5  }
6  // main:
7  for (int k : new int[]{1, 65537, 131073})
▶    System.out.println(k + " -> " + indis(k, 16));
```

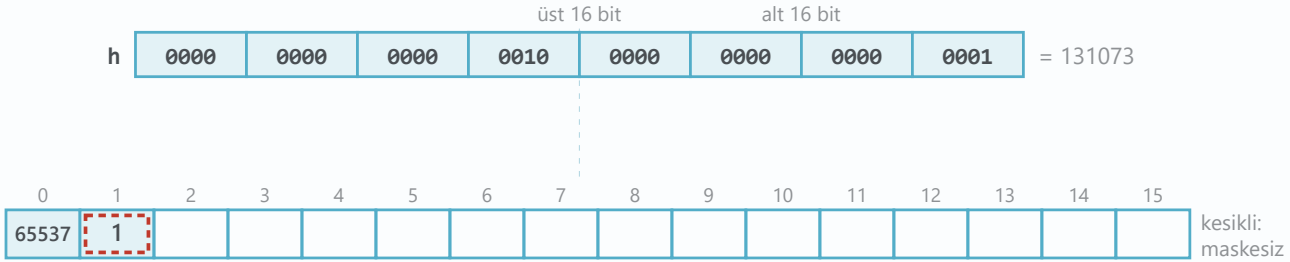
DEĞİŞKENLER

`k = 65537` `key = 65537` `n = 16` `h = 65537` `m = 65536`

ÇIKTI

```
1 -> 1
65537 -> 0
```

Kod İzleme: `indis(k, 16)` — karıştır, maskele



`Integer.hashCode()` değerin kendisidir: **h = 131073** = 0x00020001. Maskesiz kova h & 15 = **1** olurdu (kesikli).

Adım 8 / 11

```
1  static int indis(Object key, int n) { // n = 2^k
▶    int h = key.hashCode();
3    int m = h ^ (h >>> 16); // üst bitleri karıştır
4    return m & (n - 1); // alt k bit
5  }
6  // main:
7  for (int k : new int[]{1, 65537, 131073})
8    System.out.println(k + " -> " + indis(k, 16));
```

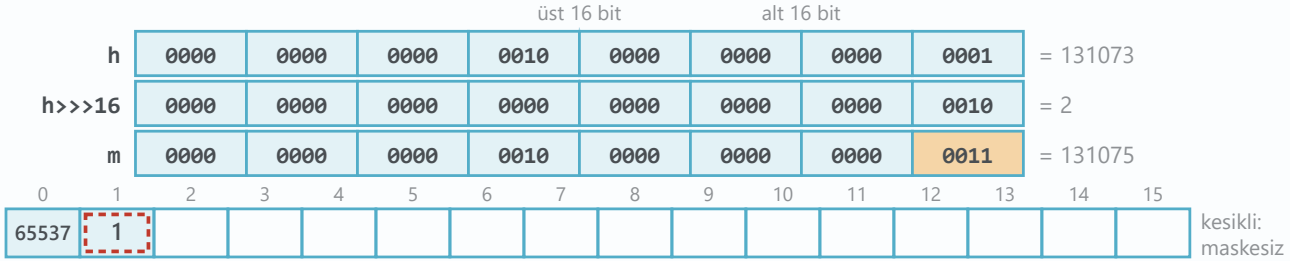
DEĞİŞKENLER

`k = 131073` `key = 131073` `n = 16` `h = 131073`

ÇIKTI

```
1 -> 1
65537 -> 0
```

Kod İzleme: `indis(k, 16)` — karıştır, maskele



`h >>> 16` = 2: üst yarı alta kayar. XOR ile **m = 131075**; artık alt 4 bit = 0011.

Adım 9 / 11

```
1  static int indis(Object key, int n) { // n = 2^k
2    int h = key.hashCode();
▶    int m = h ^ (h >>> 16); // üst bitleri karıştır
4    return m & (n - 1); // alt k bit
5  }
6  // main:
7  for (int k : new int[]{1, 65537, 131073})
8    System.out.println(k + " -> " + indis(k, 16));
```

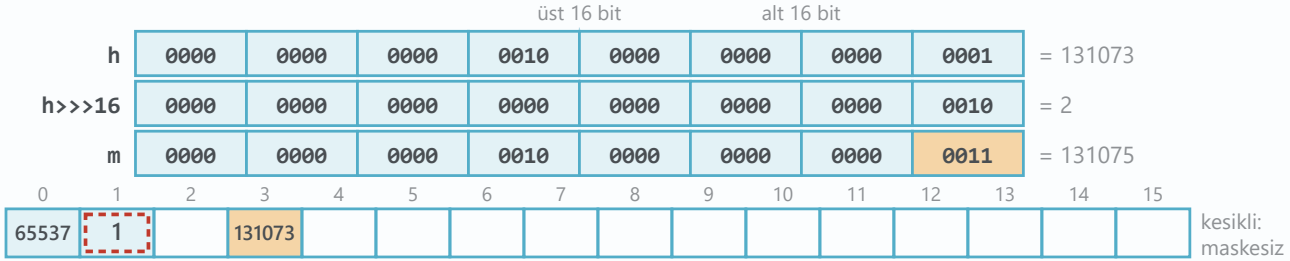
DEĞİŞKENLER

`k = 131073` `key = 131073` `n = 16` `h = 131073` `m = 131075`

ÇIKTI

```
1 -> 1
65537 -> 0
```

Kod İzleme: `indis(k, 16)` — karıştır, maskele



`m & 15` = **3** → anahtar 131073 kova 3'e gider ve yazdırılır.

Adım 10 / 11

```
1  static int indis(Object key, int n) { // n = 2^k
2    int h = key.hashCode();
3    int m = h ^ (h >>> 16); // üst bitleri karıştır
▶    return m & (n - 1); // alt k bit
5  }
6  // main:
7  for (int k : new int[]{1, 65537, 131073})
▶    System.out.println(k + " -> " + indis(k, 16));
```

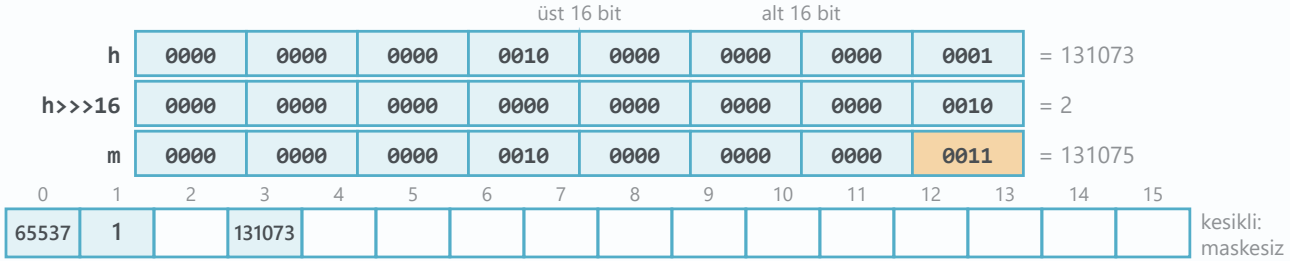
DEĞİŞKENLER

`k = 131073` `key = 131073` `n = 16` `h = 131073` `m = 131075`

ÇIKTI

```
1 -> 1
65537 -> 0
131073 -> 3
```

Kod İzleme: `indis(k, 16)` — karıştır, maskele



Karıştırmasız üçü de kova 1'de çakışırdı; karıştırmayla 1, 0, 3 kovalarına dağıldılar. Maliyet: bir kaydırma + bir XOR.

Adım 11 / 11

```
1 static int indis(Object key, int n) { // n = 2^k
2   int h = key.hashCode();
3   int m = h ^ (h >>> 16); // üst bitleri karıştır
4   return m & (n - 1); // alt k bit
5 }
6 // main:
▶ for (int k : new int[]{1, 65537, 131073})
8   System.out.println(k + " -> " + indis(k, 16));
```

DEĞİŞKENLER

`n = 16`

ÇIKTI

```
1 -> 1
65537 -> 0
131073 -> 3
```

PERFORMANS VE UYGULAMALAR

Hash Tablolarının Kullanım Alanları

Sözlük

Kelime → anlam; yazım denetiminde "bu kelime var mı?"

Veritabanı

Hash indisleri ve hash join ile anahtara göre hızlı erişim.

Bellek önbelleği

URL / sorgu → önceden hesaplanmış sonuç (ör. Redis, memcached).

Web tarayıcı geçmişi

Ziyaret edilen adres → son ziyaret zamanı; "daha önce açıldı mı?"

Çerez yönetimi

Çerez adı → değer; oturum kimliği → kullanıcı bilgisi.

Derleyici

Sembol tablosu: değişken adı → tip, adres. Java: `HashMap`, `HashSet`.

Uygulama: `ikiToplam([8, 3, 11, 5, 7], 12)`

a

hedef = 12

0	1	2	3	4
8	3	11	5	7

gor (değer → indis)

{ } boş

Toplamı 12 olan iki indis aranıyor. Kaba kuvvet tüm çiftleri dener: O(n^2). Burada bir **eşleme**, görülen değerleri indisleriyle tutar.

Adım 1 / 11

```
1  static int[] ikiToplam(int[] a, int hedef) {
▶    Map<Integer, Integer> gor = new HashMap<>();
3    for (int i = 0; i < a.length; i++) {
4      int ek = hedef - a[i]; // gereken eş
5      if (gor.containsKey(ek))
6        return new int[]{gor.get(ek), i};
7      gor.put(a[i], i); // değer -> indis
8    }
9    return null; // çift yok
10 }
```

DEĞİŞKENLER

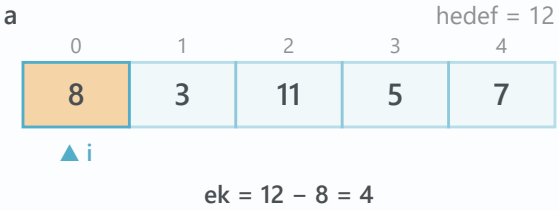
`a = [8, 3, 11, 5, 7]` `hedef = 12` `gor = {}`

ÇAĞRI YIĞINI

`ikiToplam(a, 12)`

`main()`

Uygulama: `ikiToplam([8, 3, 11, 5, 7], 12)`



i = 0: a[0] = 8, gereken eş **ek = 4**. `containsKey(4)` → false: 4 daha önce görülmedi.

Adım 2 / 11

```
1  static int[] ikiToplam(int[] a, int hedef) {
2    Map<Integer, Integer> gor = new HashMap<>();
▶    for (int i = 0; i < a.length; i++) {
▶      int ek = hedef - a[i]; // gereken eş
▶      if (gor.containsKey(ek))
6        return new int[]{gor.get(ek), i};
7      gor.put(a[i], i); // değer -> indis
8    }
9    return null; // çift yok
10 }
```

DEĞİŞKENLER

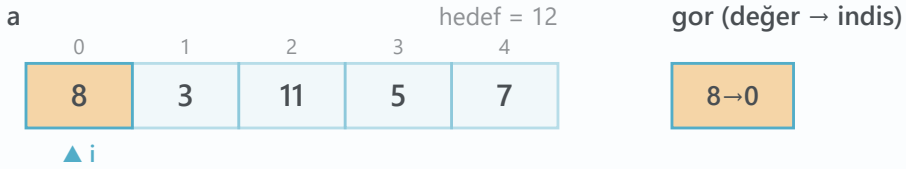
`a = [8, 3, 11, 5, 7]` `hedef = 12` `i = 0` `ek = 4` `gor = {}`

ÇAĞRI YIĞINI

`ikiToplam(a, 12)`

`main()`

Uygulama: `ikiToplam([8, 3, 11, 5, 7], 12)`



`gor.put(8, 0)` : sonraki öğeler 8'i eş olarak arayabilsin.

Adım 3 / 11

```
1  static int[] ikiToplam(int[] a, int hedef) {
2    Map<Integer, Integer> gor = new HashMap<>();
3    for (int i = 0; i < a.length; i++) {
4      int ek = hedef - a[i]; // gereken eş
5      if (gor.containsKey(ek))
6        return new int[]{gor.get(ek), i};
▶      gor.put(a[i], i); // değer -> indis
8    }
9    return null; // çift yok
10 }
```

DEĞİŞKENLER

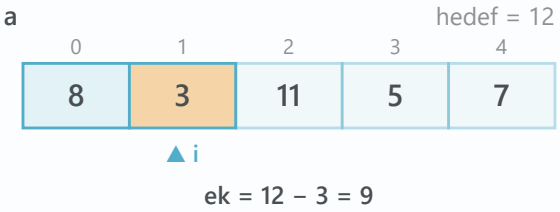
`a = [8, 3, 11, 5, 7]` `hedef = 12` `i = 0` `ek = 4` `gor = {8=0}`

ÇAĞRI YIĞINI

`ikiToplam(a, 12)`

`main()`

Uygulama: `ikiToplam([8, 3, 11, 5, 7], 12)`



i = 1: a[1] = 3, gereken eş **ek = 9**. `containsKey(9)` → false: 9 daha önce görülmedi.

Adım 4 / 11

```
1  static int[] ikiToplam(int[] a, int hedef) {
2    Map<Integer, Integer> gor = new HashMap<>();
▶    for (int i = 0; i < a.length; i++) {
▶      int ek = hedef - a[i]; // gereken eş
▶      if (gor.containsKey(ek))
6        return new int[]{gor.get(ek), i};
7      gor.put(a[i], i); // değer -> indis
8    }
9    return null; // çift yok
10 }
```

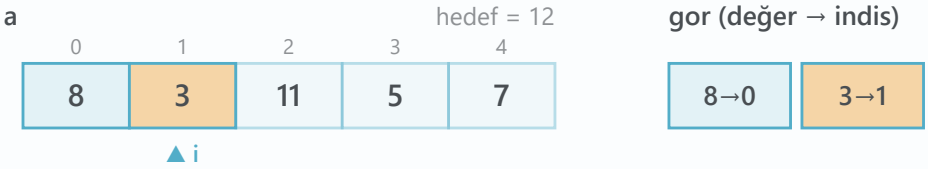
DEĞİŞKENLER

`a = [8, 3, 11, 5, 7]` `hedef = 12` `i = 1` `ek = 9` `gor = {8=0}`

ÇAĞRI YIĞINI

- `ikiToplam(a, 12)`
- `main()`

Uygulama: `ikiToplam([8, 3, 11, 5, 7], 12)`



`gor.put(3, 1)` : sonraki öğeler 3'i eş olarak arayabilsin.

Adım 5 / 11

```
1  static int[] ikiToplam(int[] a, int hedef) {
2    Map<Integer, Integer> gor = new HashMap<>();
3    for (int i = 0; i < a.length; i++) {
4      int ek = hedef - a[i]; // gereken eş
5      if (gor.containsKey(ek))
6        return new int[]{gor.get(ek), i};
▶      gor.put(a[i], i); // değer -> indis
8    }
9    return null; // çift yok
10 }
```

DEĞİŞKENLER

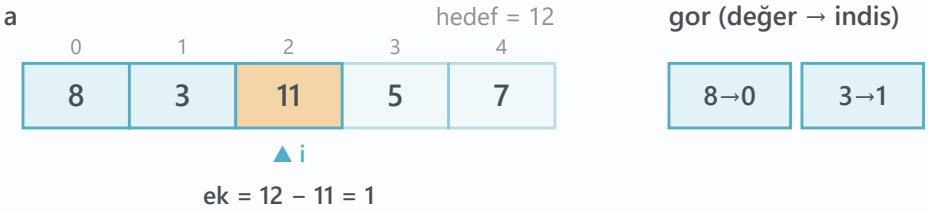
`a = [8, 3, 11, 5, 7]` `hedef = 12` `i = 1` `ek = 9` `gor = {8=0, 3=1}`

ÇAĞRI YIĞINI

`ikiToplam(a, 12)`

`main()`

Uygulama: `ikiToplam([8, 3, 11, 5, 7], 12)`



i = 2: a[2] = 11, gereken eş **ek = 1**. `containsKey(1)` → false: 1 daha önce görülmedi.


```
1  static int[] ikiToplam(int[] a, int hedef) {
2    Map<Integer, Integer> gor = new HashMap<>();
▶    for (int i = 0; i < a.length; i++) {
▶      int ek = hedef - a[i]; // gereken eş
▶      if (gor.containsKey(ek))
6        return new int[]{gor.get(ek), i};
7      gor.put(a[i], i); // değer -> indis
8    }
9    return null; // çift yok
10 }
```

DEĞİŞKENLER

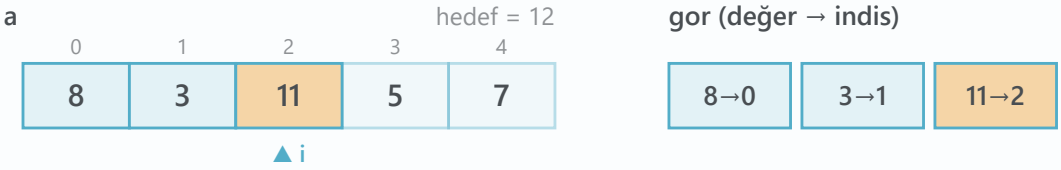
`a = [8, 3, 11, 5, 7]` `hedef = 12` `i = 2` `ek = 1` `gor = {8=0, 3=1}`

ÇAĞRI YIĞINI

`ikiToplam(a, 12)`

`main()`

Uygulama: `ikiToplam([8, 3, 11, 5, 7], 12)`



`gor.put(11, 2)` : sonraki öğeler 11'i eş olarak arayabilsin.

Adım 7 / 11

```
1  static int[] ikiToplam(int[] a, int hedef) {
2    Map<Integer, Integer> gor = new HashMap<>();
3    for (int i = 0; i < a.length; i++) {
4      int ek = hedef - a[i]; // gereken eş
5      if (gor.containsKey(ek))
6        return new int[]{gor.get(ek), i};
▶      gor.put(a[i], i); // değer -> indis
8    }
9    return null; // çift yok
10 }
```

DEĞİŞKENLER

`a = [8, 3, 11, 5, 7]` `hedef = 12`

`i = 2` `ek = 1`

`gor = {8=0, 3=1, 11=2}`

ÇAĞRI YIĞINI

`ikiToplam(a, 12)`

`main()`

Uygulama: `ikiToplam([8, 3, 11, 5, 7], 12)`



i = 3: a[3] = 5, gereken eş **ek = 7**. `containsKey(7)` → false: 7 daha önce görülmedi.

Adım 8 / 11

```
1  static int[] ikiToplam(int[] a, int hedef) {
2    Map<Integer, Integer> gor = new HashMap<>();
▶    for (int i = 0; i < a.length; i++) {
▶      int ek = hedef - a[i]; // gereken eş
▶      if (gor.containsKey(ek))
6        return new int[]{gor.get(ek), i};
7      gor.put(a[i], i); // değer -> indis
8    }
9    return null; // çift yok
10 }
```

DEĞİŞKENLER

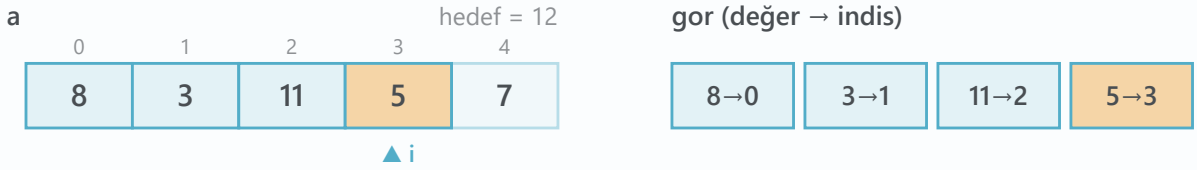
`a = [8, 3, 11, 5, 7]` `hedef = 12` `i = 3` `ek = 7` `gor = {8=0, 3=1, 11=2}`

ÇAĞRI YIĞINI

`ikiToplam(a, 12)`

`main()`

Uygulama: `ikiToplam([8, 3, 11, 5, 7], 12)`



`gor.put(5, 3)` : sonraki öğeler 5'i eş olarak arayabilsin.

Adım 9 / 11

```
1  static int[] ikiToplam(int[] a, int hedef) {
2    Map<Integer, Integer> gor = new HashMap<>();
3    for (int i = 0; i < a.length; i++) {
4      int ek = hedef - a[i]; // gereken eş
5      if (gor.containsKey(ek))
6        return new int[]{gor.get(ek), i};
▶      gor.put(a[i], i); // değer -> indis
8    }
9    return null; // çift yok
10 }
```

DEĞİŞKENLER

`a = [8, 3, 11, 5, 7]` `hedef = 12` `i = 3` `ek = 7`

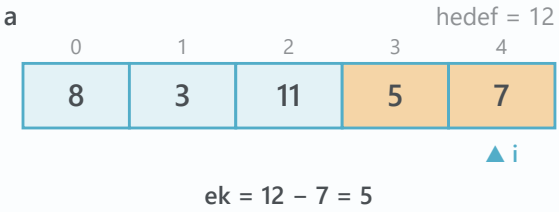
`gor = {8=0, 3=1, 11=2, 5=3}`

ÇAĞRI YIĞINI

`ikiToplam(a, 12)`

`main()`

Uygulama: `ikiToplam([8, 3, 11, 5, 7], 12)`





i = 4: ek = 5, `containsKey(5)` **true** (O(1) ortalama) → a[3] + a[4] = 5 + 7 = 12. **return [3, 4]**.

Adım 10 / 11

```
1  static int[] ikiToplam(int[] a, int hedef) {
2    Map<Integer, Integer> gor = new HashMap<>();
3    for (int i = 0; i < a.length; i++) {
▶      int ek = hedef - a[i]; // gereken eş
▶      if (gor.containsKey(ek))
▶        return new int[]{gor.get(ek), i};
7      gor.put(a[i], i); // değer -> indis
8    }
9    return null; // çift yok
10 }
```

DEĞİŞKENLER

`a = [8, 3, 11, 5, 7]` `hedef = 12` `i = 4` `ek = 5` `gor = {8=0, 3=1, 11=2, 5=3}`

ÇAĞRI YIĞINI

`ikiToplam(a, 12)`

`main()`

Uygulama: `ikiToplam([8, 3, 11, 5, 7], 12)`



Tek geçiş, her adımda ortalama O(1) arama + ekleme → toplam **O(n)** zaman, O(n) ek bellek.

Adım 11 / 11

```
1  static int[] ikiToplam(int[] a, int hedef) {
2    Map<Integer, Integer> gor = new HashMap<>();
3    for (int i = 0; i < a.length; i++) {
4      int ek = hedef - a[i]; // gereken eş
5      if (gor.containsKey(ek))
▶        return new int[]{gor.get(ek), i};
7      gor.put(a[i], i); // değer -> indis
8    }
9    return null; // çift yok
10 }
```

DEĞİŞKENLER

`a = [8, 3, 11, 5, 7]` `hedef = 12` `i = 4` `ek = 5` `gor = {8=0, 3=1, 11=2, 5=3}`

ÇAĞRI YIĞINI

`main()`

ÖRNEK UYGULAMA

Okul Numarası İçin Hash Fonksiyonu

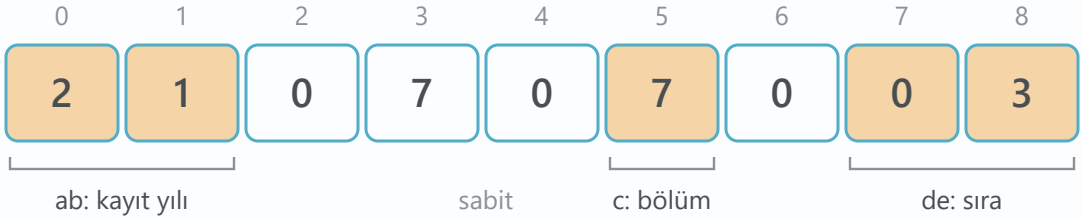
Sorun

Okul numaralarını 1000 öğelik bir diziye eşit şekilde dağıtmak.

Çözüm

Numaranın yapısını kullanan bir hash fonksiyonu tasarlayarak çakışmaları azaltmak.

Numara biçimi: `ab070c0de` · örnek: `210707003`



- **ab**: kayıt yılı (20–24) · **c**: bölüm (6 veya 7) · **de**: kayıt sırası (00–99, iki hane)
- Hedef: bu biçimi 0–999 aralığında bir dizi indisine dönüştürmek.

ÖRNEK UYGULAMA

Okul Numarası için Hash Fonksiyonu

$$\text{hash}(ab070c0de) = ((ab - 20) \times 100 + (c - 6) \times 500 + de) \bmod 1000$$

Numara	ab	c	de	Hesap	İndis
`210706003`	21	6	03	(1×100 + 0×500 + 3) mod 1000	**103**
`240707050`	24	7	50	(4×100 + 1×500 + 50) mod 1000	**950**
`230707042`	23	7	42	(3×100 + 1×500 + 42) mod 1000	**842**

Yıl
$(ab - 20) \in 0\ldots4 \rightarrow$ 100'lük blok seçer.

Bölüm
$(c - 6) \in 0, 1 \rightarrow$ 500 kaydırır.

Sıra
$de \in 0\ldots99$ doğrudan eklenir.

5 yıl × 2 bölüm × 100 sıra = **1000** olası numara, 1000 farklı indise gider: çakışma hiç yok. Bu bir **mükemmel hash** (perfect hash) örneğidir; anahtar yapısı bilindiği için mümkündür.

ÖRNEK UYGULAMA

calculateHash Fonksiyonu

```java
// s: "ab070c0de" biçiminde okul numarası
public int calculateHash(String s) {
  int ab = Integer.parseInt(s.substring(0, 2));
  int c  = Integer.parseInt(s.substring(5, 6));
  int de = Integer.parseInt(s.substring(7, 9));
  int yil   = (ab - 20) * 100;   // yıl bloğu
  int bolum = (c - 6) * 500;     // bölüm: 0/500
  return (yil + bolum + de) % 1000;
}
```

substring(başlangıç, bitiş)

bitiş dahil değildir: `substring(5, 6)` yalnızca 5. karakteri (c) verir; `substring(7, 9)` 7. ve 8. karakterleri (de).

Örnek

`calculateHash("240707050")`

ab = 24, c = 7, de = 50

400 + 500 + 50 = **950**

Biçime uymayan girdi (yıl 20–24 dışı, c ∉ {6, 7}) için sonuç çakışabilir ya da negatif olabilir; gerçek kodda girdi doğrulanmalıdır.

ÖRNEK UYGULAMA · KODU ADIM ADIM ÇALIŞTIR

Kod İzleme: `calculateHash()`

0	1	2	3	4	5	6	7	8
2	4	0	7	0	7	0	5	0

`calculateHash("240707050")` : indisler 0 … 8.

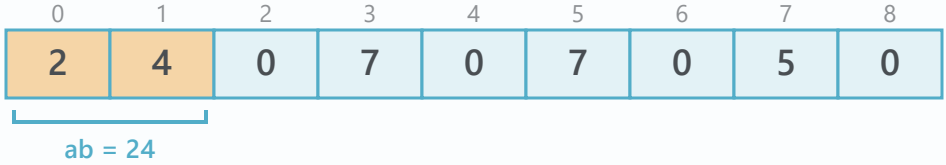
Adım 1 / 14

```
1 // s: "ab070c0de" biçiminde okul numarası
▶ public int calculateHash(String s) {
3   int ab = Integer.parseInt(s.substring(0, 2));
4   int c  = Integer.parseInt(s.substring(5, 6));
5   int de = Integer.parseInt(s.substring(7, 9));
6   int yil   = (ab - 20) * 100;   // yıl bloğu
7   int bolum = (c - 6) * 500;     // bölüm: 0/500
8   return (yil + bolum + de) % 1000;
9 }
```

DEĞİŞKENLER

`s = "240707050"`

Kod İzleme: `calculateHash()`



`substring(0, 2)` = "24" (bitiş hariç) → **ab = 24** (kayıt yılı).

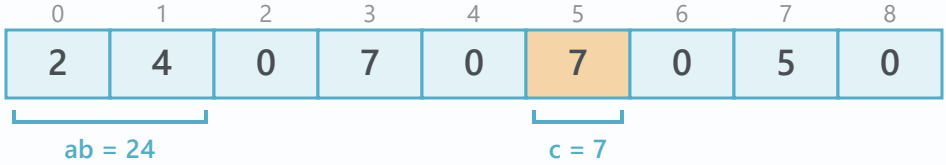
Adım 2 / 14

```
1 // s: "ab070c0de" biçiminde okul numarası
2 public int calculateHash(String s) {
▶   int ab = Integer.parseInt(s.substring(0, 2));
4   int c  = Integer.parseInt(s.substring(5, 6));
5   int de = Integer.parseInt(s.substring(7, 9));
6   int yil   = (ab - 20) * 100;   // yıl bloğu
7   int bolum = (c - 6) * 500;     // bölüm: 0/500
8   return (yil + bolum + de) % 1000;
9 }
```

DEĞİŞKENLER

`s = "240707050"` `ab = 24`

Kod İzleme: `calculateHash()`



`substring(5, 6)` = "7" → **c = 7** (bölüm).

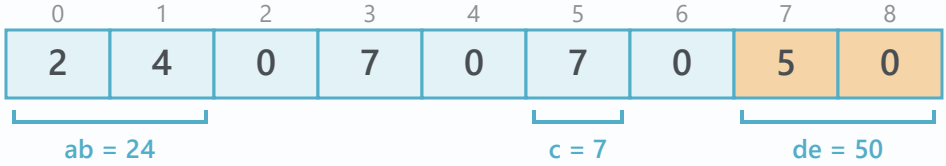
Adım 3 / 14

```
1 // s: "ab070c0de" biçiminde okul numarası
2 public int calculateHash(String s) {
3   int ab = Integer.parseInt(s.substring(0, 2));
▶   int c  = Integer.parseInt(s.substring(5, 6));
5   int de = Integer.parseInt(s.substring(7, 9));
6   int yil   = (ab - 20) * 100;   // yıl bloğu
7   int bolum = (c - 6) * 500;     // bölüm: 0/500
8   return (yil + bolum + de) % 1000;
9 }
```

DEĞİŞKENLER

`s = "240707050"` `ab = 24` `c = 7`

Kod İzleme: `calculateHash()`



`substring(7, 9)` = "50" → **de = 50** (kayıt sırası).

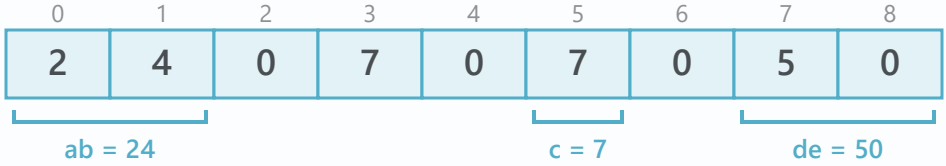
Adım 4 / 14

```
1 // s: "ab070c0de" biçiminde okul numarası
2 public int calculateHash(String s) {
3   int ab = Integer.parseInt(s.substring(0, 2));
4   int c  = Integer.parseInt(s.substring(5, 6));
▶   int de = Integer.parseInt(s.substring(7, 9));
6   int yil   = (ab - 20) * 100;   // yıl bloğu
7   int bolum = (c - 6) * 500;     // bölüm: 0/500
8   return (yil + bolum + de) % 1000;
9 }
```

DEĞİŞKENLER

`s = "240707050"` `ab = 24` `c = 7` `de = 50`

Kod İzleme: `calculateHash()`



yil = (24 − 20) · 100 = **400**: her yıl 100'lük bir blok.

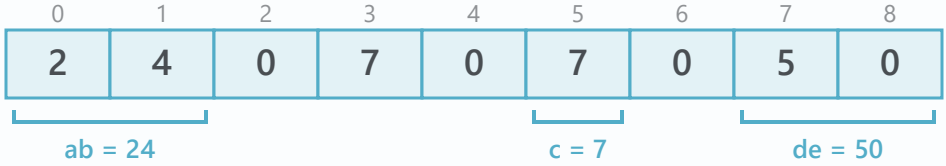
Adım 5 / 14

```
1 // s: "ab070c0de" biçiminde okul numarası
2 public int calculateHash(String s) {
3   int ab = Integer.parseInt(s.substring(0, 2));
4   int c  = Integer.parseInt(s.substring(5, 6));
5   int de = Integer.parseInt(s.substring(7, 9));
▶   int yil   = (ab - 20) * 100;   // yıl bloğu
7   int bolum = (c - 6) * 500;     // bölüm: 0/500
8   return (yil + bolum + de) % 1000;
9 }
```

DEĞİŞKENLER

`s = "240707050"` `ab = 24` `c = 7` `de = 50` `yil = 400`

Kod İzleme: `calculateHash()`



bolum = (7 − 6) · 500 = **500**: bölüm 7 ise 500 kaydırılır.

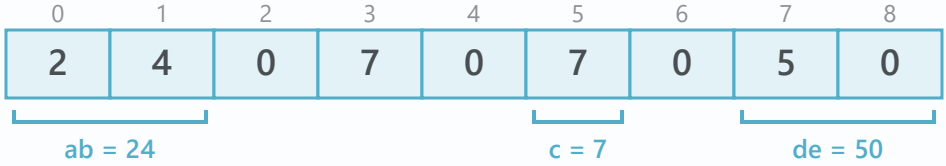
Adım 6 / 14

```
1 // s: "ab070c0de" biçiminde okul numarası
2 public int calculateHash(String s) {
3   int ab = Integer.parseInt(s.substring(0, 2));
4   int c  = Integer.parseInt(s.substring(5, 6));
5   int de = Integer.parseInt(s.substring(7, 9));
6   int yil   = (ab - 20) * 100;   // yıl bloğu
▶   int bolum = (c - 6) * 500;     // bölüm: 0/500
8   return (yil + bolum + de) % 1000;
9 }
```

DEĞİŞKENLER

`s = "240707050"` `ab = 24` `c = 7` `de = 50` `yil = 400` `bolum = 500`

Kod İzleme: `calculateHash()`



`return (400 + 500 + 50) % 1000` = **950**.

Adım 7 / 14

```
1 // s: "ab070c0de" biçiminde okul numarası
2 public int calculateHash(String s) {
3   int ab = Integer.parseInt(s.substring(0, 2));
4   int c  = Integer.parseInt(s.substring(5, 6));
5   int de = Integer.parseInt(s.substring(7, 9));
6   int yil   = (ab - 20) * 100;   // yıl bloğu
7   int bolum = (c - 6) * 500;     // bölüm: 0/500
▶   return (yil + bolum + de) % 1000;
9 }
```

DEĞİŞKENLER

`s = "240707050"` `ab = 24` `c = 7` `de = 50` `yil = 400` `bolum = 500`

Kod İzleme: `calculateHash()`

0	1	2	3	4	5	6	7	8
2	1	0	7	0	6	0	0	3

`calculateHash("210706003")` : indisler 0 … 8.

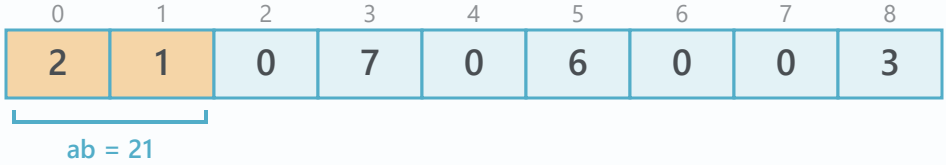
Adım 8 / 14

```
1 // s: "ab070c0de" biçiminde okul numarası
▶ public int calculateHash(String s) {
3   int ab = Integer.parseInt(s.substring(0, 2));
4   int c  = Integer.parseInt(s.substring(5, 6));
5   int de = Integer.parseInt(s.substring(7, 9));
6   int yil   = (ab - 20) * 100;   // yıl bloğu
7   int bolum = (c - 6) * 500;     // bölüm: 0/500
8   return (yil + bolum + de) % 1000;
9 }
```

DEĞİŞKENLER

`s = "210706003"`

Kod İzleme: `calculateHash()`



`substring(0, 2)` = "21" (bitiş hariç) → **ab = 21** (kayıt yılı).

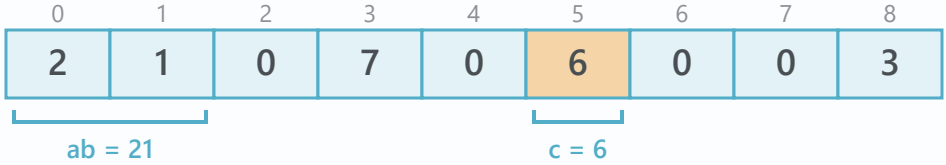
Adım 9 / 14

```
1 // s: "ab070c0de" biçiminde okul numarası
2 public int calculateHash(String s) {
▶   int ab = Integer.parseInt(s.substring(0, 2));
4   int c  = Integer.parseInt(s.substring(5, 6));
5   int de = Integer.parseInt(s.substring(7, 9));
6   int yil   = (ab - 20) * 100;   // yıl bloğu
7   int bolum = (c - 6) * 500;     // bölüm: 0/500
8   return (yil + bolum + de) % 1000;
9 }
```

DEĞİŞKENLER

`s = "210706003"` `ab = 21`

Kod İzleme: `calculateHash()`



`substring(5, 6)` = "6" → **c = 6** (bölüm).

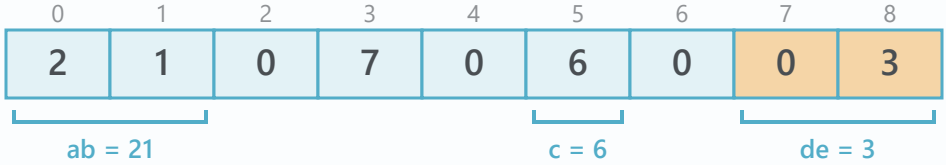
Adım 10 / 14

```
1 // s: "ab070c0de" biçiminde okul numarası
2 public int calculateHash(String s) {
3   int ab = Integer.parseInt(s.substring(0, 2));
▶   int c  = Integer.parseInt(s.substring(5, 6));
5   int de = Integer.parseInt(s.substring(7, 9));
6   int yil   = (ab - 20) * 100;   // yıl bloğu
7   int bolum = (c - 6) * 500;     // bölüm: 0/500
8   return (yil + bolum + de) % 1000;
9 }
```

DEĞİŞKENLER

`s = "210706003"` `ab = 21` `c = 6`

Kod İzleme: `calculateHash()`



`substring(7, 9)` = "03" → **de = 3** (kayıt sırası).

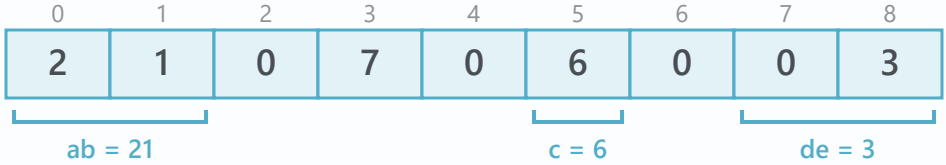
Adım 11 / 14

```
1 // s: "ab070c0de" biçiminde okul numarası
2 public int calculateHash(String s) {
3   int ab = Integer.parseInt(s.substring(0, 2));
4   int c  = Integer.parseInt(s.substring(5, 6));
▶   int de = Integer.parseInt(s.substring(7, 9));
6   int yil   = (ab - 20) * 100;   // yıl bloğu
7   int bolum = (c - 6) * 500;     // bölüm: 0/500
8   return (yil + bolum + de) % 1000;
9 }
```

DEĞİŞKENLER

`s = "210706003"` `ab = 21` `c = 6` `de = 3`

Kod İzleme: `calculateHash()`



yil = (21 − 20) · 100 = **100**: her yıl 100'lük bir blok.

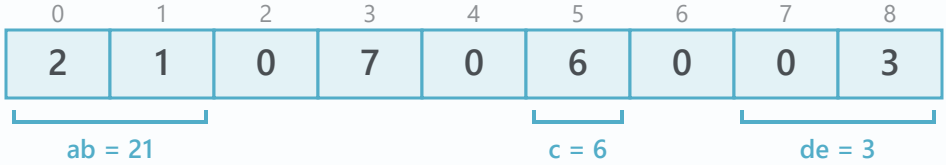
Adım 12 / 14

```
1 // s: "ab070c0de" biçiminde okul numarası
2 public int calculateHash(String s) {
3   int ab = Integer.parseInt(s.substring(0, 2));
4   int c  = Integer.parseInt(s.substring(5, 6));
5   int de = Integer.parseInt(s.substring(7, 9));
▶   int yil   = (ab - 20) * 100;   // yıl bloğu
7   int bolum = (c - 6) * 500;     // bölüm: 0/500
8   return (yil + bolum + de) % 1000;
9 }
```

DEĞİŞKENLER

`s = "210706003"` `ab = 21` `c = 6` `de = 3` `yil = 100`

Kod İzleme: `calculateHash()`



bolum = (6 − 6) · 500 = **0**: bölüm 6 ise kaydırma yok.

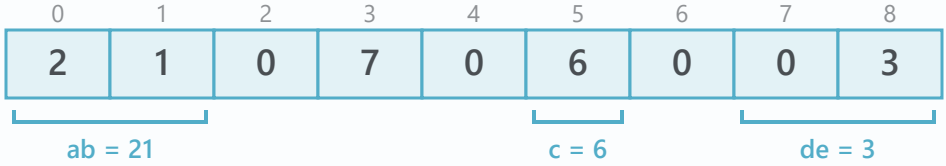
Adım 13 / 14

```
1 // s: "ab070c0de" biçiminde okul numarası
2 public int calculateHash(String s) {
3   int ab = Integer.parseInt(s.substring(0, 2));
4   int c  = Integer.parseInt(s.substring(5, 6));
5   int de = Integer.parseInt(s.substring(7, 9));
6   int yil   = (ab - 20) * 100;   // yıl bloğu
▶   int bolum = (c - 6) * 500;     // bölüm: 0/500
8   return (yil + bolum + de) % 1000;
9 }
```

DEĞİŞKENLER

`s = "210706003"` `ab = 21` `c = 6` `de = 3` `yil = 100` `bolum = 0`

Kod İzleme: `calculateHash()`



`return (100 + 0 + 3) % 1000` = **103**.

Adım 14 / 14

```
1 // s: "ab070c0de" biçiminde okul numarası
2 public int calculateHash(String s) {
3   int ab = Integer.parseInt(s.substring(0, 2));
4   int c  = Integer.parseInt(s.substring(5, 6));
5   int de = Integer.parseInt(s.substring(7, 9));
6   int yil   = (ab - 20) * 100;   // yıl bloğu
7   int bolum = (c - 6) * 500;     // bölüm: 0/500
▶   return (yil + bolum + de) % 1000;
9 }
```

DEĞİŞKENLER

`s = "210706003"` `ab = 21` `c = 6` `de = 3` `yil = 100` `bolum = 0`

ÖZET

Özet

Eşleme (Map) ADT

Tekil anahtar → değer. `put`, `get`, `remove`; aynı anahtarla put = güncelleme.

Hash fonksiyonu

hash kodu + sıkıştırma (`mod N`, N asal). Belirleyici, hızlı, düzgün dağılımlı.

Ayrı zincirleme

Her kova bir bağlı liste. Başa ekleme; maliyet O(1 + λ). λ > 1 olabilir.

Açık adresleme

Doğrusal (birincil kümelenme), karesel (ikincil kümelenme), çift hash. Silmede "silindi" işareti.

Yük faktörü λ = n/N

Eşik aşılınca (≈ 0,5–0,75) tablo ~2 katına büyütülür, tüm anahtarlar yeniden hash'lenir.

Doğruluk ve karmaşıklık

Eşit anahtarların hash kodu aynı olmalıdır. Ortalama O(1), en kötü O(n); yeniden boyutlandırma amortize O(1).

ÖZET

Etkileşimli Simülatörler

Hash Tablosu Simülatörü →

Ayrı zincirleme, adım adım hash hesabı, çakışmaların görselleştirilmesi ve otomatik yeniden boyutlandırma.

Eşleme Simülatörü →

set, get, has, delete işlemleri; anahtarlar üzerinde gezinme ve ekleme sırasının izlenmesi.

Açık Adresleme Simülatörü →

Doğrusal, karesel yoklama ve çift hash; yoklama dizisi adım adım, silindi işareti (*X*) ve yük faktörü.

ÖZET

Sık Yapılan Hatalar

`Math.abs(h) % N`

`Math.abs(Integer.MIN_VALUE)` negatiftir → dizi sınırı hatası. `Math.floorMod(h, N)` kullanın.

Yalnız `equals` ezmek

Eşit anahtarlar farklı kovaya düşer, `get` null döner. İkisi birlikte ezilir.

Değiştirilebilir anahtar

Tabloya girdikten sonra hash alanı değişen anahtar yanlış kovada kalır: bulunamaz.

`Integer` ile `==`

−128…127 dışındaki kutulu sayılar ayrı nesnelerdir; `==` false verir, `equals` kullanın.

Açık adreslemede `null` ile silmek

Yoklama zinciri kopar; "silindi" (DEFUNCT) işareti gerekir.

Sıra beklemek

`HashMap` gezinme sırası ekleme sırası değildir: `LinkedHashMap` ya da `TreeMap`.

ÖZET

Örnek Problemler

1. N = 7, h(k) = k mod 7, **doğrusal yoklama**: 10, 17, 24, 3, 5 sırayla eklenirse son tablo nedir?
2. Aynı anahtarları **karesel yoklama** ile ekleyin.
3. Zincirlemeli, N = 10 kovalı bir tabloda 25 çift var. λ nedir? Başarısız bir arama ortalama kaç düğüm inceler?
4. Açık adreslemede silinen yuvayı neden doğrudan `null` yapamayız?
5. 230706015 numarası için hash(ab070c0de) değerini hesaplayın.
6. 0, 12, 24, 36 anahtarları N = 12 ve N = 11 ile `k mod N` kullanıldığında hangi kovalara düşer?

▶ **Cevapları göster**

ÖZET

Alıştırmalar

1. N = 11, h(k) = k mod 11, $h_2(k) = 7 - (k \bmod 7)$ ile **çift hash**: 14, 25, 36, 3 hangi yuvalara yerleşir?
2. N = 5 zincirli tablodaki 12, 7, 20, 33 anahtarları N = 11'e büyütülünce hangi kovalara gider?
3. Doğrusal yoklamada λ = 0,8 iken başarısız bir arama ortalama kaç yoklama yapar?
4. Bir kelime dizisindeki her kelimenin kaç kez geçtiğini `HashMap` ile sayan döngüyü yazın.
5. `hashCode()` her nesne için 42 döndürürse tablo nasıl davranır?
6. `indis(32, 16)` ve `indis(65568, 16)` nedir? Karıştırma olmasa?

▶ **Cevap ve ipuçları**

Kaynaklar

- Goodrich, Tamassia ve Goldwasser, *Data Structures and Algorithms in Java*, 6. baskı, 2014.
- Cormen, Leiserson, Rivest ve Stein, *Introduction to Algorithms*, 4. baskı, 2022.
- Oracle, Java SE 17 `Map`, `HashMap` ve `HashSet` API belgeleri.

Son

Bölüm 12: Eşlemeler ve Hash Tabloları

Sonraki sunum: Bölüm 13: Kümeler ve Ayrık Kümeler · Çalışma soruları · Simülatörler: Hash Tablosu Simülatörü, Eşleme Simülatörü, Açık Adresleme Simülatörü · Terim sözlüğü