Bölüm 10: Arama Ağaçları

Veri Yapıları

İkili arama ağacı | Arama ve ekleme | Silme | O(h) analizi | Denge ve döndürme

AVL ağaçları | Kırmızı-siyah ağaçlar | Splay ağaçları | B-ağacı ve B+ ağacı

Sercan KÜLCÜ · Giresun Üniversitesi · Bilgisayar Mühendisliği

GİRİŞ

Öğrenme Çıktıları

BST işlemleri

Arama, ekleme ve silme işlemlerini ikili arama ağacında uygulamak.

Denge gereksinimi

Ağaç yüksekliğinin işlem maliyetlerine etkisini açıklamak.

Dengeleme

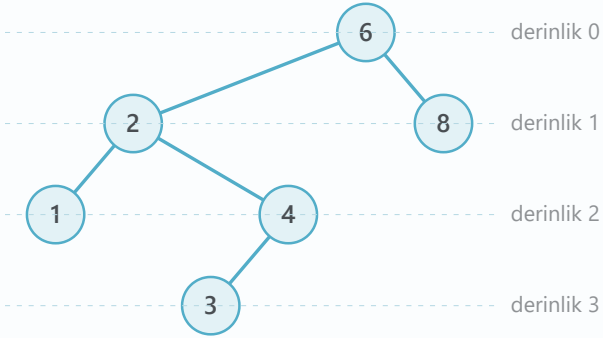
AVL döndürmelerini ve kırmızı-siyah ağaç kurallarını yorumlamak.

Karşılaştırma

BST, AVL, kırmızı-siyah, splay ve B-ağaçlarını kullanım amacına göre seçmek.

Arama Ağacı (Search Tree)

- Her düğüm bir öğeyi temsil eder; her öğenin bir **anahtarı** (key) vardır.
- Öğeler anahtarlarına göre **sıralı bir düzende** saklanır.
- Arama, kökten başlayıp her düğümde tek bir dala inerek yapılır.
- Maliyet, ağacın **yüksekliği h** ile orantılıdır: **O(h)**.
- n düğümlü ağaçta h, en iyi ≈ $\log_2 n$, en kötü n − 1 olur.



Arama ağaçlarının bütün hikâyesi: işlemler O(h) sürer; o hâlde **h'yi küçük tutmak** (dengeli ağaç) işlemleri O(log n)'e indirir.

Temel İşlemler

Arama (Search)

Verilen anahtar ağaçta varsa true, yoksa false döndürür.

```
ara(kok, 5) → true
```

Ekleme (Insertion)

Yeni öğeyi sıralama düzenini bozmadan uygun yaprak konumuna ekler.

```
ekle(kok, 7)
```

Silme (Deletion)

Verilen anahtara sahip öğeyi çıkarır ve düzeni korur.

```
sil(kok, 40)
```

Gezinme (Traversal)

Öğeleri belirli bir sırayla gezinir; kök ortada (inorder) gezinme anahtarları sıralı verir.

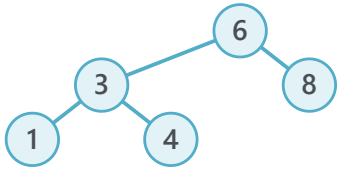
```
inorder(kok) → 1 2 3 4 6 8
```

Ek olarak: en küçük / en büyük anahtar, bir anahtarın ardılı / öncülü, aralık sorguları. Hepsi O(h).

Arama Ağacı Türleri

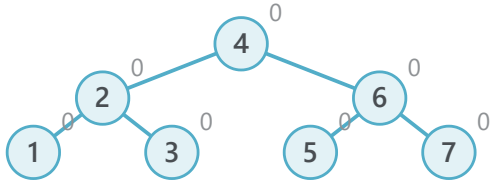
İkili Arama Ağacı (BST)

Her düğümün anahtarı, sol alt ağaçtaki anahtarlardan büyük, sağ alt ağaçtakilerden küçüktür. Denge garantisi yoktur.



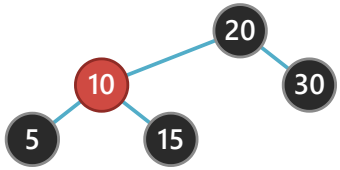
AVL Ağacı

Her düğümde iki alt ağacın yükseklik farkı (denge faktörü) en fazla 1'dir. Döndürmelerle dengelenir.



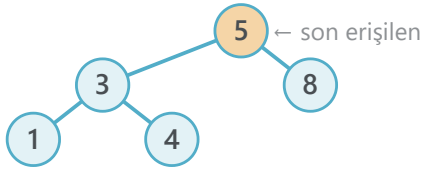
Kırmızı-Siyah Ağaç (Red-Black)

Düğüm renkleri ve kurallarıyla yüksekliği en fazla $2 \cdot \log_2(n+1)$ ile sınırlar.



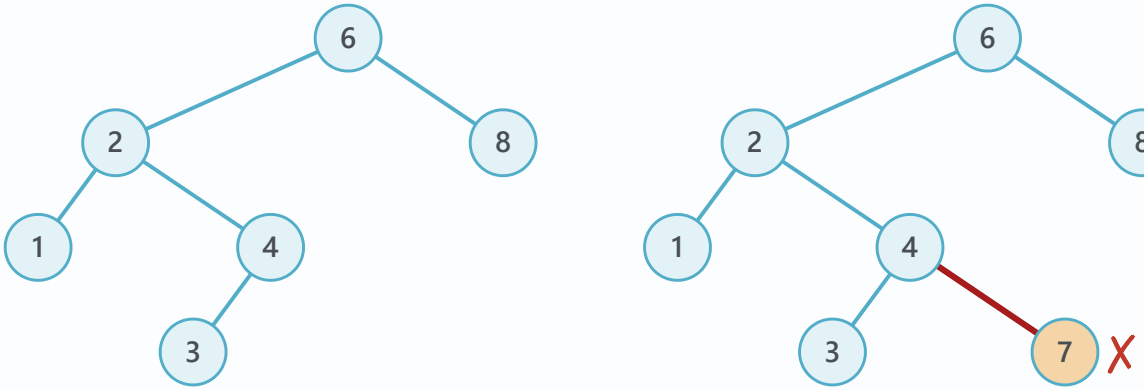
Splay Ağacı

Erişilen düğümü döndürmelerle köke taşır; işlem başına amortize O(log n).



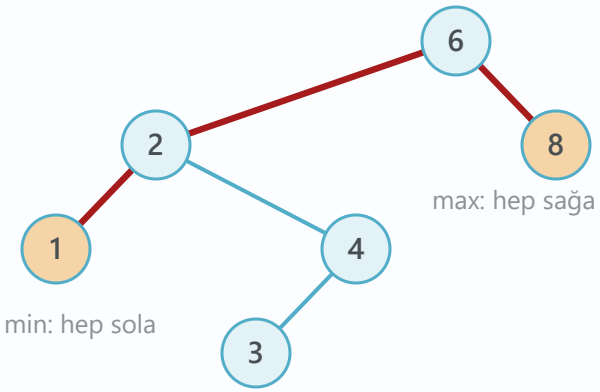
İkili Arama Ağacı (Binary Search Tree)

Her düğüm x için: **sol alt ağaçtaki tüm anahtarlar < x.anahtar < sağ alt ağaçtaki tüm anahtarlar**.



- Özellik yalnızca çocuklar için değil, **bütün alt ağaç** için geçerlidir: sağdaki ağaçta 7 > 4 doğru, ama 7, 6'nın solunda olamaz.
- Tekrarlayan anahtarlar ya reddedilir ya da tutarlı biçimde tek tarafa (ör. sağa) konur.

En Küçük, En Büyük ve Sıralı Gezinme



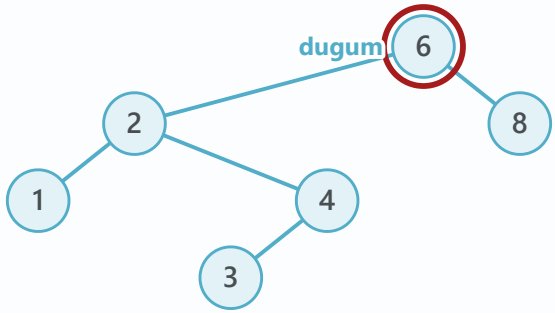
Kök ortada (sol → kök → sağ) gezinme anahtarları **sıralı** verir:
1 2 3 4 6 8

```
// Ön koşul: dugum != null
IkiliAgacDugum<E> minDeger(
        IkiliAgacDugum<E> dugum) {
    while (dugum.solCocuk != null)
        dugum = dugum.solCocuk; // sola in
    return dugum;  // en küçük
}
// maxDeger: aynısı, sagCocuk ile

void inorder(IkiliAgacDugum<E> d) {
    if (d == null) return;
    inorder(d.solCocuk);  // sol
    System.out.print(d.veri + " ");
    inorder(d.sagCocuk);  // sağ
}
```

Kod İzleme: `minDeger` ve `maxDeger`

```
▶ IkiliAgacDugum<E> minDeger(
2         IkiliAgacDugum<E> dugum) {
3     while (dugum.solCocuk != null)
4         dugum = dugum.solCocuk; // sola in
5     return dugum;  // en küçük
6 }
7 IkiliAgacDugum<E> maxDeger(
8         IkiliAgacDugum<E> dugum) {
9     while (dugum.sagCocuk != null)
10        dugum = dugum.sagCocuk; // sağa in
11    return dugum;  // en büyük
12 }
```



`minDeger(kok)` : dugum kökten (6) başlar. Ön koşul: ağaç boş değil.

Adım 1 / 12

DEĞİŞKENLER

`dugum = düğüm(6)`

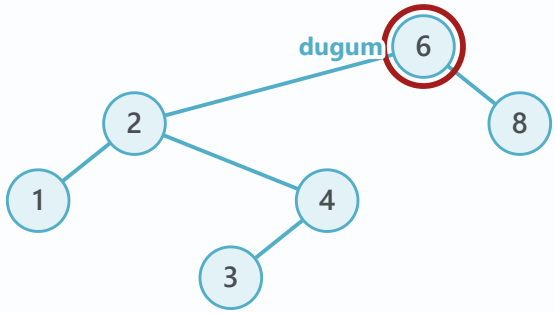
ÇAĞRI YIĞINI

`minDeger(6)`

`main()`

Kod İzleme: `minDeger` ve `maxDeger`

```
1  IkiliAgacDugum<E> minDeger(
2          IkiliAgacDugum<E> dugum) {
▶      while (dugum.solCocuk != null)
4          dugum = dugum.solCocuk; // sola in
5      return dugum;  // en küçük
6  }
7  IkiliAgacDugum<E> maxDeger(
8          IkiliAgacDugum<E> dugum) {
9      while (dugum.sagCocuk != null)
10         dugum = dugum.sagCocuk; // sağa in
11     return dugum;  // en büyük
12 }
```



dugum.solCocuk = 2 ≠ null → döngü devam.

Adım 2 / 12

DEĞİŞKENLER

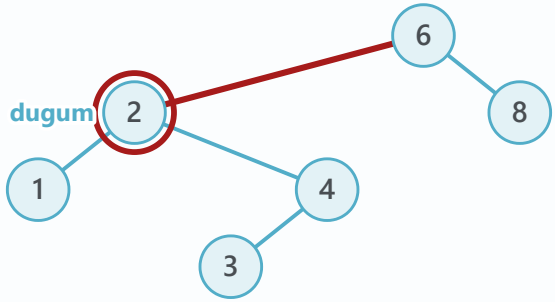
`dugum` = **düğüm(6)** | `dugum.solCocuk` = **düğüm(2)**

ÇAĞRI YIĞINI

- **minDeger(6)**
- main()

Kod İzleme: `minDeger` ve `maxDeger`

```
1  IkiliAgacDugum<E> minDeger(
2          IkiliAgacDugum<E> dugum) {
3      while (dugum.solCocuk != null)
▶          dugum = dugum.solCocuk; // sola in
5      return dugum;  // en küçük
6  }
7  IkiliAgacDugum<E> maxDeger(
8          IkiliAgacDugum<E> dugum) {
9      while (dugum.sagCocuk != null)
10         dugum = dugum.sagCocuk; // sağa in
11     return dugum;  // en büyük
12 }
```



dugum = dugum.solCocuk → 2. Sola iniyoruz.

DEĞİŞKENLER

`dugum = düğüm(2)`

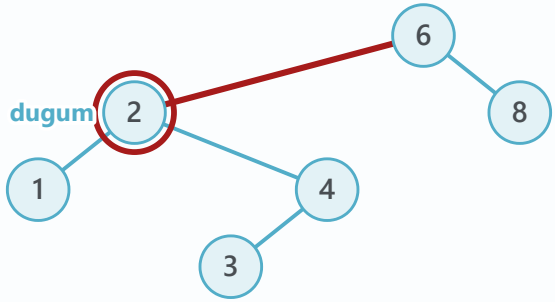
ÇAĞRI YIĞINI

`minDeger(6)`

`main()`

Kod İzleme: `minDeger` ve `maxDeger`

```
1  IkiliAgacDugum<E> minDeger(
2          IkiliAgacDugum<E> dugum) {
▶      while (dugum.solCocuk != null)
4          dugum = dugum.solCocuk; // sola in
5      return dugum;  // en küçük
6  }
7  IkiliAgacDugum<E> maxDeger(
8          IkiliAgacDugum<E> dugum) {
9      while (dugum.sagCocuk != null)
10         dugum = dugum.sagCocuk; // sağa in
11     return dugum;  // en büyük
12 }
```



dugum.solCocuk = 1 ≠ null → döngü devam.

Adım 4 / 12

DEĞİŞKENLER

`dugum = düğüm(2)` `dugum.solCocuk = düğüm(1)`

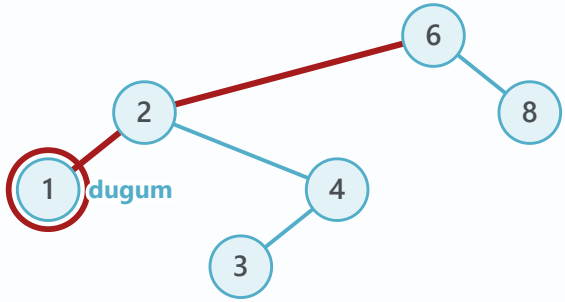
ÇAĞRI YIĞINI

`minDeger(6)`

`main()`

Kod İzleme: `minDeger` ve `maxDeger`

```
1  IkiliAgacDugum<E> minDeger(
2          IkiliAgacDugum<E> dugum) {
3      while (dugum.solCocuk != null)
▶          dugum = dugum.solCocuk; // sola in
5      return dugum;  // en küçük
6  }
7  IkiliAgacDugum<E> maxDeger(
8          IkiliAgacDugum<E> dugum) {
9      while (dugum.sagCocuk != null)
10         dugum = dugum.sagCocuk; // sağa in
11     return dugum;  // en büyük
12 }
```



dugum = dugum.solCocuk → 1. Sola iniyoruz.

Adım 5 / 12

DEĞİŞKENLER

`dugum = düğüm(1)`

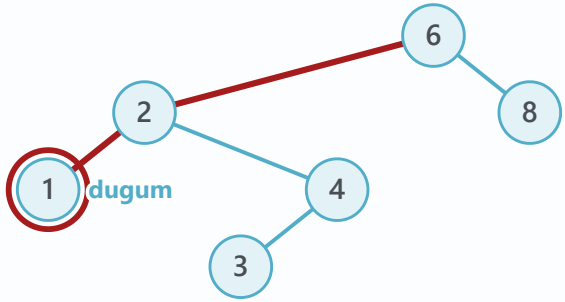
ÇAĞRI YIĞINI

`minDeger(6)`

`main()`

Kod İzleme: `minDeger` ve `maxDeger`

```
1  IkiliAgacDugum<E> minDeger(
2          IkiliAgacDugum<E> dugum) {
3      while (dugum.solCocuk != null)
4          dugum = dugum.solCocuk; // sola in
5      return dugum;  // en küçük
6  }
7  IkiliAgacDugum<E> maxDeger(
8          IkiliAgacDugum<E> dugum) {
9      while (dugum.sagCocuk != null)
10         dugum = dugum.sagCocuk; // sağa in
11     return dugum;  // en büyük
12 }
```



dugum.solCocuk == null → döngü biter.
1'in sol çocuğu yok.

DEĞİŞKENLER

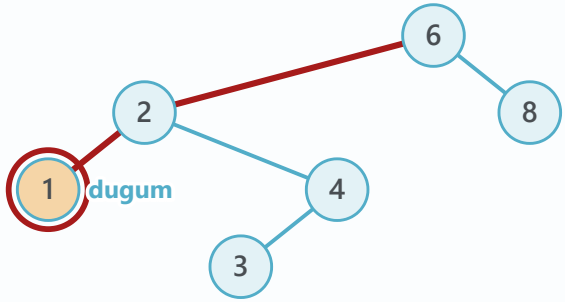
`dugum = düğüm(1)` `dugum.solCocuk = null`

ÇAĞRI YIĞINI

`minDeger(6)`
`main()`

Kod İzleme: `minDeger` ve `maxDeger`

```
1  IkiliAgacDugum<E> minDeger(
2          IkiliAgacDugum<E> dugum) {
3      while (dugum.solCocuk != null)
4          dugum = dugum.solCocuk; // sola in
▶      return dugum;  // en küçük
6  }
7  IkiliAgacDugum<E> maxDeger(
8          IkiliAgacDugum<E> dugum) {
9      while (dugum.sagCocuk != null)
10         dugum = dugum.sagCocuk; // sağa in
11     return dugum;  // en büyük
12 }
```



return 1: ağacın en küçük anahtarı. (3 daha derinde ama 4'ün solunda; en küçük hep en soldadır.)

Adım 7 / 12

DEĞİŞKENLER

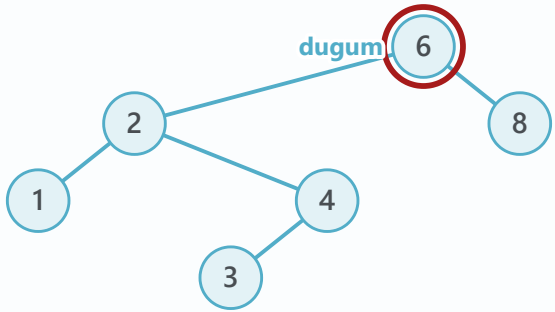
`dugum = düğüm(1)` `sonuç = 1`

ÇAĞRI YIĞINI

`main()`

Kod İzleme: `minDeger` ve `maxDeger`

```
1  IkiliAgacDugum<E> minDeger(
2          IkiliAgacDugum<E> dugum) {
3      while (dugum.solCocuk != null)
4          dugum = dugum.solCocuk; // sola in
5      return dugum;  // en küçük
6  }
▶  IkiliAgacDugum<E> maxDeger(
8          IkiliAgacDugum<E> dugum) {
9      while (dugum.sagCocuk != null)
10         dugum = dugum.sagCocuk; // sağa in
11     return dugum;  // en büyük
12 }
```



`maxDeger(kok)` : dugum kökten (6) başlar. Ön koşul: ağaç boş değil.

Adım 8 / 12

DEĞİŞKENLER

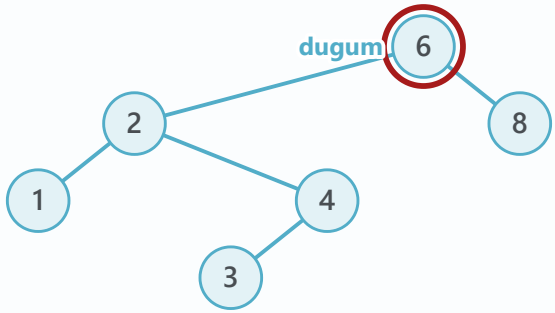
`dugum = düğüm(6)`

ÇAĞRI YIĞINI

- `maxDeger(6)`
- `main()`

Kod İzleme: `minDeger` ve `maxDeger`

```
1  IkiliAgacDugum<E> minDeger(
2          IkiliAgacDugum<E> dugum) {
3      while (dugum.solCocuk != null)
4          dugum = dugum.solCocuk; // sola in
5      return dugum;  // en küçük
6  }
7  IkiliAgacDugum<E> maxDeger(
8          IkiliAgacDugum<E> dugum) {
▶      while (dugum.sagCocuk != null)
10         dugum = dugum.sagCocuk; // sağa in
11     return dugum;  // en büyük
12 }
```



dugum.sagCocuk = 8 ≠ null → döngü devam.

Adım 9 / 12

DEĞİŞKENLER

`dugum = düğüm(6)` `dugum.sagCocuk = düğüm(8)`

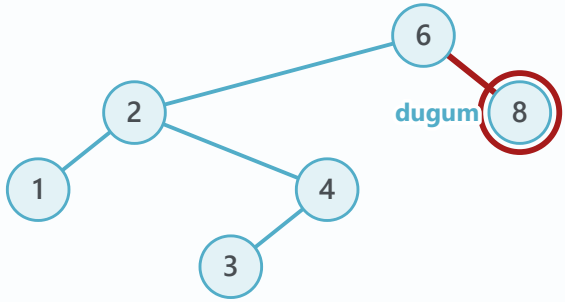
ÇAĞRI YIĞINI

`maxDeger(6)`

`main()`

Kod İzleme: `minDeger` ve `maxDeger`

```
1  IkiliAgacDugum<E> minDeger(
2          IkiliAgacDugum<E> dugum) {
3      while (dugum.solCocuk != null)
4          dugum = dugum.solCocuk; // sola in
5      return dugum;  // en küçük
6  }
7  IkiliAgacDugum<E> maxDeger(
8          IkiliAgacDugum<E> dugum) {
9      while (dugum.sagCocuk != null)
▶          dugum = dugum.sagCocuk; // sağa in
11     return dugum;  // en büyük
12 }
```



dugum = dugum.sagCocuk → 8. Sağa iniyoruz.

Adım 10 / 12

DEĞİŞKENLER

`dugum = düğüm(8)`

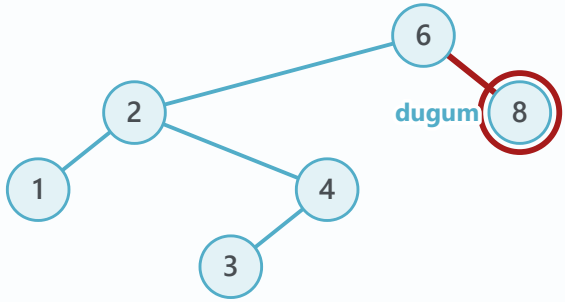
ÇAĞRI YIĞINI

`maxDeger(6)`

`main()`

Kod İzleme: `minDeger` ve `maxDeger`

```
1  IkiliAgacDugum<E> minDeger(
2          IkiliAgacDugum<E> dugum) {
3      while (dugum.solCocuk != null)
4          dugum = dugum.solCocuk; // sola in
5      return dugum;  // en küçük
6  }
7  IkiliAgacDugum<E> maxDeger(
8          IkiliAgacDugum<E> dugum) {
▶      while (dugum.sagCocuk != null)
10         dugum = dugum.sagCocuk; // sağa in
11     return dugum;  // en büyük
12 }
```



dugum.sagCocuk == null → döngü biter. 8'in sağ çocuğu yok.

Adım 11 / 12

DEĞİŞKENLER

`dugum = düğüm(8)` `dugum.sagCocuk = null`

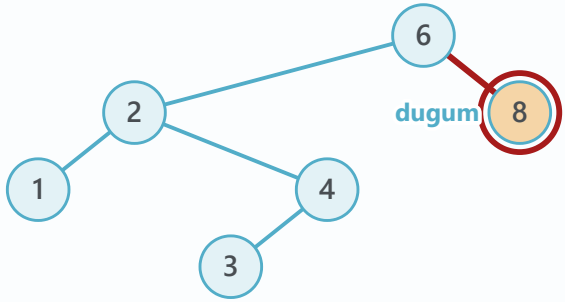
ÇAĞRI YIĞINI

`maxDeger(6)`

`main()`

Kod İzleme: `minDeger` ve `maxDeger`

```
1  IkiliAgacDugum<E> minDeger(
2          IkiliAgacDugum<E> dugum) {
3      while (dugum.solCocuk != null)
4          dugum = dugum.solCocuk; // sola in
5      return dugum;  // en küçük
6  }
7  IkiliAgacDugum<E> maxDeger(
8          IkiliAgacDugum<E> dugum) {
9      while (dugum.sagCocuk != null)
10         dugum = dugum.sagCocuk; // sağa in
▶      return dugum;  // en büyük
12 }
```



return 8: ağacın en büyük anahtarı. Karşılaştırma yok, yalnızca bağ takibi: **O(h)**.

Adım 12 / 12

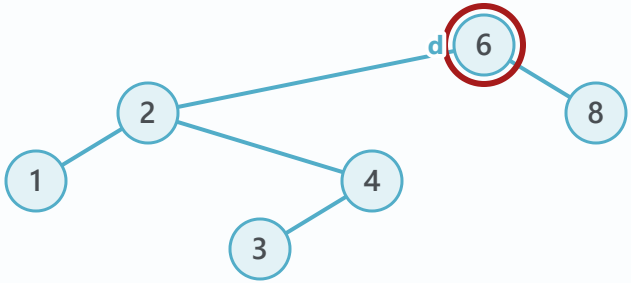
DEĞİŞKENLER

`dugum = düğüm(8)` `sonuç = 8`

ÇAĞRI YIĞINI

`main()`

Kod İzleme: `inorder(kok)`



`inorder(6)` : d ≠ null. Önce **sol** alt ağaç: `inorder(2)` çağrılır; 6 bekler.

Adım 1 / 20

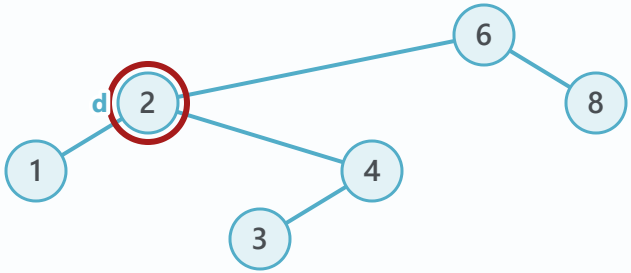
```
1 void inorder(IkiliAgacDugum<E> d) {
▶     if (d == null) return;
▶     inorder(d.solCocuk);  // sol
4     System.out.print(d.veri + " ");
5     inorder(d.sagCocuk);  // sağ
6 }
```

ÇAĞRI YIĞINI

- `inorder(6)`
- `main()`

ÇIKTI

Kod İzleme: `inorder(kok)`



`inorder(2)` : d ≠ null. Önce **sol** alt ağaç: `inorder(1)` çağrılır; 2 bekler.

Adım 2 / 20

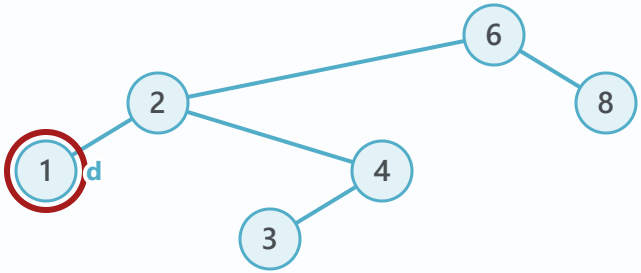
```
1 void inorder(IkiliAgacDugum<E> d) {
▶     if (d == null) return;
▶     inorder(d.solCocuk);  // sol
4     System.out.print(d.veri + " ");
5     inorder(d.sagCocuk);  // sağ
6 }
```

ÇAĞRI YIĞINI

- **`inorder(2)`**
- `inorder(6)`
- `main()`

ÇIKTI

Kod İzleme: `inorder(kok)`



`inorder(1)` : sol çocuk yok → `inorder(null)` satır 2'de hemen döner.

Adım 3 / 20

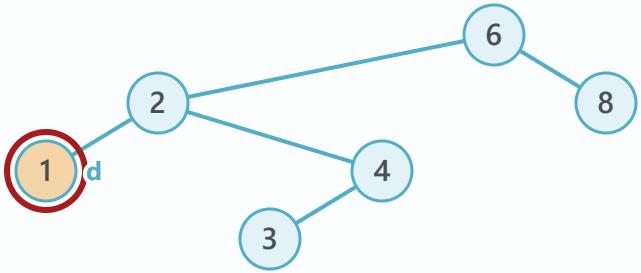
```
1 void inorder(IkiliAgacDugum<E> d) {
▶     if (d == null) return;
▶     inorder(d.solCocuk);  // sol
4     System.out.print(d.veri + " ");
5     inorder(d.sagCocuk);  // sağ
6 }
```

ÇAĞRI YIĞINI

- **`inorder(1)`**
- `inorder(2)`
- `inorder(6)`
- `main()`

ÇIKTI

Kod İzleme: `inorder(kok)`



Sol taraf bitti → **1 yazdırılır**. (Sol alt ağaçtaki her anahtar 1'den küçüktü.)

Adım 4 / 20

```
1 void inorder(IkiliAgacDugum<E> d) {
2     if (d == null) return;
3     inorder(d.solCocuk);  // sol
▶     System.out.print(d.veri + " ");
5     inorder(d.sagCocuk);  // sağ
6 }
```

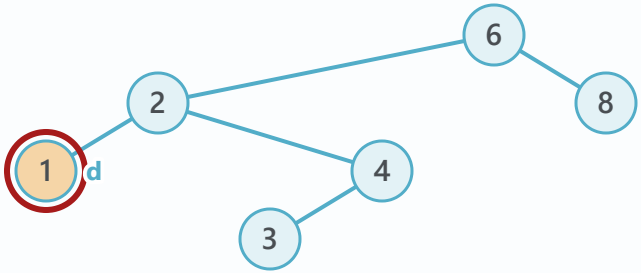
ÇAĞRI YIĞINI

- **`inorder(1)`**
- `inorder(2)`
- `inorder(6)`
- `main()`

ÇIKTI

```
1
```

Kod İzleme: `inorder(kok)`



Sağ çocuk yok → çerçeve kapanır; çağıran `inorder(2)` kaldığı yerden devam eder.

Adım 5 / 20

```
1 void inorder(IkiliAgacDugum<E> d) {
2     if (d == null) return;
3     inorder(d.solCocuk);  // sol
4     System.out.print(d.veri + " ");
▶     inorder(d.sagCocuk);  // sağ
▶ }
```

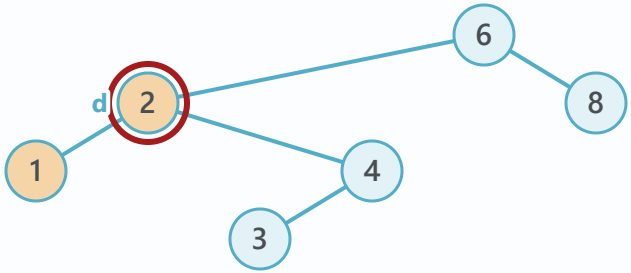
ÇAĞRI YIĞINI

- **`inorder(1)`**
- `inorder(2)`
- `inorder(6)`
- `main()`

ÇIKTI

```
1
```

Kod İzleme: `inorder(kok)`



Sol taraf bitti → **2 yazdırılır**. (Sol alt ağaçtaki her anahtar 2'den küçüktü.)

Adım 6 / 20

```
1 void inorder(IkiliAgacDugum<E> d) {
2     if (d == null) return;
3     inorder(d.solCocuk);  // sol
▶     System.out.print(d.veri + " ");
5     inorder(d.sagCocuk);  // sağ
6 }
```

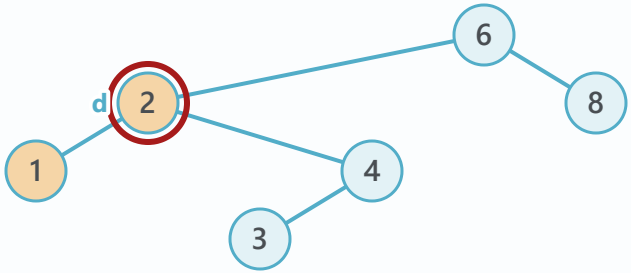
ÇAĞRI YIĞINI

- **`inorder(2)`**
- `inorder(6)`
- `main()`

ÇIKTI

```
1 2
```

Kod İzleme: `inorder(kok)`



Şimdi **sağ** alt ağaç: `inorder(4)` .

Adım 7 / 20

```
1 void inorder(IkiliAgacDugum<E> d) {
2     if (d == null) return;
3     inorder(d.solCocuk);  // sol
4     System.out.print(d.veri + " ");
▶     inorder(d.sagCocuk);  // sağ
6 }
```

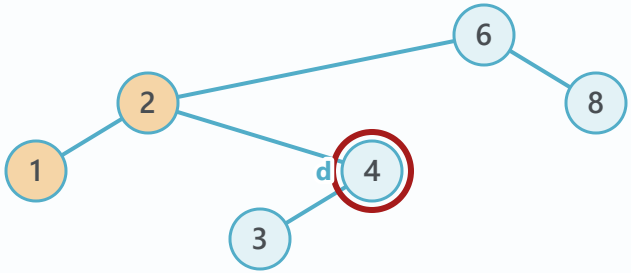
ÇAĞRI YIĞINI

- **`inorder(2)`**
- `inorder(6)`
- `main()`

ÇIKTI

```
1 2
```

Kod İzleme: `inorder(kok)`



`inorder(4)` : d ≠ null. Önce **sol** alt ağaç: `inorder(3)` çağrılır; 4 bekler.

Adım 8 / 20

```
1 void inorder(IkiliAgacDugum<E> d) {
▶       if (d == null) return;
▶       inorder(d.solCocuk);  // sol
4       System.out.print(d.veri + " ");
5       inorder(d.sagCocuk);  // sağ
6   }
```

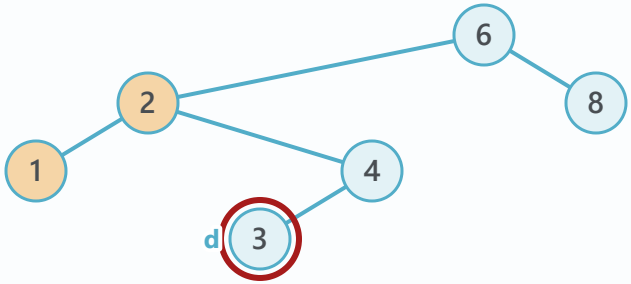
ÇAĞRI YIĞINI

- **`inorder(4)`**
- `inorder(2)`
- `inorder(6)`
- `main()`

ÇIKTI

```
1 2
```

Kod İzleme: `inorder(kok)`



`inorder(3)` : sol çocuk yok → `inorder(null)` satır 2'de hemen döner.

Adım 9 / 20

```
1 void inorder(IkiliAgacDugum<E> d) {
▶     if (d == null) return;
▶     inorder(d.solCocuk);  // sol
4     System.out.print(d.veri + " ");
5     inorder(d.sagCocuk);  // sağ
6 }
```

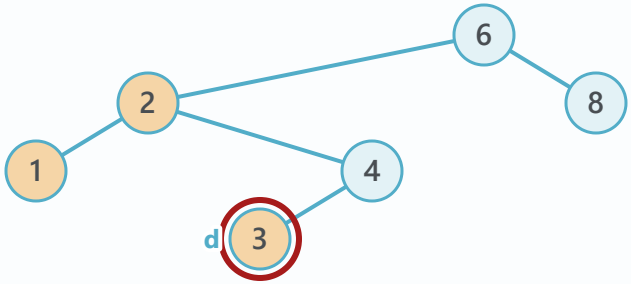
ÇAĞRI YIĞINI

- **`inorder(3)`**
- `inorder(4)`
- `inorder(2)`
- `inorder(6)`
- `main()`

ÇIKTI

```
1 2
```

Kod İzleme: `inorder(kok)`



Sol taraf bitti → **3 yazdırılır**. (Sol alt ağaçtaki her anahtar 3'ten küçüktü.)

Adım 10 / 20

```
1 void inorder(IkiliAgacDugum<E> d) {
2     if (d == null) return;
3     inorder(d.solCocuk);  // sol
▶     System.out.print(d.veri + " ");
5     inorder(d.sagCocuk);  // sağ
6 }
```

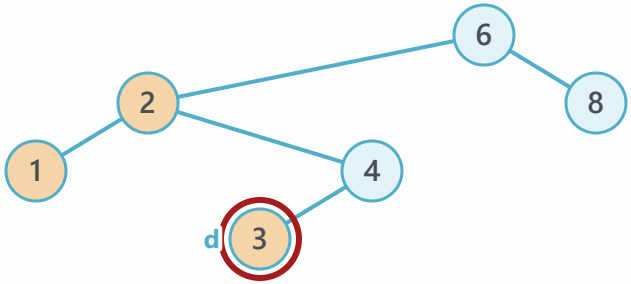
ÇAĞRI YIĞINI

- **`inorder(3)`**
- `inorder(4)`
- `inorder(2)`
- `inorder(6)`
- `main()`

ÇIKTI

```
1 2 3
```

Kod İzleme: `inorder(kok)`



Sağ çocuk yok → çerçeve kapanır; çağıran `inorder(4)` kaldığı yerden devam eder.

Adım 11 / 20

```
1 void inorder(IkiliAgacDugum<E> d) {
2     if (d == null) return;
3     inorder(d.solCocuk);  // sol
4     System.out.print(d.veri + " ");
▶     inorder(d.sagCocuk);  // sağ
▶ }
```

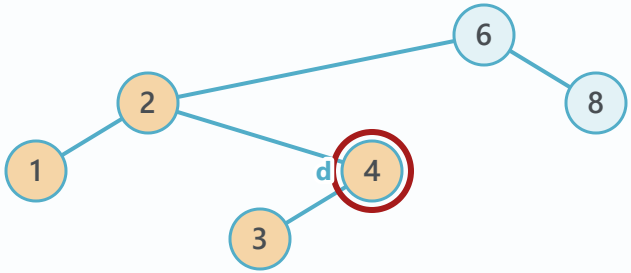
ÇAĞRI YIĞINI

- **`inorder(3)`**
- `inorder(4)`
- `inorder(2)`
- `inorder(6)`
- `main()`

ÇIKTI

```
1 2 3
```

Kod İzleme: `inorder(kok)`



Sol taraf bitti → **4 yazdırılır**. (Sol alt ağaçtaki her anahtar 4'ten küçüktü.)

Adım 12 / 20

```
1 void inorder(IkiliAgacDugum<E> d) {
2     if (d == null) return;
3     inorder(d.solCocuk);  // sol
▶     System.out.print(d.veri + " ");
5     inorder(d.sagCocuk);  // sağ
6 }
```

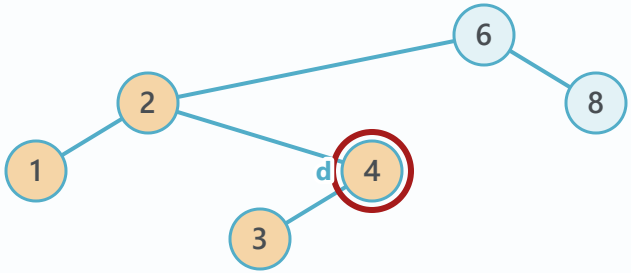
ÇAĞRI YIĞINI

- **`inorder(4)`**
- `inorder(2)`
- `inorder(6)`
- `main()`

ÇIKTI

```
1 2 3 4
```

Kod İzleme: `inorder(kok)`



Sağ çocuk yok → çerçeve kapanır; çağıran `inorder(2)` kaldığı yerden devam eder.

Adım 13 / 20

```
1 void inorder(IkiliAgacDugum<E> d) {
2     if (d == null) return;
3     inorder(d.solCocuk);  // sol
4     System.out.print(d.veri + " ");
▶     inorder(d.sagCocuk);  // sağ
▶ }
```

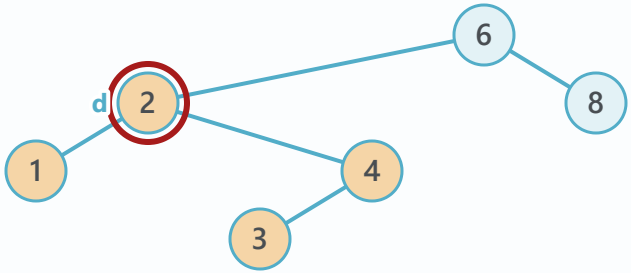
ÇAĞRI YIĞINI

- **`inorder(4)`**
- `inorder(2)`
- `inorder(6)`
- `main()`

ÇIKTI

```
1 2 3 4
```

Kod İzleme: `inorder(kok)`



2'nin sağ alt ağacı da bitti → `inorder(2)` kapanır; çağıran kaldığı yerden devam eder.

Adım 14 / 20

```
1 void inorder(IkiliAgacDugum<E> d) {
2     if (d == null) return;
3     inorder(d.solCocuk);  // sol
4     System.out.print(d.veri + " ");
▶     inorder(d.sagCocuk);  // sağ
▶ }
```

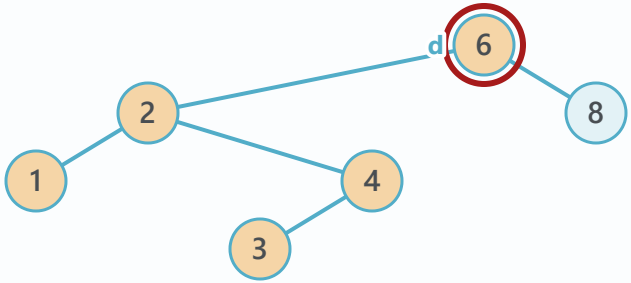
ÇAĞRI YIĞINI

- **`inorder(2)`**
- `inorder(6)`
- `main()`

ÇIKTI

```
1 2 3 4
```

Kod İzleme: `inorder(kok)`



Sol taraf bitti → **6 yazdırılır**. (Sol alt ağaçtaki her anahtar 6'dan küçüktü.)

Adım 15 / 20

```
1 void inorder(IkiliAgacDugum<E> d) {
2     if (d == null) return;
3     inorder(d.solCocuk);  // sol
▶     System.out.print(d.veri + " ");
5     inorder(d.sagCocuk);  // sağ
6 }
```

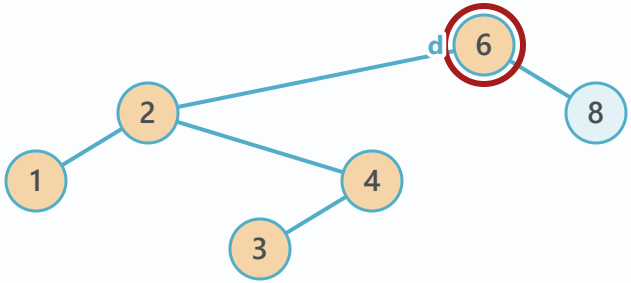
ÇAĞRI YIĞINI

- **`inorder(6)`**
- `main()`

ÇIKTI

```
1 2 3 4 6
```

Kod İzleme: `inorder(kok)`



Şimdi **sağ** alt ağaç: `inorder(8)` .

Adım 16 / 20

```
1 void inorder(IkiliAgacDugum<E> d) {
2     if (d == null) return;
3     inorder(d.solCocuk);  // sol
4     System.out.print(d.veri + " ");
▶     inorder(d.sagCocuk);  // sağ
6 }
```

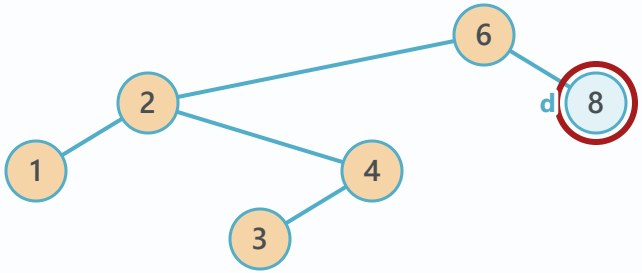
ÇAĞRI YIĞINI

- **`inorder(6)`**
- `main()`

ÇIKTI

```
1 2 3 4 6
```

Kod İzleme: `inorder(kok)`



`inorder(8)` : sol çocuk yok → `inorder(null)` satır 2'de hemen döner.

Adım 17 / 20

```
1 void inorder(IkiliAgacDugum<E> d) {
▶     if (d == null) return;
▶     inorder(d.solCocuk);  // sol
4     System.out.print(d.veri + " ");
5     inorder(d.sagCocuk);  // sağ
6 }
```

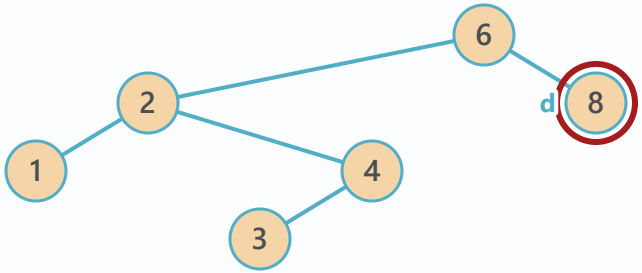
ÇAĞRI YIĞINI

- **`inorder(8)`**
- `inorder(6)`
- `main()`

ÇIKTI

```
1 2 3 4 6
```

Kod İzleme: `inorder(kok)`



Sol taraf bitti → **8 yazdırılır**. (Sol alt ağaçtaki her anahtar 8'den küçüktü.)

Adım 18 / 20

```
1 void inorder(IkiliAgacDugum<E> d) {
2     if (d == null) return;
3     inorder(d.solCocuk);  // sol
▶     System.out.print(d.veri + " ");
5     inorder(d.sagCocuk);  // sağ
6 }
```

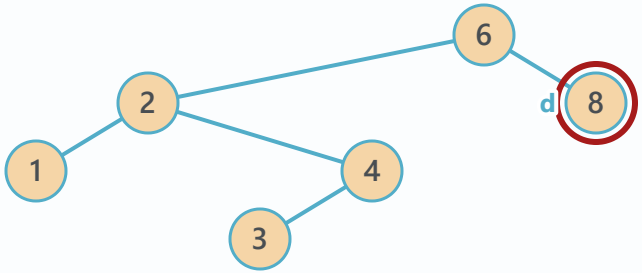
ÇAĞRI YIĞINI

- **`inorder(8)`**
- `inorder(6)`
- `main()`

ÇIKTI

```
1 2 3 4 6 8
```

Kod İzleme: `inorder(kok)`



Sağ çocuk yok → çerçeve kapanır; çağıran `inorder(6)` kaldığı yerden devam eder.

Adım 19 / 20

```
1 void inorder(IkiliAgacDugum<E> d) {
2     if (d == null) return;
3     inorder(d.solCocuk);  // sol
4     System.out.print(d.veri + " ");
▶     inorder(d.sagCocuk);  // sağ
▶ }
```

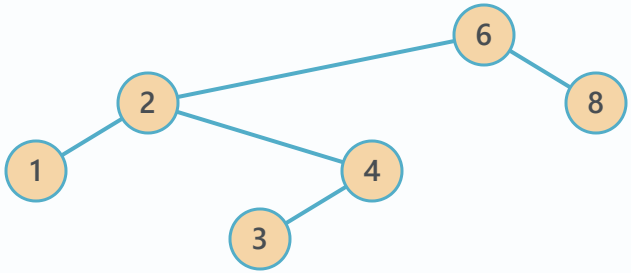
ÇAĞRI YIĞINI

- **`inorder(8)`**
- `inorder(6)`
- `main()`

ÇIKTI

```
1 2 3 4 6 8
```

Kod İzleme: `inorder(kok)`



Tüm çağrılar bitti; çıktı **sıralı**: 1 2 3 4 6 8. Her düğüm bir kez yazıldı → **O(n)**.

Adım 20 / 20

```
1 void inorder(IkiliAgacDugum<E> d) {
2     if (d == null) return;
3     inorder(d.solCocuk);  // sol
4     System.out.print(d.veri + " ");
5     inorder(d.sagCocuk);  // sağ
6 }
```

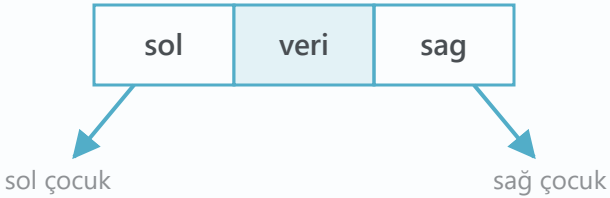
ÇAĞRI YIĞINI

`main()`

ÇIKTI

```
1 2 3 4 6 8
```

Ağaç Düğümü Yapısı



int anahtarlı

```
class AgacDugumu {
    int veri;
    AgacDugumu sol, sag;

    AgacDugumu(int veri) {
        this.veri = veri;
        this.sol = null;
        this.sag = null;
    }
}
```

generic

```
class IkiliAgacDugum<E extends Comparable<E>> {
    E veri;                      // nesne
    IkiliAgacDugum<E> solCocuk; // sol çocuk
    IkiliAgacDugum<E> sagCocuk; // sağ çocuk

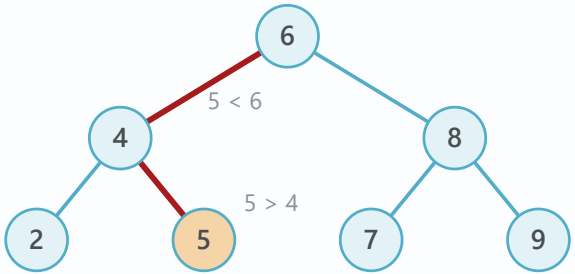
    IkiliAgacDugum(E veri) {
        this.veri = veri;
        // çocuklar başlangıçta null
    }
}
```

Anahtarlar `compareTo` ile karşılaştırılır; bu yüzden E, `Comparable<E>` olmalıdır.

BST İŞLEMLERİ

Öğe Arama

- Kökten başla; anahtar eşitse bulundu.
- Küçükse sol, büyükse sağ alt ağaçta devam et.
- null'a ulaşırsan anahtar ağaçta yoktur.




```
boolean ara(IkiliAgacDugum<E> kok, E veri) {
    if (kok == null)
        return false;              // yok
    int k = veri.compareTo(kok.veri);
    if (k == 0)
        return true;               // bulundu
    else if (k < 0)
        return ara(kok.solCocuk, veri);
    else
        return ara(kok.sagCocuk, veri);
}
```

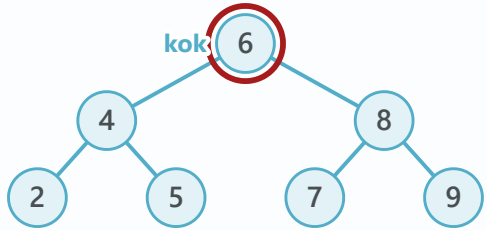
Nesneleri `==` ile değil `compareTo` (veya `equals`) ile karşılaştır: `==` yalnızca referansların aynı olup olmadığına bakar.

Kod İzleme: `ara(kok, 5)` ve `ara(kok, 10)`

```
1  boolean ara(IkiliAgacDugum<E> kok, E veri) {
▶      if (kok == null)
3          return false;            // yok
▶      int k = veri.compareTo(kok.veri);
5      if (k == 0)
6          return true;             // bulundu
7      else if (k < 0)
8          return ara(kok.solCocuk, veri);
9      else
10         return ara(kok.sagCocuk, veri);
11 }
```

ÇAĞRI YIĞINI

- **`ara(6, 5)`**
- `main()`



`ara(kok, 5)` çağrıldı; kok kökü (6) gösterir. kok ≠ null. **k = compareTo = -1** çünkü 5 < 6.

Adım 1 / 20

DEĞİŞKENLER

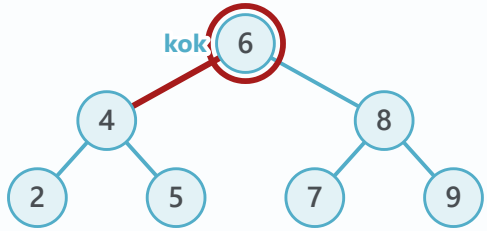
`kok` = **düğüm(6)** `veri` = **5** `k` = **−1**

Kod İzleme: `ara(kok, 5)` ve `ara(kok, 10)`

```
1  boolean ara(IkiliAgacDugum<E> kok, E veri) {
2      if (kok == null)
3          return false;            // yok
4      int k = veri.compareTo(kok.veri);
5      if (k == 0)
6          return true;             // bulundu
▶      else if (k < 0)
▶          return ara(kok.solCocuk, veri);
9      else
10         return ara(kok.sagCocuk, veri);
11 }
```

ÇAĞRI YIĞINI

- **`ara(6, 5)`**
- `main()`



k < 0 → sol alt ağaçta ara: `ara(kok.solCocuk, 5)` yeni bir çerçeve açar (kok = 4).

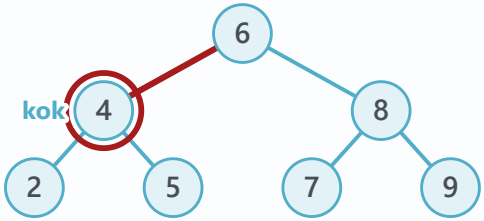
Adım 2 / 20

DEĞİŞKENLER

`kok` = **düğüm(6)** `veri` = **5** `k` = **−1**

Kod İzleme: `ara(kok, 5)` ve `ara(kok, 10)`

```
1  boolean ara(IkiliAgacDugum<E> kok, E veri) {
▶      if (kok == null)
3          return false;            // yok
▶      int k = veri.compareTo(kok.veri);
5      if (k == 0)
6          return true;             // bulundu
7      else if (k < 0)
8          return ara(kok.solCocuk, veri);
9      else
10         return ara(kok.sagCocuk, veri);
11 }
```



kok ≠ null. **k = compareTo = 1** çünkü 5 > 4.

Adım 3 / 20

ÇAĞRI YIĞINI

- **ara(4, 5)**
- ara(6, 5)
- main()

DEĞİŞKENLER

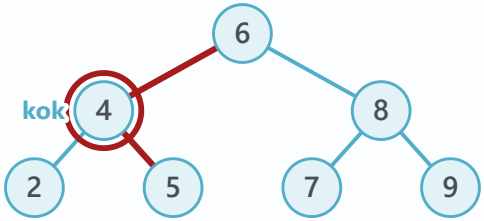
`kok = düğüm(4)` `veri = 5` `k = 1`

Kod İzleme: `ara(kok, 5)` ve `ara(kok, 10)`

```
1  boolean ara(IkiliAgacDugum<E> kok, E veri) {
2      if (kok == null)
3          return false;            // yok
4      int k = veri.compareTo(kok.veri);
5      if (k == 0)
6          return true;             // bulundu
7      else if (k < 0)
8          return ara(kok.solCocuk, veri);
▶      else
▶          return ara(kok.sagCocuk, veri);
11 }
```

ÇAĞRI YIĞINI

- **`ara(4, 5)`**
- `ara(6, 5)`
- `main()`



k > 0 → sağ alt ağaçta ara: `ara(kok.sagCocuk, 5)` yeni bir çerçeve açar (kok = 5).

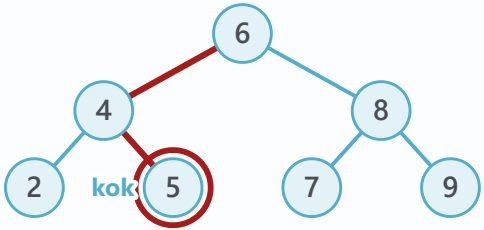
Adım 4 / 20

DEĞİŞKENLER

`kok` = **düğüm(4)** `veri` = **5** `k` = **1**

Kod İzleme: `ara(kok, 5)` ve `ara(kok, 10)`

```
1  boolean ara(IkiliAgacDugum<E> kok, E veri) {
▶      if (kok == null)
3          return false;            // yok
▶      int k = veri.compareTo(kok.veri);
5      if (k == 0)
6          return true;             // bulundu
7      else if (k < 0)
8          return ara(kok.solCocuk, veri);
9      else
10         return ara(kok.sagCocuk, veri);
11 }
```



kok ≠ null. **k = compareTo = 0** çünkü 5 = 5.

Adım 5 / 20

ÇAĞRI YIĞINI

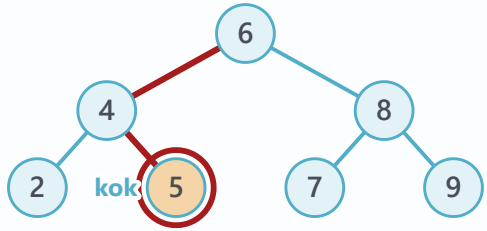
- **ara(5, 5)**
- ara(4, 5)
- ara(6, 5)
- main()

DEĞİŞKENLER

`kok = düğüm(5)` `veri = 5` `k = 0`

Kod İzleme: `ara(kok, 5)` ve `ara(kok, 10)`

```
1  boolean ara(IkiliAgacDugum<E> kok, E veri) {
2      if (kok == null)
3          return false;            // yok
4      int k = veri.compareTo(kok.veri);
▶      if (k == 0)
▶          return true;             // bulundu
7      else if (k < 0)
8          return ara(kok.solCocuk, veri);
9      else
10         return ara(kok.sagCocuk, veri);
11 }
```



k == 0 → anahtar bulundu: **return true**. Bu çerçeve kapanır.

Adım 6 / 20

ÇAĞRI YIĞINI

- **ara(5, 5)**
- ara(4, 5)
- ara(6, 5)
- main()

DEĞİŞKENLER

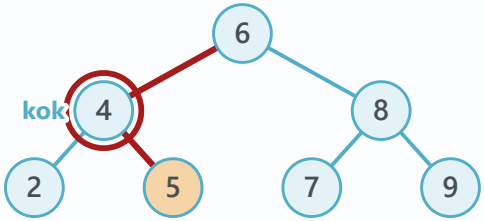
`kok` = **düğüm(5)** `veri` = **5** `k` = **0**

Kod İzleme: `ara(kok, 5)` ve `ara(kok, 10)`

```
1  boolean ara(IkiliAgacDugum<E> kok, E veri) {
2      if (kok == null)
3          return false;            // yok
4      int k = veri.compareTo(kok.veri);
5      if (k == 0)
6          return true;             // bulundu
7      else if (k < 0)
8          return ara(kok.solCocuk, veri);
9      else
▶          return ara(kok.sagCocuk, veri);
11 }
```

ÇAĞRI YIĞINI

- **ara(4, 5)**
- ara(6, 5)
- main()



`ara(5, 5)` çağrısı **true** döndürdü; 4'teki çerçeve bu değeri aynen **return** eder.

Adım 7 / 20

DEĞİŞKENLER

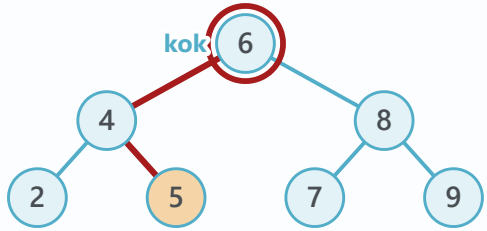
`kok = düğüm(4)` `veri = 5` `k = 1` `dönen = true`

Kod İzleme: `ara(kok, 5)` ve `ara(kok, 10)`

```
1  boolean ara(IkiliAgacDugum<E> kok, E veri) {
2      if (kok == null)
3          return false;            // yok
4      int k = veri.compareTo(kok.veri);
5      if (k == 0)
6          return true;             // bulundu
7      else if (k < 0)
▶          return ara(kok.solCocuk, veri);
9      else
10         return ara(kok.sagCocuk, veri);
11 }
```

ÇAĞRI YIĞINI

- **ara(6, 5)**
- main()



`ara(4, 5)` çağrısı **true** döndürdü; 6'daki çerçeve bu değeri aynen **return** eder.

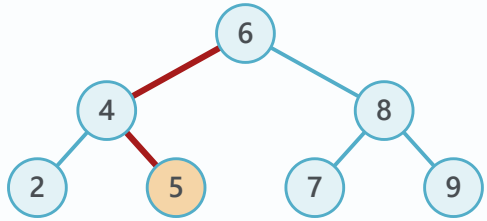
Adım 8 / 20

DEĞİŞKENLER

`kok = düğüm(6)` `veri = 5` `k = -1` `dönen = true`

Kod İzleme: `ara(kok, 5)` ve `ara(kok, 10)`

```
▶ boolean ara(IkiliAgacDugum<E> kok, E veri) {
2     if (kok == null)
3         return false;            // yok
4     int k = veri.compareTo(kok.veri);
5     if (k == 0)
6         return true;             // bulundu
7     else if (k < 0)
8         return ara(kok.solCocuk, veri);
9     else
10        return ara(kok.sagCocuk, veri);
11 }
```



main sonucu alır: **true**. 3 düğümle karşılaştırıldı = bulunan düğümün derinliği + 1.

Adım 9 / 20

ÇAĞRI YIĞINI

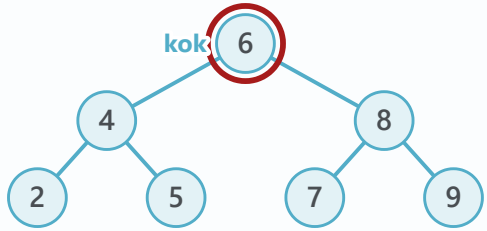
`main()`

DEĞİŞKENLER

`sonuç = true`

Kod İzleme: `ara(kok, 5)` ve `ara(kok, 10)`

```
1  boolean ara(IkiliAgacDugum<E> kok, E veri) {
▶      if (kok == null)
3          return false;            // yok
▶      int k = veri.compareTo(kok.veri);
5      if (k == 0)
6          return true;             // bulundu
7      else if (k < 0)
8          return ara(kok.solCocuk, veri);
9      else
10         return ara(kok.sagCocuk, veri);
11 }
```



`ara(kok, 10)` çağrıldı; kok kökü (6) gösterir. kok ≠ null. **k = compareTo = 1** çünkü 10 > 6.

Adım 10 / 20

ÇAĞRI YIĞINI

- **ara(6, 10)**
- main()

DEĞİŞKENLER

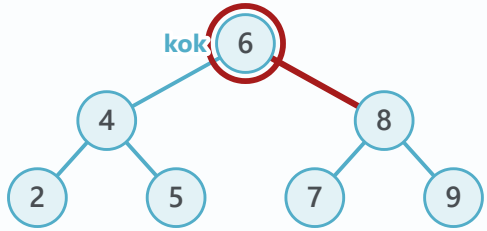
`kok` = **düğüm(6)** `veri` = **10** `k` = **1**

Kod İzleme: `ara(kok, 5)` ve `ara(kok, 10)`

```
1  boolean ara(IkiliAgacDugum<E> kok, E veri) {
2      if (kok == null)
3          return false;            // yok
4      int k = veri.compareTo(kok.veri);
5      if (k == 0)
6          return true;             // bulundu
7      else if (k < 0)
8          return ara(kok.solCocuk, veri);
▶      else
▶          return ara(kok.sagCocuk, veri);
11 }
```

ÇAĞRI YIĞINI

- **`ara(6, 10)`**
- `main()`



k > 0 → sağ alt ağaçta ara: `ara(kok.sagCocuk, 10)` yeni bir çerçeve açar (kok = 8).

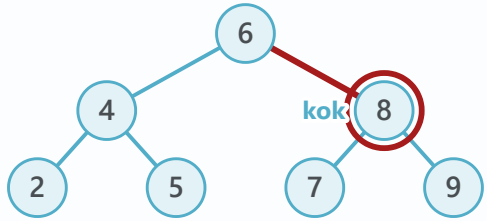
Adım 11 / 20

DEĞİŞKENLER

`kok` = **düğüm(6)** `veri` = **10** `k` = **1**

Kod İzleme: `ara(kok, 5)` ve `ara(kok, 10)`

```
1  boolean ara(IkiliAgacDugum<E> kok, E veri) {
▶      if (kok == null)
3          return false;            // yok
▶      int k = veri.compareTo(kok.veri);
5      if (k == 0)
6          return true;             // bulundu
7      else if (k < 0)
8          return ara(kok.solCocuk, veri);
9      else
10         return ara(kok.sagCocuk, veri);
11 }
```



kok ≠ null. **k = compareTo = 1** çünkü 10 > 8.

Adım 12 / 20

ÇAĞRI YIĞINI

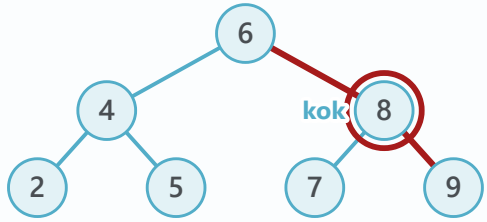
- **ara(8, 10)**
- ara(6, 10)
- main()

DEĞİŞKENLER

`kok` = **düğüm(8)** | `veri` = **10** | `k` = **1**

Kod İzleme: `ara(kok, 5)` ve `ara(kok, 10)`

```
1  boolean ara(IkiliAgacDugum<E> kok, E veri) {
2      if (kok == null)
3          return false;            // yok
4      int k = veri.compareTo(kok.veri);
5      if (k == 0)
6          return true;             // bulundu
7      else if (k < 0)
8          return ara(kok.solCocuk, veri);
▶      else
▶          return ara(kok.sagCocuk, veri);
11 }
```



k > 0 → sağ alt ağaçta ara: `ara(kok.sagCocuk, 10)` yeni bir çerçeve açar (kok = 9).

Adım 13 / 20

ÇAĞRI YIĞINI

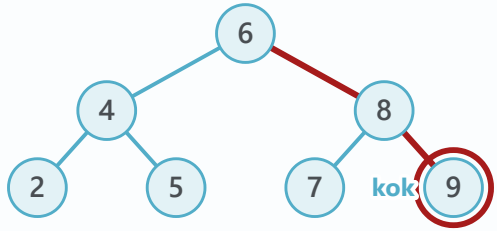
- **`ara(8, 10)`**
- `ara(6, 10)`
- `main()`

DEĞİŞKENLER

`kok` = **düğüm(8)** · `veri` = **10** · `k` = **1**

Kod İzleme: `ara(kok, 5)` ve `ara(kok, 10)`

```
1  boolean ara(IkiliAgacDugum<E> kok, E veri) {
▶      if (kok == null)
3          return false;            // yok
▶      int k = veri.compareTo(kok.veri);
5      if (k == 0)
6          return true;             // bulundu
7      else if (k < 0)
8          return ara(kok.solCocuk, veri);
9      else
10         return ara(kok.sagCocuk, veri);
11 }
```



kok ≠ null. **k = compareTo = 1** çünkü 10 > 9.

Adım 14 / 20

ÇAĞRI YIĞINI

- **ara(9, 10)**
- ara(8, 10)
- ara(6, 10)
- main()

DEĞİŞKENLER

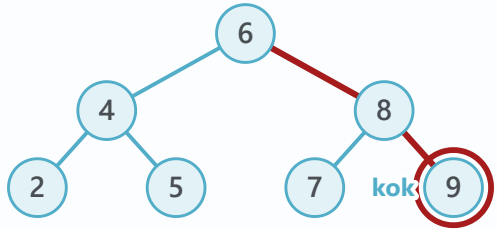
`kok` = **düğüm(9)** | `veri` = **10** | `k` = **1**

Kod İzleme: `ara(kok, 5)` ve `ara(kok, 10)`

```
1  boolean ara(IkiliAgacDugum<E> kok, E veri) {
2      if (kok == null)
3          return false;            // yok
4      int k = veri.compareTo(kok.veri);
5      if (k == 0)
6          return true;             // bulundu
7      else if (k < 0)
8          return ara(kok.solCocuk, veri);
▶      else
▶          return ara(kok.sagCocuk, veri);
11 }
```

ÇAĞRI YIĞINI

- **`ara(9, 10)`**
- `ara(8, 10)`
- `ara(6, 10)`
- `main()`



k > 0 → sağ alt ağaçta ara: `ara(kok.sagCocuk, 10)` yeni bir çerçeve açar (kok = null).

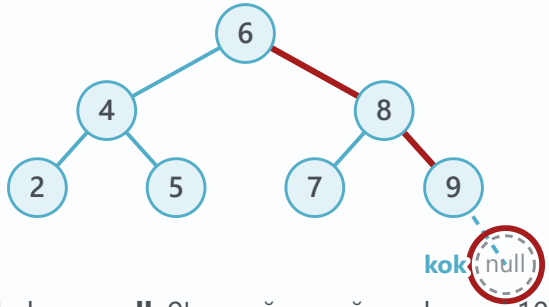
Adım 15 / 20

DEĞİŞKENLER

`kok` = **düğüm(9)** `veri` = **10** `k` = **1**

Kod İzleme: `ara(kok, 5)` ve `ara(kok, 10)`

```
1  boolean ara(IkiliAgacDugum<E> kok, E veri) {
▶      if (kok == null)
▶          return false;            // yok
4      int k = veri.compareTo(kok.veri);
5      if (k == 0)
6          return true;             // bulundu
7      else if (k < 0)
8          return ara(kok.solCocuk, veri);
9      else
10         return ara(kok.sagCocuk, veri);
11 }
```



kok == **null**: 9'un sağ çocuğu yok, yani 10 ağaçta **yok** → **return false**.

Adım 16 / 20

ÇAĞRI YIĞINI

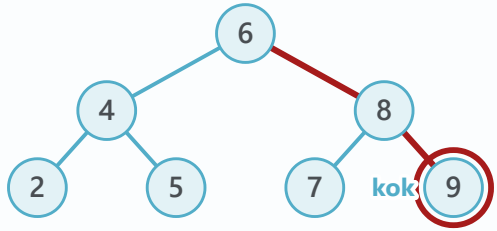
- **`ara(null, 10)`**
- `ara(9, 10)`
- `ara(8, 10)`
- `ara(6, 10)`
- `main()`

DEĞİŞKENLER

`kok = null` `veri = 10`

Kod İzleme: `ara(kok, 5)` ve `ara(kok, 10)`

```
1  boolean ara(IkiliAgacDugum<E> kok, E veri) {
2      if (kok == null)
3          return false;            // yok
4      int k = veri.compareTo(kok.veri);
5      if (k == 0)
6          return true;             // bulundu
7      else if (k < 0)
8          return ara(kok.solCocuk, veri);
9      else
▶          return ara(kok.sagCocuk, veri);
11 }
```



`ara(null, 10)` çağrısı **false** döndürdü; 9'daki çerçeve bu değeri aynen **return** eder.

Adım 17 / 20

ÇAĞRI YIĞINI

- **ara(9, 10)**
- ara(8, 10)
- ara(6, 10)
- main()

DEĞİŞKENLER

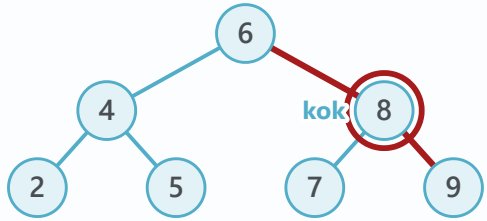
`kok = düğüm(9)` `veri = 10` `k = 1` `dönen = false`

Kod İzleme: `ara(kok, 5)` ve `ara(kok, 10)`

```
1  boolean ara(IkiliAgacDugum<E> kok, E veri) {
2      if (kok == null)
3          return false;            // yok
4      int k = veri.compareTo(kok.veri);
5      if (k == 0)
6          return true;             // bulundu
7      else if (k < 0)
8          return ara(kok.solCocuk, veri);
9      else
▶          return ara(kok.sagCocuk, veri);
11 }
```

ÇAĞRI YIĞINI

- **`ara(8, 10)`**
- `ara(6, 10)`
- `main()`



`ara(9, 10)` çağrısı **false** döndürdü; 8'deki çerçeve bu değeri aynen **return** eder.

Adım 18 / 20

DEĞİŞKENLER

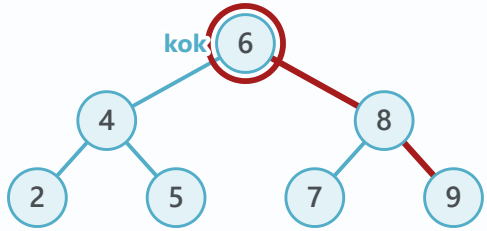
`kok = düğüm(8)` `veri = 10` `k = 1` `dönen = false`

Kod İzleme: `ara(kok, 5)` ve `ara(kok, 10)`

```
1  boolean ara(IkiliAgacDugum<E> kok, E veri) {
2      if (kok == null)
3          return false;            // yok
4      int k = veri.compareTo(kok.veri);
5      if (k == 0)
6          return true;             // bulundu
7      else if (k < 0)
8          return ara(kok.solCocuk, veri);
9      else
▶          return ara(kok.sagCocuk, veri);
11 }
```

ÇAĞRI YIĞINI

- **`ara(6, 10)`**
- `main()`



`ara(8, 10)` çağrısı **false** döndürdü; 6'daki çerçeve bu değeri aynen **return** eder.

Adım 19 / 20

DEĞİŞKENLER

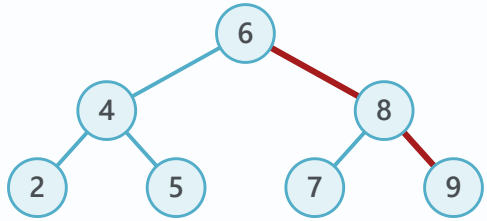
`kok = düğüm(6)` `veri = 10` `k = 1` `dönen = false`

Kod İzleme: `ara(kok, 5)` ve `ara(kok, 10)`

```
▶ boolean ara(IkiliAgacDugum<E> kok, E veri) {
2     if (kok == null)
3         return false;            // yok
4     int k = veri.compareTo(kok.veri);
5     if (k == 0)
6         return true;             // bulundu
7     else if (k < 0)
8         return ara(kok.solCocuk, veri);
9     else
10        return ara(kok.sagCocuk, veri);
11 }
```

ÇAĞRI YIĞINI

`main()`



Sonuç **false**: 3 düğüm ziyaret edildi. En fazla h + 1 düğüm → **O(h)**.

Adım 20 / 20

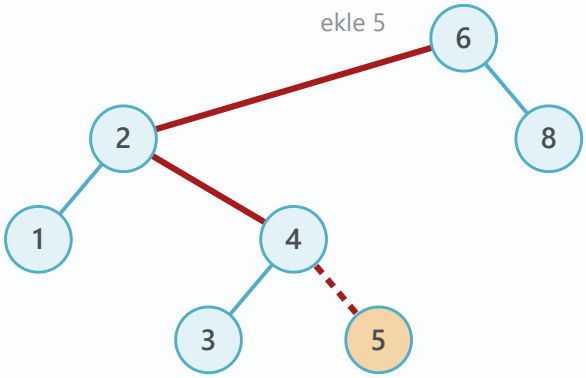
DEĞİŞKENLER

`sonuç = false`

BST İŞLEMLERİ

Öğe Ekleme

- Aranıyormuş gibi in; düştüğün **null** konuma yeni yaprak olarak ekle.
- Yeni düğüm her zaman **yaprak** olur; O(h).

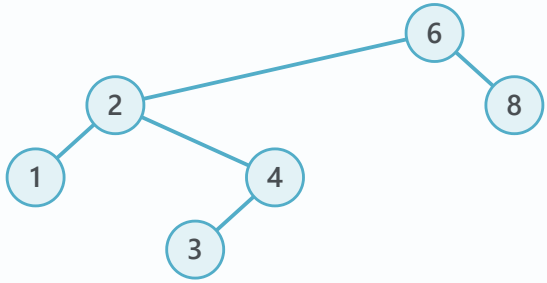



```
IkiliAgacDugum<E> ekle(
        IkiliAgacDugum<E> kok, E veri) {
    if (kok == null)
        return new IkiliAgacDugum<>(veri);
    int k = veri.compareTo(kok.veri);
    if (k < 0)
        kok.solCocuk = ekle(kok.solCocuk, veri);
    else if (k > 0)
        kok.sagCocuk = ekle(kok.sagCocuk, veri);
    // k == 0: anahtar zaten var, eklenmez
    return kok;
}
// kullanım: kok = ekle(kok, 5);
```

Özyinelemeli çağrının sonucu **çocuk bağına geri atanır**; böylece yeni düğüm, null olan bağın yerine bağlanır.

Kod İzleme: `ekle(kok, 5)`

```
1  IkiliAgacDugum<E> ekle(
2          IkiliAgacDugum<E> kok, E veri) {
3      if (kok == null)
4          return new IkiliAgacDugum<>(veri);
5      int k = veri.compareTo(kok.veri);
6      if (k < 0)
7          kok.solCocuk = ekle(kok.solCocuk, veri);
8      else if (k > 0)
9          kok.sagCocuk = ekle(kok.sagCocuk, veri);
10     // k == 0: anahtar zaten var, eklenmez
11     return kok;
12 }
▶  // kullanım: kok = ekle(kok, 5);
```



main, `kok = ekle(kok, 5)` çağırır. Ekleme de aramada olduğu gibi kökten başlar.

Adım 1 / 13

ÇAĞRI YIĞINI

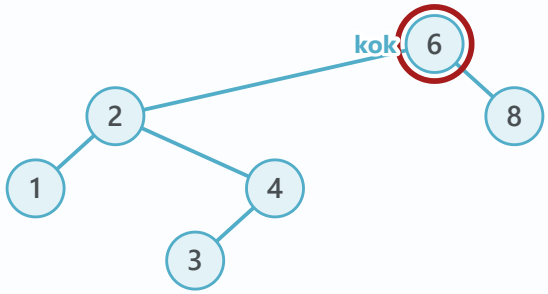
`main()`

DEĞİŞKENLER

`veri = 5`

Kod İzleme: `ekle(kok, 5)`

```
1  IkiliAgacDugum<E> ekle(
2          IkiliAgacDugum<E> kok, E veri) {
▶      if (kok == null)
4          return new IkiliAgacDugum<>(veri);
▶      int k = veri.compareTo(kok.veri);
6      if (k < 0)
7          kok.solCocuk = ekle(kok.solCocuk, veri);
8      else if (k > 0)
9          kok.sagCocuk = ekle(kok.sagCocuk, veri);
10     // k == 0: anahtar zaten var, eklenmez
11     return kok;
12 }
13 // kullanım: kok = ekle(kok, 5);
```



kok ≠ null (6). **k = -1** çünkü 5 < 6.

Adım 2 / 13

ÇAĞRI YIĞINI

- `ekle(6, 5)`
- `main()`

DEĞİŞKENLER

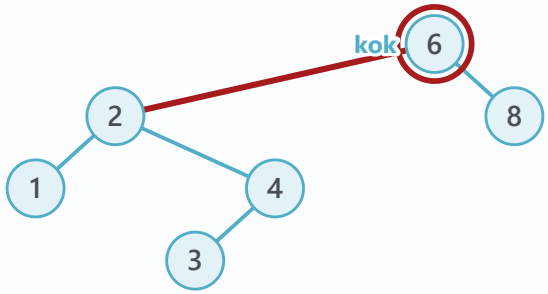
`kok = düğüm(6)` `veri = 5` `k = -1`

Kod İzleme: `ekle(kok, 5)`

```
1  IkiliAgacDugum<E> ekle(
2          IkiliAgacDugum<E> kok, E veri) {
3      if (kok == null)
4          return new IkiliAgacDugum<>(veri);
5      int k = veri.compareTo(kok.veri);
▶      if (k < 0)
▶          kok.solCocuk = ekle(kok.solCocuk, veri);
8      else if (k > 0)
9          kok.sagCocuk = ekle(kok.sagCocuk, veri);
10     // k == 0: anahtar zaten var, eklenmez
11     return kok;
12 }
13 // kullanım: kok = ekle(kok, 5);
```

ÇAĞRI YIĞINI

- **`ekle(6, 5)`**
- `main()`



Sol alt ağaca in: `ekle(kok.solCocuk, 5)` ; kok = 2. Dönen değer **kok.solCocuk**'a atanacak.

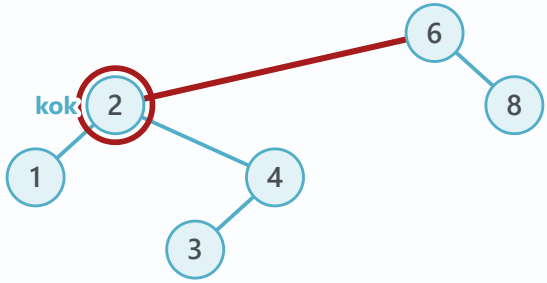
Adım 3 / 13

DEĞİŞKENLER

`kok` = **düğüm(6)** | `veri` = **5** | `k` = **−1**

Kod İzleme: `ekle(kok, 5)`

```
1  IkiliAgacDugum<E> ekle(
2          IkiliAgacDugum<E> kok, E veri) {
▶      if (kok == null)
4          return new IkiliAgacDugum<>(veri);
▶      int k = veri.compareTo(kok.veri);
6      if (k < 0)
7          kok.solCocuk = ekle(kok.solCocuk, veri);
8      else if (k > 0)
9          kok.sagCocuk = ekle(kok.sagCocuk, veri);
10     // k == 0: anahtar zaten var, eklenmez
11     return kok;
12 }
13 // kullanım: kok = ekle(kok, 5);
```



kok ≠ null (2). **k = 1** çünkü 5 > 2.

Adım 4 / 13

ÇAĞRI YIĞINI

- **ekle(2, 5)**
- ekle(6, 5)
- main()

DEĞİŞKENLER

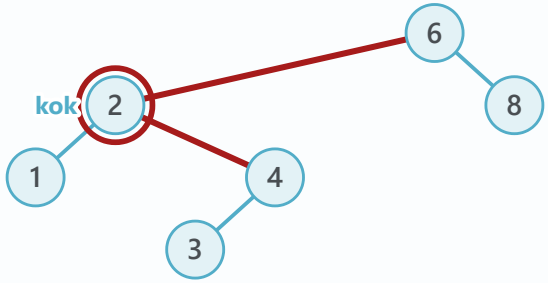
`kok` = **düğüm(2)** `veri` = **5** `k` = **1**

Kod İzleme: `ekle(kok, 5)`

```
1  IkiliAgacDugum<E> ekle(
2          IkiliAgacDugum<E> kok, E veri) {
3      if (kok == null)
4          return new IkiliAgacDugum<>(veri);
5      int k = veri.compareTo(kok.veri);
6      if (k < 0)
7          kok.solCocuk = ekle(kok.solCocuk, veri);
▶      else if (k > 0)
▶          kok.sagCocuk = ekle(kok.sagCocuk, veri);
10     // k == 0: anahtar zaten var, eklenmez
11     return kok;
12 }
13 // kullanım: kok = ekle(kok, 5);
```

ÇAĞRI YIĞINI

- **`ekle(2, 5)`**
- `ekle(6, 5)`
- `main()`



Sağ alt ağaca in: `ekle(kok.sagCocuk, 5)` ; kok = 4. Dönen değer **kok.sagCocuk**'a atanacak.

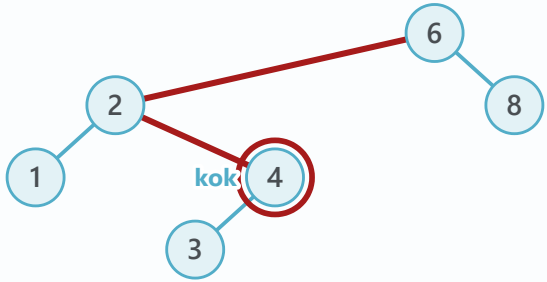
Adım 5 / 13

DEĞİŞKENLER

`kok` = **düğüm(2)** `veri` = **5** `k` = **1**

Kod İzleme: `ekle(kok, 5)`

```
1  IkiliAgacDugum<E> ekle(
2          IkiliAgacDugum<E> kok, E veri) {
▶      if (kok == null)
4          return new IkiliAgacDugum<>(veri);
▶      int k = veri.compareTo(kok.veri);
6      if (k < 0)
7          kok.solCocuk = ekle(kok.solCocuk, veri);
8      else if (k > 0)
9          kok.sagCocuk = ekle(kok.sagCocuk, veri);
10     // k == 0: anahtar zaten var, eklenmez
11     return kok;
12 }
13 // kullanım: kok = ekle(kok, 5);
```



kok ≠ null (4). **k = 1** çünkü 5 > 4.

Adım 6 / 13

ÇAĞRI YIĞINI

- **ekle(4, 5)**
- ekle(2, 5)
- ekle(6, 5)
- main()

DEĞİŞKENLER

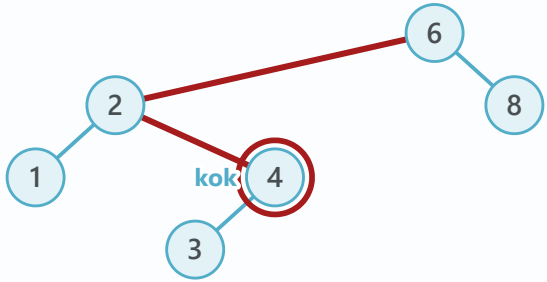
`kok` = **düğüm(4)** | `veri` = **5** | `k` = **1**

Kod İzleme: `ekle(kok, 5)`

```
1  IkiliAgacDugum<E> ekle(
2          IkiliAgacDugum<E> kok, E veri) {
3      if (kok == null)
4          return new IkiliAgacDugum<>(veri);
5      int k = veri.compareTo(kok.veri);
6      if (k < 0)
7          kok.solCocuk = ekle(kok.solCocuk, veri);
▶      else if (k > 0)
▶          kok.sagCocuk = ekle(kok.sagCocuk, veri);
10     // k == 0: anahtar zaten var, eklenmez
11     return kok;
12 }
13 // kullanım: kok = ekle(kok, 5);
```

ÇAĞRI YIĞINI

- **`ekle(4, 5)`**
- `ekle(2, 5)`
- `ekle(6, 5)`
- `main()`



Sağ alt ağaca in: `ekle(kok.sagCocuk, 5)` ; kok = null. Dönen değer **kok.sagCocuk**'a atanacak.

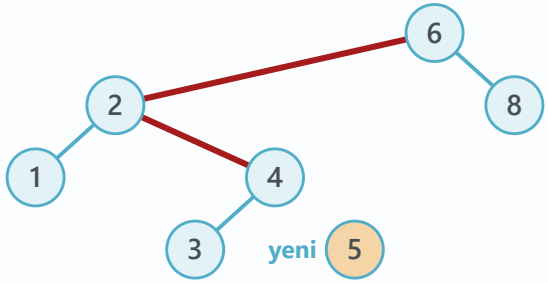
Adım 7 / 13

DEĞİŞKENLER

`kok` = **düğüm(4)** | `veri` = **5** | `k` = **1**

Kod İzleme: `ekle(kok, 5)`

```
1  IkiliAgacDugum<E> ekle(
2          IkiliAgacDugum<E> kok, E veri) {
▶      if (kok == null)
▶          return new IkiliAgacDugum<>(veri);
5      int k = veri.compareTo(kok.veri);
6      if (k < 0)
7          kok.solCocuk = ekle(kok.solCocuk, veri);
8      else if (k > 0)
9          kok.sagCocuk = ekle(kok.sagCocuk, veri);
10     // k == 0: anahtar zaten var, eklenmez
11     return kok;
12 }
13 // kullanım: kok = ekle(kok, 5);
```



kok == **null**: boş yere ulaşıldı. `new` ile düğüm 5 oluşturulur ve **döndürülür** — henüz hiçbir bağ onu göstermiyor.

Adım 8 / 13

ÇAĞRI YIĞINI

- **`ekle(null, 5)`**
- `ekle(4, 5)`
- `ekle(2, 5)`
- `ekle(6, 5)`
- `main()`

DEĞİŞKENLER

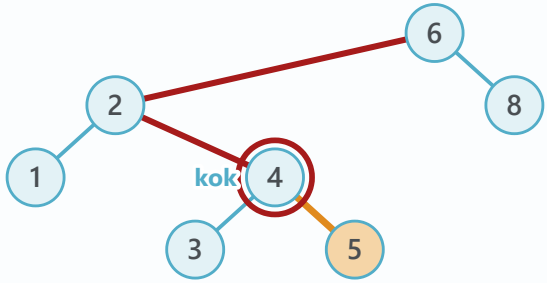
`kok = null` `veri = 5`

Kod İzleme: `ekle(kok, 5)`

```
1  IkiliAgacDugum<E> ekle(
2          IkiliAgacDugum<E> kok, E veri) {
3      if (kok == null)
4          return new IkiliAgacDugum<>(veri);
5      int k = veri.compareTo(kok.veri);
6      if (k < 0)
7          kok.solCocuk = ekle(kok.solCocuk, veri);
8      else if (k > 0)
▶          kok.sagCocuk = ekle(kok.sagCocuk, veri);
10     // k == 0: anahtar zaten var, eklenmez
11     return kok;
12 }
13 // kullanım: kok = ekle(kok, 5);
```

ÇAĞRI YIĞINI

- **ekle(4, 5)**
- ekle(2, 5)
- ekle(6, 5)
- main()



4'ün çerçevesine dönüldü: **kok.sagCocuk = düğüm(5)**. Null olan bağın yerine yeni düğüm bağlandı (turuncu).

Adım 9 / 13

DEĞİŞKENLER

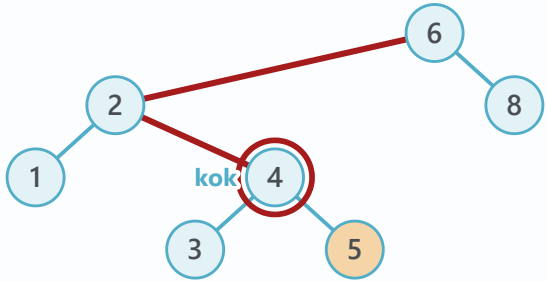
`kok = düğüm(4)` `veri = 5` `k = 1` `dönen = düğüm(5)`

Kod İzleme: `ekle(kok, 5)`

```
1  IkiliAgacDugum<E> ekle(
2          IkiliAgacDugum<E> kok, E veri) {
3      if (kok == null)
4          return new IkiliAgacDugum<>(veri);
5      int k = veri.compareTo(kok.veri);
6      if (k < 0)
7          kok.solCocuk = ekle(kok.solCocuk, veri);
8      else if (k > 0)
9          kok.sagCocuk = ekle(kok.sagCocuk, veri);
10     // k == 0: anahtar zaten var, eklenmez
▶      return kok;
12 }
13 // kullanım: kok = ekle(kok, 5);
```

ÇAĞRI YIĞINI

- **ekle(4, 5)**
- ekle(2, 5)
- ekle(6, 5)
- main()



return kok: 4, kendi ebeveynine (2) geri döndürülür.

Adım 10 / 13

DEĞİŞKENLER

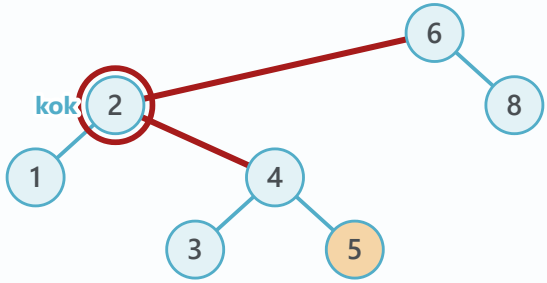
`kok` = **düğüm(4)** | `veri` = **5** | `k` = **1**

Kod İzleme: `ekle(kok, 5)`

```
1  IkiliAgacDugum<E> ekle(
2          IkiliAgacDugum<E> kok, E veri) {
3      if (kok == null)
4          return new IkiliAgacDugum<>(veri);
5      int k = veri.compareTo(kok.veri);
6      if (k < 0)
7          kok.solCocuk = ekle(kok.solCocuk, veri);
8      else if (k > 0)
▶          kok.sagCocuk = ekle(kok.sagCocuk, veri);
10     // k == 0: anahtar zaten var, eklenmez
▶      return kok;
12 }
13 // kullanım: kok = ekle(kok, 5);
```

ÇAĞRI YIĞINI

- **ekle(2, 5)**
- ekle(6, 5)
- main()



2'de `kok.sagCocuk = düğüm(4)` : bağ zaten 4'ü gösteriyordu, **değişmez**. Sonra return 2.

Adım 11 / 13

DEĞİŞKENLER

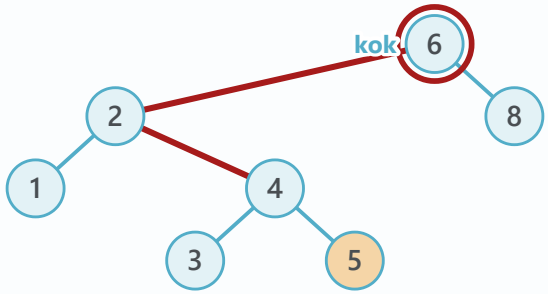
`kok = düğüm(2)` `veri = 5` `k = 1` `dönen = düğüm(4)`

Kod İzleme: `ekle(kok, 5)`

```
1  IkiliAgacDugum<E> ekle(
2          IkiliAgacDugum<E> kok, E veri) {
3      if (kok == null)
4          return new IkiliAgacDugum<>(veri);
5      int k = veri.compareTo(kok.veri);
6      if (k < 0)
▶          kok.solCocuk = ekle(kok.solCocuk, veri);
8      else if (k > 0)
9          kok.sagCocuk = ekle(kok.sagCocuk, veri);
10     // k == 0: anahtar zaten var, eklenmez
▶      return kok;
12 }
13 // kullanım: kok = ekle(kok, 5);
```

ÇAĞRI YIĞINI

- **`ekle(6, 5)`**
- `main()`



6'da `kok.solCocuk = düğüm(2)` (aynı bağ), sonra **return 6**.

Adım 12 / 13

DEĞİŞKENLER

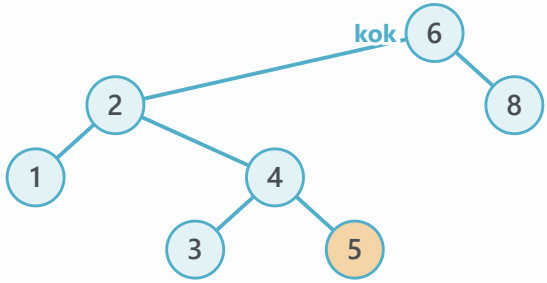
`kok = düğüm(6)` `veri = 5` `k = −1` `dönen = düğüm(2)`

Kod İzleme: `ekle(kok, 5)`

```
1  IkiliAgacDugum<E> ekle(
2          IkiliAgacDugum<E> kok, E veri) {
3      if (kok == null)
4          return new IkiliAgacDugum<>(veri);
5      int k = veri.compareTo(kok.veri);
6      if (k < 0)
7          kok.solCocuk = ekle(kok.solCocuk, veri);
8      else if (k > 0)
9          kok.sagCocuk = ekle(kok.sagCocuk, veri);
10     // k == 0: anahtar zaten var, eklenmez
11     return kok;
12 }
▶  // kullanım: kok = ekle(kok, 5);
```

ÇAĞRI YIĞINI

`main()`



main: `kok = 6` (kök değişmedi). Yeni düğüm hep **yaprak** olur; yol uzunluğu kadar çağrı → **O(h)**.

Adım 13 / 13

DEĞİŞKENLER

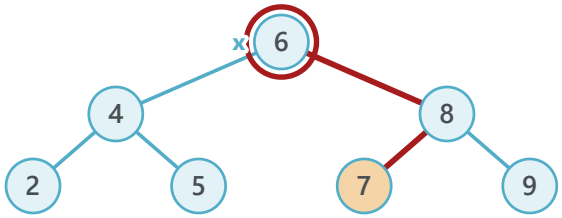
`kok = düğüm(6)`

BST İŞLEMLERİ

Ardıl (Successor) ve Öncül (Predecessor)

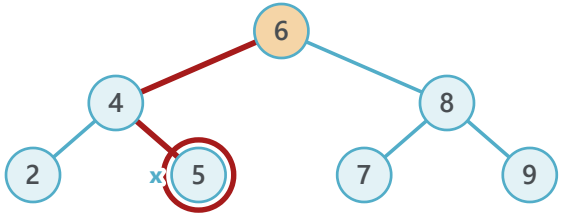
1) Sağ alt ağaç varsa

ardıl = sağ alt ağacın **en küçüğü**: bir sağa, sonra hep sola.



2) Sağ alt ağaç yoksa

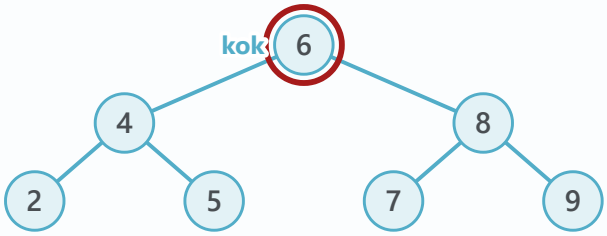
ardıl = **solundan geçtiğimiz** en yakın ata (5 → 4 → 6: 5, 6'nın sol alt ağacında).



İşlem	Anlamı	Java `TreeMap`	Yukarıdaki ağaçta
ardıl	x'ten büyük en küçük anahtar	`higherKey(x)`	ardil(5) = 6
öncül	x'ten küçük en büyük anahtar	`lowerKey(x)`	oncul(6) = 5
tavan (ceiling)	≥ x olan en küçük anahtar	`ceilingKey(x)`	tavan(3) = 4
taban (floor)	≤ x olan en büyük anahtar	`floorKey(x)`	taban(3) = 2

Hepsi kökten tek bir yol izler: **O(h)**. İki çocuklu düğüm silinirken yerine konan düğüm, ardılıdır.

Kod İzleme: `ardil(kok, 5)` ve `ardil(kok, 6)`



`ardil(kok, 5)` : henüz aday yok (`aday = null`); kökten (6) başla.

Adım 1 / 10

```
AgacDugumu ardil(AgacDugumu kok, int x) {
    AgacDugumu aday = null; // x'ten büyük en küçük
    while (kok != null) {
        if (x < kok.veri) {
            aday = kok;     // kok bir aday
            kok = kok.sol;  // daha küçüğü solda
        } else
            kok = kok.sag;  // x'ten büyükler sağda
    }
    return aday;
}
```

DEĞİŞKENLER

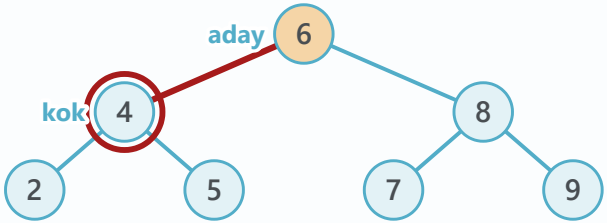
`x = 5` `kok = düğüm(6)` `aday = null`

ÇAĞRI YIĞINI

`ardil(kok, 5)`

`main()`

Kod İzleme: `ardil(kok, 5)` ve `ardil(kok, 6)`



5 < 6 → **6 bir aday** (x'ten büyük); daha küçük bir aday ancak solda olabilir → sola in.

Adım 2 / 10

```
1  AgacDugumu ardil(AgacDugumu kok, int x) {
2      AgacDugumu aday = null; // x'ten büyük en küçük
▶      while (kok != null) {
▶          if (x < kok.veri) {
▶              aday = kok;      // kok bir aday
▶              kok = kok.sol;   // daha küçüğü solda
7          } else
8              kok = kok.sag;   // x'ten büyükler sağda
9      }
10     return aday;
11 }
```

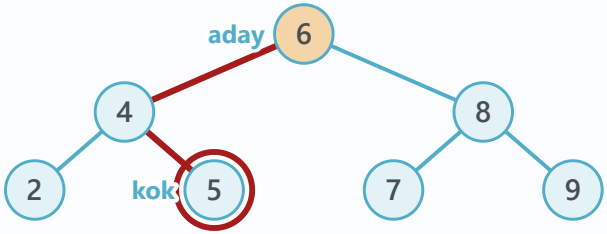
DEĞİŞKENLER

`x = 5` `kok = düğüm(4)` `aday = düğüm(6)`

ÇAĞRI YIĞINI

- `ardil(kok, 5)`
- `main()`

Kod İzleme: `ardil(kok, 5)` ve `ardil(kok, 6)`



5 ≥ 4 → 4 ve solundakiler ardıl olamaz; x'ten büyükler **sağda** → sağa in.

Adım 3 / 10

```
1  AgacDugumu ardil(AgacDugumu kok, int x) {
2      AgacDugumu aday = null; // x'ten büyük en küçük
▶      while (kok != null) {
▶          if (x < kok.veri) {
5              aday = kok;      // kok bir aday
6              kok = kok.sol;   // daha küçüğü solda
▶          } else
▶              kok = kok.sag;   // x'ten büyükler sağda
9      }
10     return aday;
11 }
```

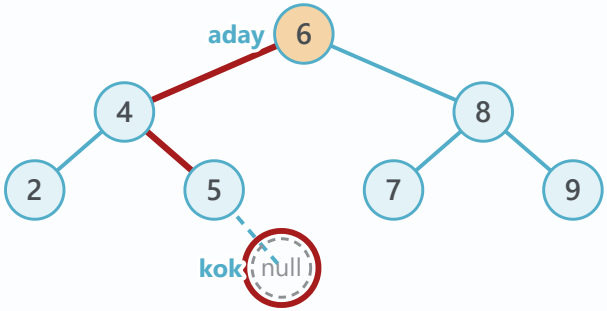
DEĞİŞKENLER

`x = 5` `kok = düğüm(5)` `aday = düğüm(6)`

ÇAĞRI YIĞINI

- `ardil(kok, 5)`
- `main()`

Kod İzleme: `ardil(kok, 5)` ve `ardil(kok, 6)`



5 ≥ 5 → 5 ve solundakiler ardıl olamaz; x'ten büyükler **sağda** → sağa in. `kok = null` oldu.

Adım 4 / 10

```
1  AgacDugumu ardil(AgacDugumu kok, int x) {
2      AgacDugumu aday = null; // x'ten büyük en küçük
▶      while (kok != null) {
▶          if (x < kok.veri) {
5              aday = kok;      // kok bir aday
6              kok = kok.sol;   // daha küçüğü solda
▶          } else
▶              kok = kok.sag;   // x'ten büyükler sağda
9      }
10     return aday;
11 }
```

DEĞİŞKENLER

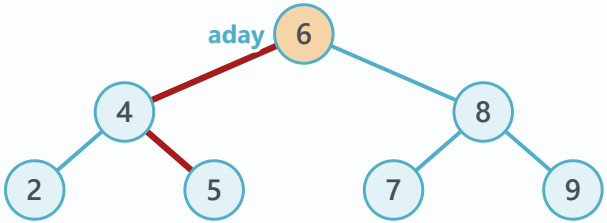
`x = 5` `kok = null` `aday = düğüm(6)`

ÇAĞRI YIĞINI

`ardil(kok, 5)`

`main()`

Kod İzleme: `ardil(kok, 5)` ve `ardil(kok, 6)`



Döngü bitti (`kok == null`) → **return düğüm(6)**: ardil(5) = **6**.

Adım 5 / 10

```
1  AgacDugumu ardil(AgacDugumu kok, int x) {
2      AgacDugumu aday = null; // x'ten büyük en küçük
▶      while (kok != null) {
4          if (x < kok.veri) {
5              aday = kok;      // kok bir aday
6              kok = kok.sol;   // daha küçüğü solda
7          } else
8              kok = kok.sag;   // x'ten büyükler sağda
9      }
▶      return aday;
11 }
```

DEĞİŞKENLER

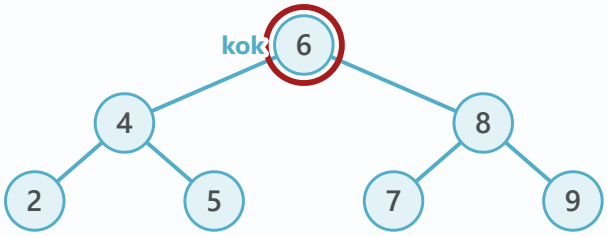
`x = 5` `kok = null` `aday = düğüm(6)`

ÇAĞRI YIĞINI

`ardil(kok, 5)`

`main()`

Kod İzleme: `ardil(kok, 5)` ve `ardil(kok, 6)`



İkinci çağrı: `ardil(kok, 6)` : henüz aday yok (`aday = null`); kökten (6) başla.

Adım 6 / 10

```
AgacDugumu ardil(AgacDugumu kok, int x) {
    AgacDugumu aday = null; // x'ten büyük en küçük
    while (kok != null) {
        if (x < kok.veri) {
            aday = kok;      // kok bir aday
            kok = kok.sol;   // daha küçüğü solda
        } else
            kok = kok.sag;   // x'ten büyükler sağda
    }
    return aday;
}
```

DEĞİŞKENLER

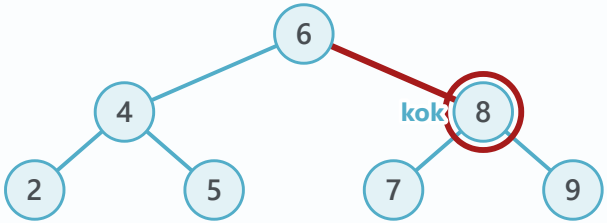
`x = 6` `kok = düğüm(6)` `aday = null`

ÇAĞRI YIĞINI

`ardil(kok, 6)`

`main()`

Kod İzleme: `ardil(kok, 5)` ve `ardil(kok, 6)`



6 ≥ 6 → 6 ve solundakiler ardıl olamaz; x'ten büyükler **sağda** → sağa in.

Adım 7 / 10

```
1  AgacDugumu ardil(AgacDugumu kok, int x) {
2      AgacDugumu aday = null; // x'ten büyük en küçük
▶      while (kok != null) {
▶          if (x < kok.veri) {
5              aday = kok;      // kok bir aday
6              kok = kok.sol;   // daha küçüğü solda
▶          } else
▶              kok = kok.sag;   // x'ten büyükler sağda
9      }
10     return aday;
11 }
```

DEĞİŞKENLER

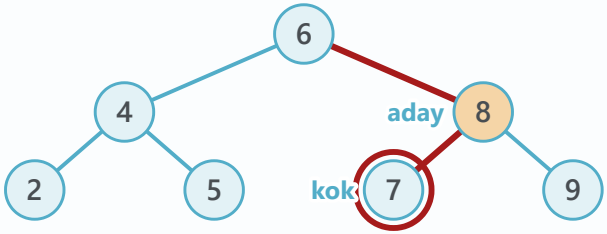
`x = 6` `kok = düğüm(8)` `aday = null`

ÇAĞRI YIĞINI

`ardil(kok, 6)`

`main()`

Kod İzleme: `ardil(kok, 5)` ve `ardil(kok, 6)`



6 < 8 → **8 bir aday** (x'ten büyük); daha küçük bir aday ancak solda olabilir → sola in.

Adım 8 / 10

```
1  AgacDugumu ardil(AgacDugumu kok, int x) {
2      AgacDugumu aday = null; // x'ten büyük en küçük
▶      while (kok != null) {
▶          if (x < kok.veri) {
▶              aday = kok;      // kok bir aday
▶              kok = kok.sol;   // daha küçüğü solda
7          } else
8              kok = kok.sag;   // x'ten büyükler sağda
9      }
10     return aday;
11 }
```

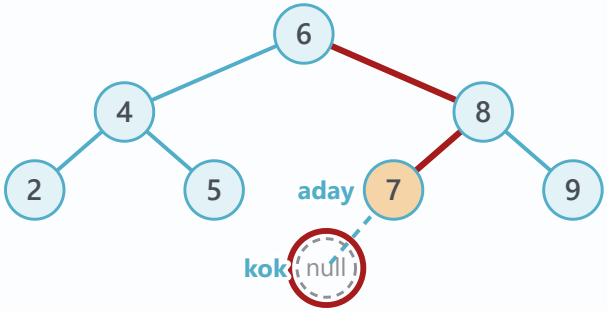
DEĞİŞKENLER

`x = 6` `kok = düğüm(7)` `aday = düğüm(8)`

ÇAĞRI YIĞINI

- `ardil(kok, 6)`
- `main()`

Kod İzleme: `ardil(kok, 5)` ve `ardil(kok, 6)`



6 < 7 → **7 bir aday** (x'ten büyük); daha küçük bir aday ancak solda olabilir → sola in. `kok = null` oldu.

Adım 9 / 10

```
1  AgacDugumu ardil(AgacDugumu kok, int x) {
2      AgacDugumu aday = null; // x'ten büyük en küçük
▶      while (kok != null) {
▶          if (x < kok.veri) {
▶              aday = kok;      // kok bir aday
▶              kok = kok.sol;   // daha küçüğü solda
7          } else
8              kok = kok.sag;   // x'ten büyükler sağda
9      }
10     return aday;
11 }
```

DEĞİŞKENLER

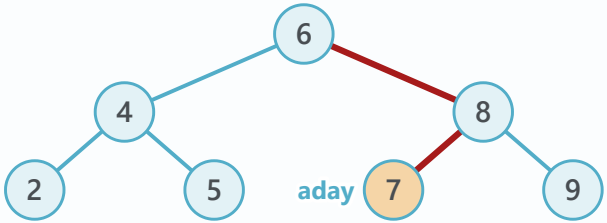
`x = 6` `kok = null` `aday = düğüm(7)`

ÇAĞRI YIĞINI

`ardil(kok, 6)`

`main()`

Kod İzleme: `ardil(kok, 5)` ve `ardil(kok, 6)`



Döngü bitti (`kok == null`) → **return düğüm(7)**: ardil(6) = **7**.

Adım 10 / 10

```
1  AgacDugumu ardil(AgacDugumu kok, int x) {
2      AgacDugumu aday = null; // x'ten büyük en küçük
▶      while (kok != null) {
4          if (x < kok.veri) {
5              aday = kok;      // kok bir aday
6              kok = kok.sol;   // daha küçüğü solda
7          } else
8              kok = kok.sag;   // x'ten büyükler sağda
9      }
▶      return aday;
11 }
```

DEĞİŞKENLER

`x = 6` `kok = null` `aday = düğüm(7)`

ÇAĞRI YIĞINI

`ardil(kok, 6)`

`main()`

BST İŞLEMLERİ

Öğe Silme: Üç Durum

Önce silinecek düğüm aranır (O(h)). Sonra çocuk sayısına göre:

1. Yaprak

Çocuğu yok. Ebeveynin ilgili bağı `null` yapılır.

```
sil 70
```

2. Tek çocuk

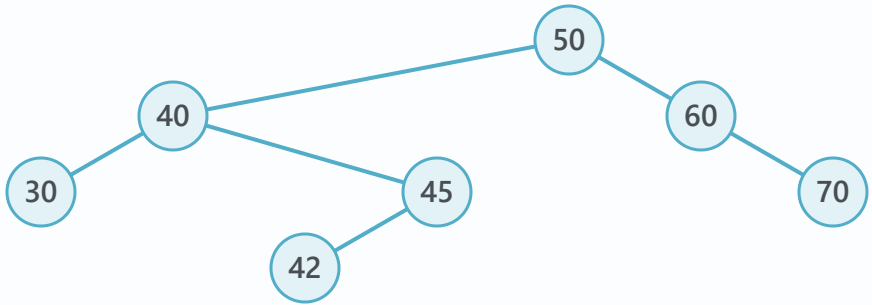
Düğüm atlanır: ebeveyn doğrudan tek çocuğa bağlanır.

```
sil 45 · sil 60
```

3. İki çocuk

Değer, kök ortada ardıl (sağın en küçüğü) ya da öncül (solun en büyüğü) ile değiştirilir.

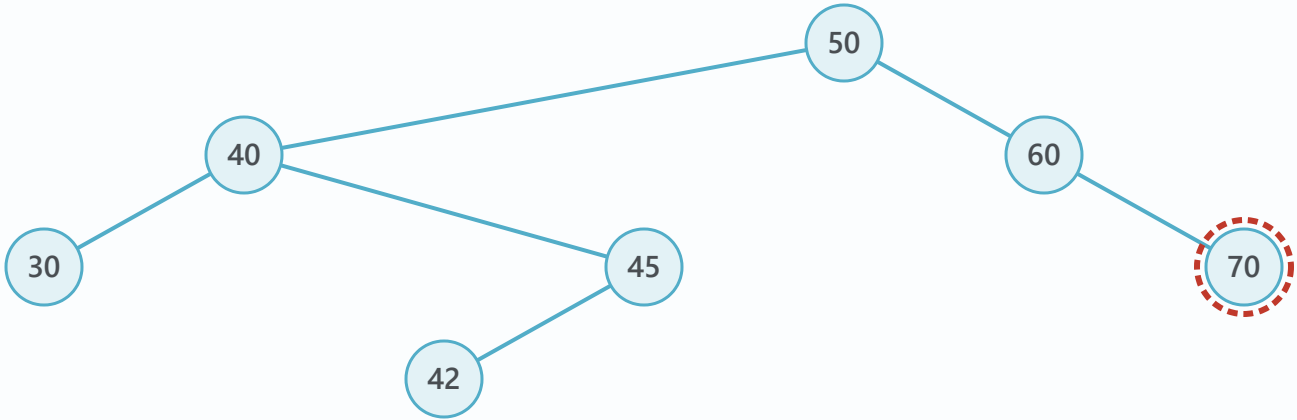
```
sil 40
```



Örnek ağaç

BST İŞLEMLERİ

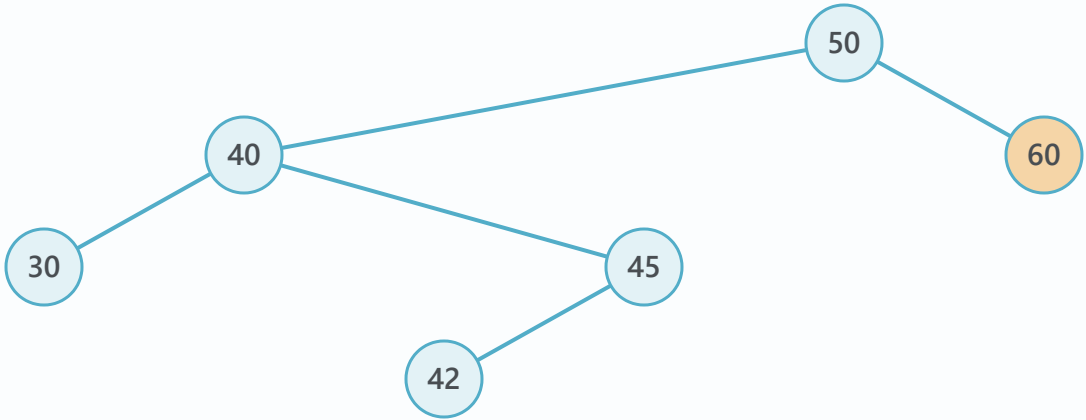
Silme: Yaprak ve Tek Çocuk



Yaprak: 70'in çocuğu yok. Ebeveyni 60'ın sağ bağı null yapılır.

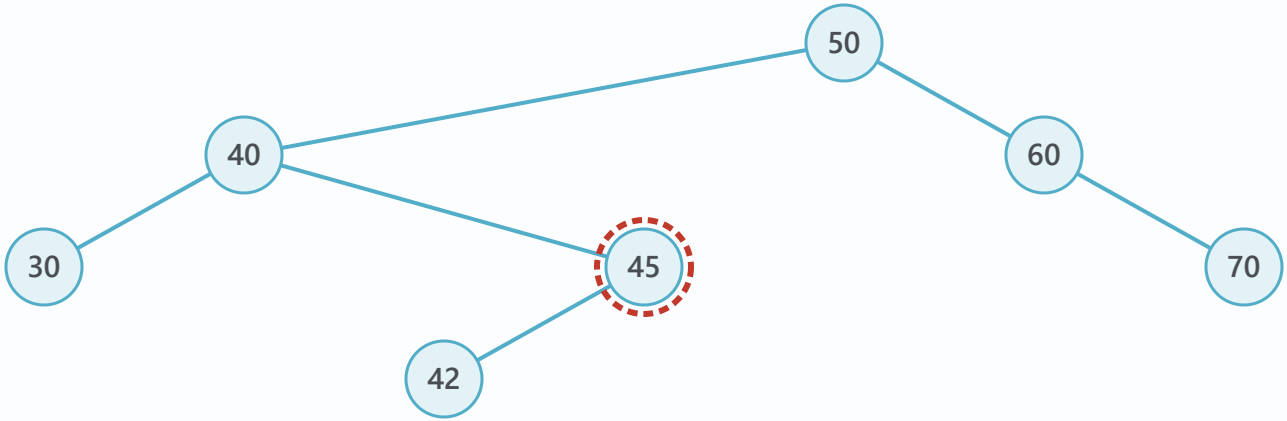
Adım 1 / 6

Silme: Yaprak ve Tek Çocuk



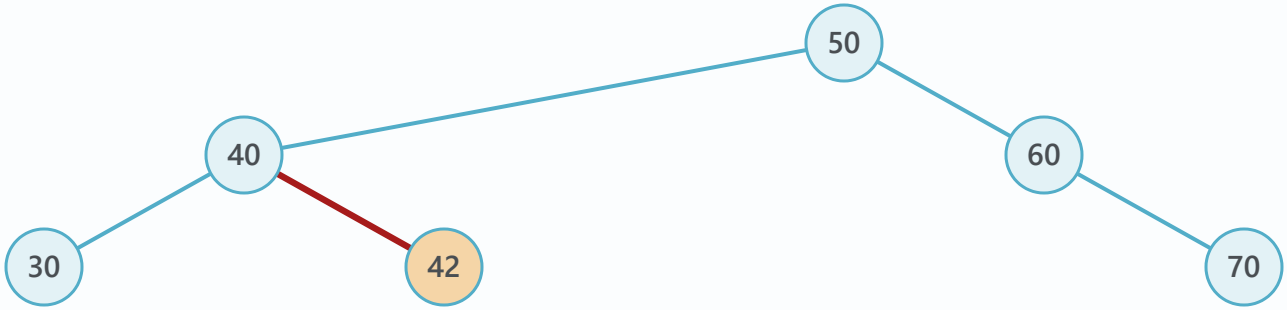
70 silindi; ağacın geri kalanı değişmedi.

Silme: Yaprak ve Tek Çocuk



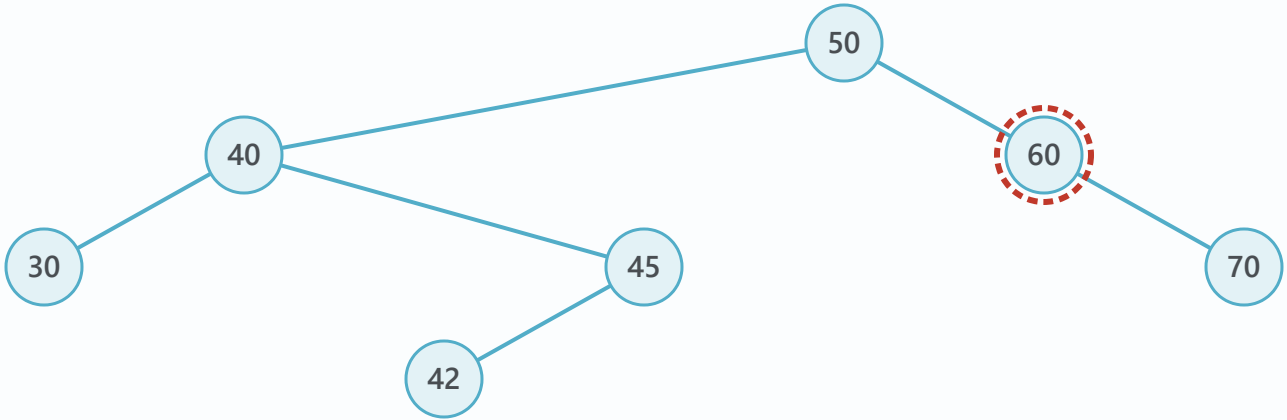
Tek çocuk (sol): 45'in yalnızca sol çocuğu 42 var. (Orijinal ağaçtan başlıyoruz.)

Silme: Yaprak ve Tek Çocuk



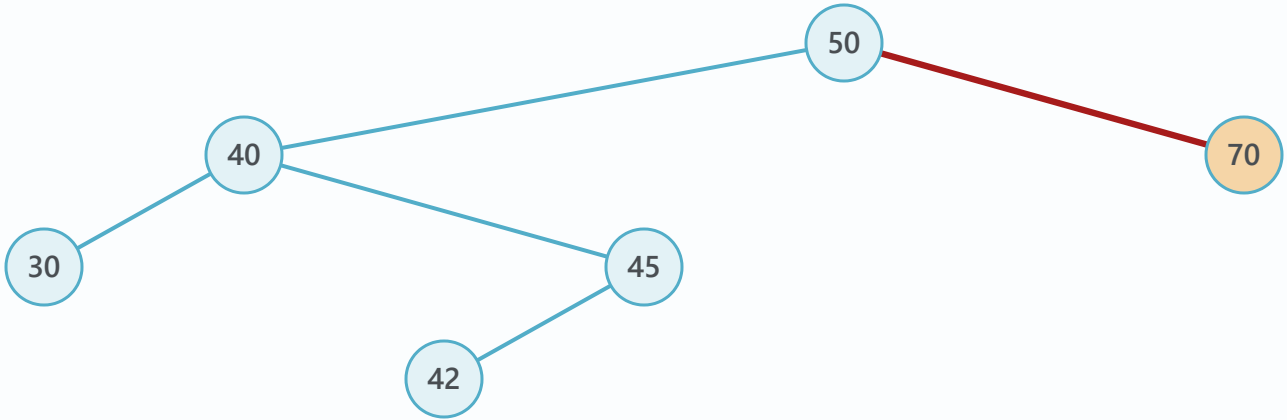
42, 45'in yerine 40'ın sağ çocuğu olur: ebeveyn doğrudan toruna bağlanır.

Silme: Yaprak ve Tek Çocuk



Tek çocuk (sağ): 60'ın yalnızca sağ çocuğu 70 var.

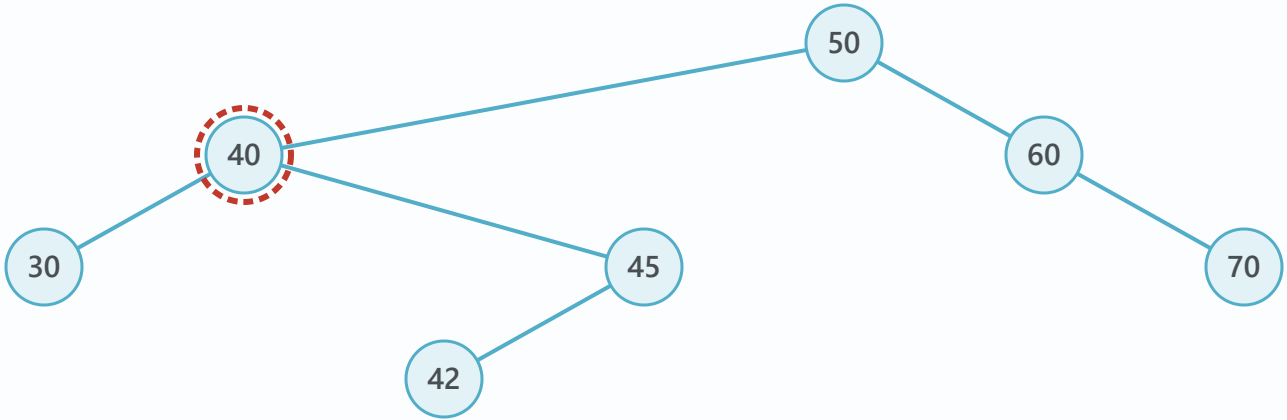
Silme: Yaprak ve Tek Çocuk



70, 60'ın yerine 50'nin sağ çocuğu olur. Sıralama düzeni korunur.

BST İŞLEMLERİ

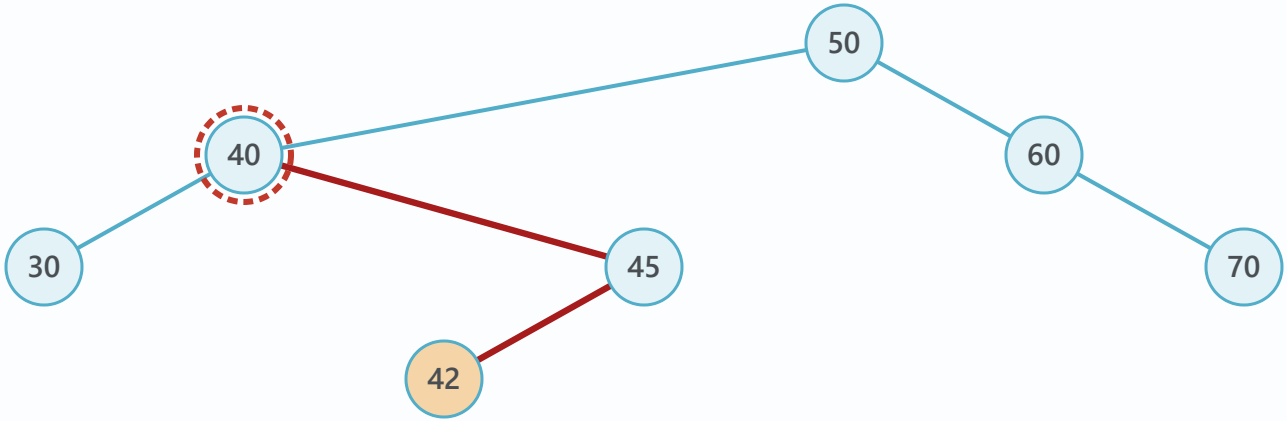
Silme: İki Çocuklu Düğüm



İki çocuk: 40'ın hem solu (30) hem sağı (45) var; doğrudan çıkarılamaz.

Adım 1 / 5

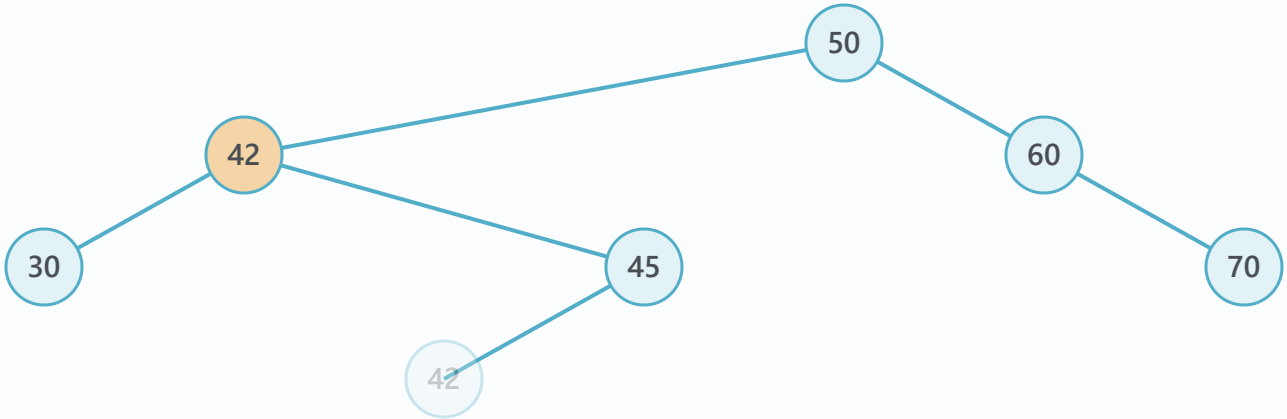
Silme: İki Çocuklu Düğüm



İnorder ardıl = sağ alt ağacın en küçüğü: 45'e geç, sonra hep sola → 42.

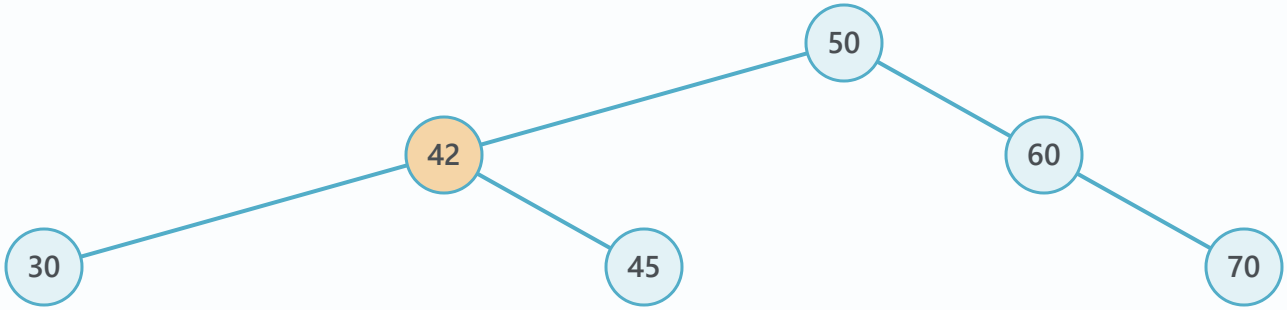
BST İŞLEMLERİ

Silme: İki Çocuklu Düğüm



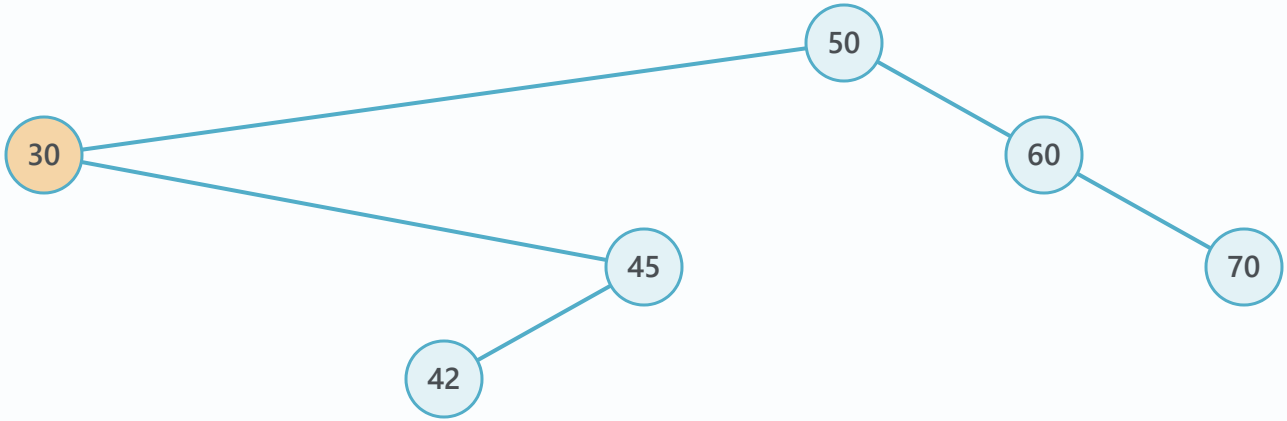
Ardılın değeri (42), 40'ın düğümüne kopyalanır. Şimdi eski 42 fazladan duruyor.

Silme: İki Çocuklu Düğüm



Sağ alt ağaçtan eski 42 silinir; ardılın sol çocuğu olmadığı için bu hep Durum 1 veya 2'dir.

Silme: İki Çocuklu Düğüm



Alternatif: **kök ortada öncül** (sol alt ağacın en büyüğü = 30) kullanılabilir. İkisi de geçerli BST verir.

Silme Kodu

```
IkiliAgacDugum<E> sil(
        IkiliAgacDugum<E> kok, E veri) {
    if (kok == null) return null;   // yok
    int k = veri.compareTo(kok.veri);
    if (k < 0)
        kok.solCocuk = sil(kok.solCocuk, veri);
    else if (k > 0)
        kok.sagCocuk = sil(kok.sagCocuk, veri);
    else if (kok.solCocuk == null)
        return kok.sagCocuk;  // yaprak/sağ çocuk
    else if (kok.sagCocuk == null)
        return kok.solCocuk;  // yalnız sol çocuk
    else {                    // iki çocuk
        kok.veri = minDeger(kok.sagCocuk).veri;
        kok.sagCocuk = sil(kok.sagCocuk, kok.veri);
    }
    return kok;
}
```

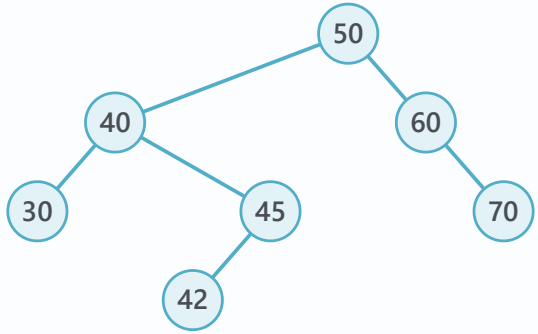
Dönen değer ebeveynin bağına atanır: tek çocuk ya da null dönünce düğüm ağaçtan kopar. Kullanım: `kok = sil(kok, 40);` (kök silinirse yeni kök bu atamayla güncellenir).

Kod İzleme: `sil(kok, 70)` — yaprak

```
▶ IkiliAgacDugum<E> sil(
2         IkiliAgacDugum<E> kok, E veri) {
3     if (kok == null) return null;   // yok
4     int k = veri.compareTo(kok.veri);
5     if (k < 0)
6         kok.solCocuk = sil(kok.solCocuk, veri);
7     else if (k > 0)
8         kok.sagCocuk = sil(kok.sagCocuk, veri);
9     else if (kok.solCocuk == null)
10        return kok.sagCocuk;  // yaprak/sağ çocuk
11    else if (kok.sagCocuk == null)
12        return kok.solCocuk;  // yalnız sol çocuk
13    else {                    // iki çocuk
14        kok.veri = minDeger(kok.sagCocuk).veri;
15        kok.sagCocuk = sil(kok.sagCocuk, kok.veri);
16    }
17    return kok;
18 }
```

DEĞİŞKENLER

`veri = 70`



main, `kok = sil(kok, 70)` çağırır. Önce silinecek düğüm aranır.

Adım 1 / 11

ÇAĞRI YIĞINI

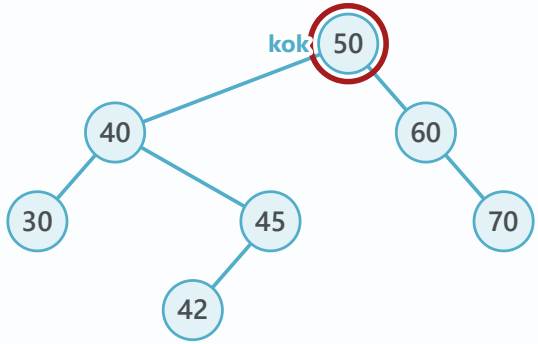
`main()`

Kod İzleme: `sil(kok, 70)` — yaprak

```
1  IkiliAgacDugum<E> sil(
2          IkiliAgacDugum<E> kok, E veri) {
▶      if (kok == null) return null;   // yok
▶      int k = veri.compareTo(kok.veri);
5      if (k < 0)
6          kok.solCocuk = sil(kok.solCocuk, veri);
7      else if (k > 0)
8          kok.sagCocuk = sil(kok.sagCocuk, veri);
9      else if (kok.solCocuk == null)
10         return kok.sagCocuk;  // yaprak/sağ çocuk
11     else if (kok.sagCocuk == null)
12         return kok.solCocuk;  // yalnız sol çocuk
13     else {                    // iki çocuk
14         kok.veri = minDeger(kok.sagCocuk).veri;
15         kok.sagCocuk = sil(kok.sagCocuk, kok.veri);
16     }
17     return kok;
18 }
```

DEĞİŞKENLER

`kok = düğüm(50)` `veri = 70` `k = 1`



kok ≠ null. **k = 1** çünkü 70 > 50.

ÇAĞRI YIĞINI

`sil(50, 70)`

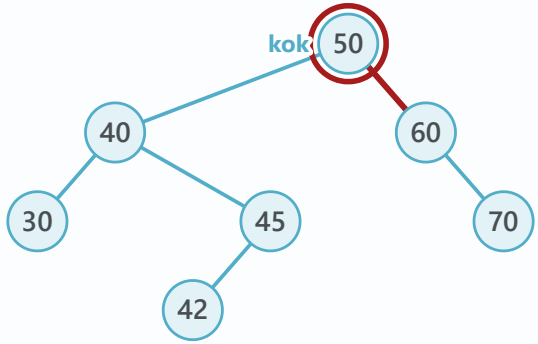
`main()`

Kod İzleme: `sil(kok, 70)` — yaprak

```
1  IkiliAgacDugum<E> sil(
2          IkiliAgacDugum<E> kok, E veri) {
3      if (kok == null) return null;   // yok
4      int k = veri.compareTo(kok.veri);
5      if (k < 0)
6          kok.solCocuk = sil(kok.solCocuk, veri);
▶      else if (k > 0)
▶          kok.sagCocuk = sil(kok.sagCocuk, veri);
9      else if (kok.solCocuk == null)
10         return kok.sagCocuk;  // yaprak/sağ çocuk
11     else if (kok.sagCocuk == null)
12         return kok.solCocuk;  // yalnız sol çocuk
13     else {                    // iki çocuk
14         kok.veri = minDeger(kok.sagCocuk).veri;
15         kok.sagCocuk = sil(kok.sagCocuk, kok.veri);
16     }
17     return kok;
18 }
```

DEĞİŞKENLER

`kok` = **düğüm(50)** | `veri` = **70** | `k` = **1**



Sağ alt ağaçta sil: `sil(kok.sagCocuk, 70)`.
Dönen değer **kok.sagCocuk**'a atanacak.

Adım 3 / 11

ÇAĞRI YIĞINI

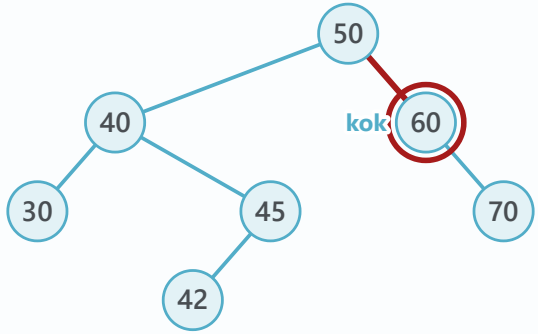
- `sil(50, 70)`
- `main()`

Kod İzleme: `sil(kok, 70)` — yaprak

```
1  IkiliAgacDugum<E> sil(
2          IkiliAgacDugum<E> kok, E veri) {
▶      if (kok == null) return null;   // yok
▶      int k = veri.compareTo(kok.veri);
5      if (k < 0)
6          kok.solCocuk = sil(kok.solCocuk, veri);
7      else if (k > 0)
8          kok.sagCocuk = sil(kok.sagCocuk, veri);
9      else if (kok.solCocuk == null)
10         return kok.sagCocuk;  // yaprak/sağ çocuk
11     else if (kok.sagCocuk == null)
12         return kok.solCocuk;  // yalnız sol çocuk
13     else {                    // iki çocuk
14         kok.veri = minDeger(kok.sagCocuk).veri;
15         kok.sagCocuk = sil(kok.sagCocuk, kok.veri);
16     }
17     return kok;
18 }
```

DEĞİŞKENLER

`kok = düğüm(60)` `veri = 70` `k = 1`



kok ≠ null. **k = 1** çünkü 70 > 60.

Adım 4 / 11

ÇAĞRI YIĞINI

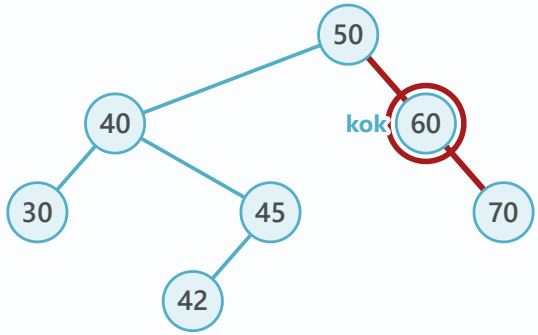
- `sil(60, 70)`
- `sil(50, 70)`
- `main()`

Kod İzleme: `sil(kok, 70)` — yaprak

```
1  IkiliAgacDugum<E> sil(
2          IkiliAgacDugum<E> kok, E veri) {
3      if (kok == null) return null;   // yok
4      int k = veri.compareTo(kok.veri);
5      if (k < 0)
6          kok.solCocuk = sil(kok.solCocuk, veri);
▶      else if (k > 0)
▶          kok.sagCocuk = sil(kok.sagCocuk, veri);
9      else if (kok.solCocuk == null)
10         return kok.sagCocuk;  // yaprak/sağ çocuk
11     else if (kok.sagCocuk == null)
12         return kok.solCocuk;  // yalnız sol çocuk
13     else {                    // iki çocuk
14         kok.veri = minDeger(kok.sagCocuk).veri;
15         kok.sagCocuk = sil(kok.sagCocuk, kok.veri);
16     }
17     return kok;
18 }
```

DEĞİŞKENLER

`kok` = **düğüm(60)** | `veri` = **70** | `k` = **1**



Sağ alt ağaçta sil: `sil(kok.sagCocuk, 70)`.
Dönen değer **kok.sagCocuk**'a atanacak.

Adım 5 / 11

ÇAĞRI YIĞINI

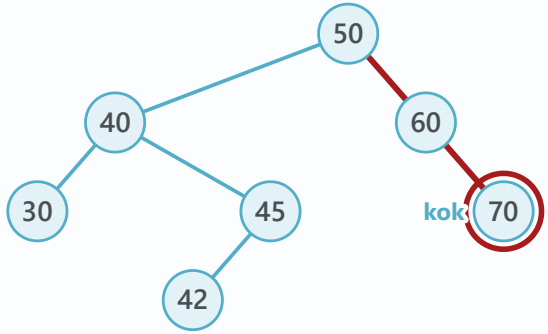
- **`sil(60, 70)`**
- `sil(50, 70)`
- `main()`

Kod İzleme: `sil(kok, 70)` — yaprak

```
1  IkiliAgacDugum<E> sil(
2          IkiliAgacDugum<E> kok, E veri) {
▶      if (kok == null) return null;   // yok
▶      int k = veri.compareTo(kok.veri);
5      if (k < 0)
6          kok.solCocuk = sil(kok.solCocuk, veri);
7      else if (k > 0)
8          kok.sagCocuk = sil(kok.sagCocuk, veri);
9      else if (kok.solCocuk == null)
10         return kok.sagCocuk;  // yaprak/sağ çocuk
11     else if (kok.sagCocuk == null)
12         return kok.solCocuk;  // yalnız sol çocuk
13     else {                    // iki çocuk
14         kok.veri = minDeger(kok.sagCocuk).veri;
15         kok.sagCocuk = sil(kok.sagCocuk, kok.veri);
16     }
17     return kok;
18 }
```

DEĞİŞKENLER

`kok = düğüm(70)` `veri = 70` `k = 0`



kok ≠ null. **k = 0** çünkü 70 = 70.

Adım 6 / 11

ÇAĞRI YIĞINI

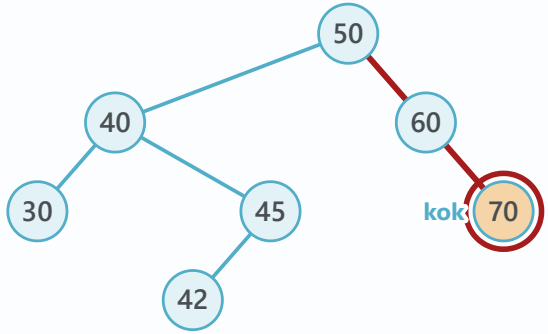
- `sil(70, 70)`
- `sil(60, 70)`
- `sil(50, 70)`
- `main()`

Kod İzleme: `sil(kok, 70)` — yaprak

```
1  IkiliAgacDugum<E> sil(
2          IkiliAgacDugum<E> kok, E veri) {
3      if (kok == null) return null;   // yok
4      int k = veri.compareTo(kok.veri);
5      if (k < 0)
6          kok.solCocuk = sil(kok.solCocuk, veri);
7      else if (k > 0)
8          kok.sagCocuk = sil(kok.sagCocuk, veri);
▶      else if (kok.solCocuk == null)
▶          return kok.sagCocuk;  // yaprak/sağ çocuk
11     else if (kok.sagCocuk == null)
12         return kok.solCocuk;  // yalnız sol çocuk
13     else {                    // iki çocuk
14         kok.veri = minDeger(kok.sagCocuk).veri;
15         kok.sagCocuk = sil(kok.sagCocuk, kok.veri);
16     }
17     return kok;
18 }
```

DEĞİŞKENLER

`kok = düğüm(70)` `veri = 70` `k = 0`



k = 0 → silinecek düğüm bulundu. Sol çocuk yok → **return kok.sagCocuk** = null. (Yaprak: null döner.)

Adım 7 / 11

ÇAĞRI YIĞINI

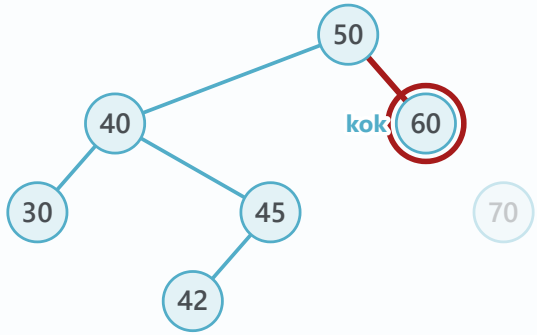
- `sil(70, 70)`
- `sil(60, 70)`
- `sil(50, 70)`
- `main()`

Kod İzleme: `sil(kok, 70)` — yaprak

```
1  IkiliAgacDugum<E> sil(
2          IkiliAgacDugum<E> kok, E veri) {
3      if (kok == null) return null;   // yok
4      int k = veri.compareTo(kok.veri);
5      if (k < 0)
6          kok.solCocuk = sil(kok.solCocuk, veri);
7      else if (k > 0)
▶          kok.sagCocuk = sil(kok.sagCocuk, veri);
9      else if (kok.solCocuk == null)
10         return kok.sagCocuk;  // yaprak/sağ çocuk
11     else if (kok.sagCocuk == null)
12         return kok.solCocuk;  // yalnız sol çocuk
13     else {                    // iki çocuk
14         kok.veri = minDeger(kok.sagCocuk).veri;
15         kok.sagCocuk = sil(kok.sagCocuk, kok.veri);
16     }
17     return kok;
18 }
```

DEĞİŞKENLER

`kok = düğüm(60)` `veri = 70` `k = 1` `dönen = null`



Değişen bağ: **kok.sagCocuk = null**. Silinen düğüme artık hiçbir bağ yok; çöp toplayıcı temizler.

Adım 8 / 11

ÇAĞRI YIĞINI

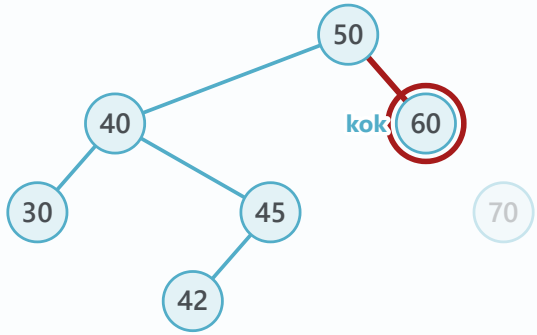
- `sil(60, 70)`
- `sil(50, 70)`
- `main()`

Kod İzleme: `sil(kok, 70)` — yaprak

```
1  IkiliAgacDugum<E> sil(
2          IkiliAgacDugum<E> kok, E veri) {
3      if (kok == null) return null;   // yok
4      int k = veri.compareTo(kok.veri);
5      if (k < 0)
6          kok.solCocuk = sil(kok.solCocuk, veri);
7      else if (k > 0)
8          kok.sagCocuk = sil(kok.sagCocuk, veri);
9      else if (kok.solCocuk == null)
10         return kok.sagCocuk;  // yaprak/sağ çocuk
11     else if (kok.sagCocuk == null)
12         return kok.solCocuk;  // yalnız sol çocuk
13     else {                    // iki çocuk
14         kok.veri = minDeger(kok.sagCocuk).veri;
15         kok.sagCocuk = sil(kok.sagCocuk, kok.veri);
16     }
▶      return kok;
18 }
```

DEĞİŞKENLER

`kok = düğüm(60)` `veri = 70` `k = 1`



return kok: 60 kendi ebeveynine döndürülür.

Adım 9 / 11

ÇAĞRI YIĞINI

- `sil(60, 70)`
- `sil(50, 70)`
- `main()`

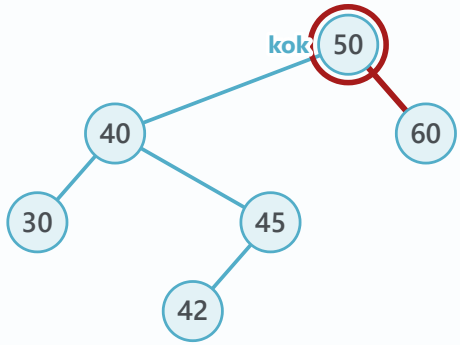
Kod İzleme: `sil(kok, 70)` — yaprak

```
1  IkiliAgacDugum<E> sil(
2          IkiliAgacDugum<E> kok, E veri) {
3      if (kok == null) return null;   // yok
4      int k = veri.compareTo(kok.veri);
5      if (k < 0)
6          kok.solCocuk = sil(kok.solCocuk, veri);
7      else if (k > 0)
▶          kok.sagCocuk = sil(kok.sagCocuk, veri);
9      else if (kok.solCocuk == null)
10         return kok.sagCocuk;  // yaprak/sağ çocuk
11     else if (kok.sagCocuk == null)
12         return kok.solCocuk;  // yalnız sol çocuk
13     else {                    // iki çocuk
14         kok.veri = minDeger(kok.sagCocuk).veri;
15         kok.sagCocuk = sil(kok.sagCocuk, kok.veri);
16     }
▶      return kok;
18 }
```

DEĞİŞKENLER

`kok = düğüm(50)` `veri = 70` `k = 1`

`dönen = düğüm(60)`



`kok.sagCocuk = düğüm(60)` : bağ zaten onu gösteriyordu, **değişmez**. Sonra **return kok** (50).

Adım 10 / 11

ÇAĞRI YIĞINI

`sil(50, 70)`

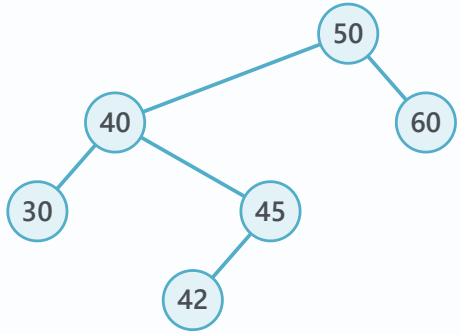
`main()`

Kod İzleme: `sil(kok, 70)` — yaprak

```
1  IkiliAgacDugum<E> sil(
2          IkiliAgacDugum<E> kok, E veri) {
3      if (kok == null) return null;   // yok
4      int k = veri.compareTo(kok.veri);
5      if (k < 0)
6          kok.solCocuk = sil(kok.solCocuk, veri);
7      else if (k > 0)
8          kok.sagCocuk = sil(kok.sagCocuk, veri);
9      else if (kok.solCocuk == null)
10         return kok.sagCocuk;  // yaprak/sağ çocuk
11     else if (kok.sagCocuk == null)
12         return kok.solCocuk;  // yalnız sol çocuk
13     else {                    // iki çocuk
14         kok.veri = minDeger(kok.sagCocuk).veri;
15         kok.sagCocuk = sil(kok.sagCocuk, kok.veri);
16     }
17     return kok;
18 }
```

DEĞİŞKENLER

`kok` = **düğüm(50)**



main: `kok = 50`. 70 silindi: yaprak olduğu için yalnızca 60'ın sağ bağı null oldu.

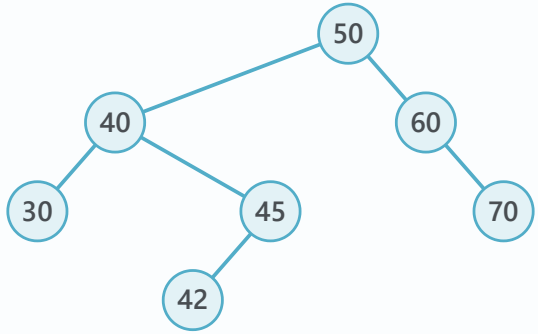
Adım 11 / 11

ÇAĞRI YIĞINI

`main()`

Kod İzleme: `sil(kok, 45)` — tek çocuk

```
▶ IkiliAgacDugum<E> sil(
2         IkiliAgacDugum<E> kok, E veri) {
3     if (kok == null) return null;   // yok
4     int k = veri.compareTo(kok.veri);
5     if (k < 0)
6         kok.solCocuk = sil(kok.solCocuk, veri);
7     else if (k > 0)
8         kok.sagCocuk = sil(kok.sagCocuk, veri);
9     else if (kok.solCocuk == null)
10        return kok.sagCocuk;  // yaprak/sağ çocuk
11    else if (kok.sagCocuk == null)
12        return kok.solCocuk;  // yalnız sol çocuk
13    else {                    // iki çocuk
14        kok.veri = minDeger(kok.sagCocuk).veri;
15        kok.sagCocuk = sil(kok.sagCocuk, kok.veri);
16    }
17    return kok;
18 }
```



main, `kok = sil(kok, 45)` çağırır. Önce silinecek düğüm aranır.

Adım 1 / 11

DEĞİŞKENLER

`veri = 45`

ÇAĞRI YIĞINI

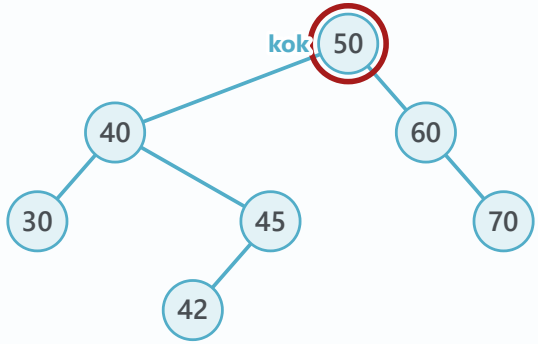
`main()`

Kod İzleme: `sil(kok, 45)` — tek çocuk

```
1  IkiliAgacDugum<E> sil(
2          IkiliAgacDugum<E> kok, E veri) {
▶      if (kok == null) return null;   // yok
▶      int k = veri.compareTo(kok.veri);
5      if (k < 0)
6          kok.solCocuk = sil(kok.solCocuk, veri);
7      else if (k > 0)
8          kok.sagCocuk = sil(kok.sagCocuk, veri);
9      else if (kok.solCocuk == null)
10         return kok.sagCocuk;  // yaprak/sağ çocuk
11     else if (kok.sagCocuk == null)
12         return kok.solCocuk;  // yalnız sol çocuk
13     else {                    // iki çocuk
14         kok.veri = minDeger(kok.sagCocuk).veri;
15         kok.sagCocuk = sil(kok.sagCocuk, kok.veri);
16     }
17     return kok;
18 }
```

DEĞİŞKENLER

`kok = düğüm(50)` `veri = 45` `k = -1`



kok ≠ null. **k = -1** çünkü 45 < 50.

Adım 2 / 11

ÇAĞRI YIĞINI

`sil(50, 45)`

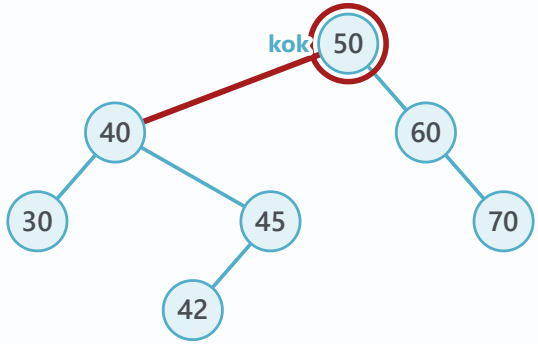
`main()`

Kod İzleme: `sil(kok, 45)` — tek çocuk

```
1  IkiliAgacDugum<E> sil(
2          IkiliAgacDugum<E> kok, E veri) {
3      if (kok == null) return null;   // yok
4      int k = veri.compareTo(kok.veri);
▶      if (k < 0)
▶          kok.solCocuk = sil(kok.solCocuk, veri);
7      else if (k > 0)
8          kok.sagCocuk = sil(kok.sagCocuk, veri);
9      else if (kok.solCocuk == null)
10         return kok.sagCocuk;  // yaprak/sağ çocuk
11     else if (kok.sagCocuk == null)
12         return kok.solCocuk;  // yalnız sol çocuk
13     else {                    // iki çocuk
14         kok.veri = minDeger(kok.sagCocuk).veri;
15         kok.sagCocuk = sil(kok.sagCocuk, kok.veri);
16     }
17     return kok;
18 }
```

DEĞİŞKENLER

`kok` = **düğüm(50)** | `veri` = **45** | `k` = **-1**



Sol alt ağaçta sil: `sil(kok.solCocuk, 45)`. Dönen değer **kok.solCocuk**'a atanacak.

Adım 3 / 11

ÇAĞRI YIĞINI

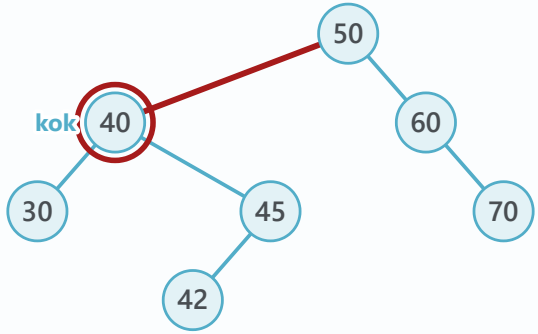
- `sil(50, 45)`
- `main()`

Kod İzleme: `sil(kok, 45)` — tek çocuk

```
1  IkiliAgacDugum<E> sil(
2          IkiliAgacDugum<E> kok, E veri) {
▶      if (kok == null) return null;   // yok
▶      int k = veri.compareTo(kok.veri);
5      if (k < 0)
6          kok.solCocuk = sil(kok.solCocuk, veri);
7      else if (k > 0)
8          kok.sagCocuk = sil(kok.sagCocuk, veri);
9      else if (kok.solCocuk == null)
10         return kok.sagCocuk;  // yaprak/sağ çocuk
11     else if (kok.sagCocuk == null)
12         return kok.solCocuk;  // yalnız sol çocuk
13     else {                    // iki çocuk
14         kok.veri = minDeger(kok.sagCocuk).veri;
15         kok.sagCocuk = sil(kok.sagCocuk, kok.veri);
16     }
17     return kok;
18 }
```

DEĞİŞKENLER

`kok = düğüm(40)` `veri = 45` `k = 1`



kok ≠ null. **k = 1** çünkü 45 > 40.

ÇAĞRI YIĞINI

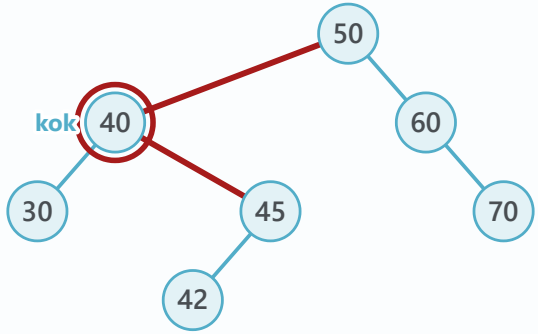
- `sil(40, 45)`
- `sil(50, 45)`
- `main()`

Kod İzleme: `sil(kok, 45)` — tek çocuk

```
1  IkiliAgacDugum<E> sil(
2          IkiliAgacDugum<E> kok, E veri) {
3      if (kok == null) return null;   // yok
4      int k = veri.compareTo(kok.veri);
5      if (k < 0)
6          kok.solCocuk = sil(kok.solCocuk, veri);
▶      else if (k > 0)
▶          kok.sagCocuk = sil(kok.sagCocuk, veri);
9      else if (kok.solCocuk == null)
10         return kok.sagCocuk;  // yaprak/sağ çocuk
11     else if (kok.sagCocuk == null)
12         return kok.solCocuk;  // yalnız sol çocuk
13     else {                    // iki çocuk
14         kok.veri = minDeger(kok.sagCocuk).veri;
15         kok.sagCocuk = sil(kok.sagCocuk, kok.veri);
16     }
17     return kok;
18 }
```

DEĞİŞKENLER

`kok = düğüm(40)` `veri = 45` `k = 1`



Sağ alt ağaçta sil: `sil(kok.sagCocuk, 45)`.
Dönen değer **kok.sagCocuk**'a atanacak.

Adım 5 / 11

ÇAĞRI YIĞINI

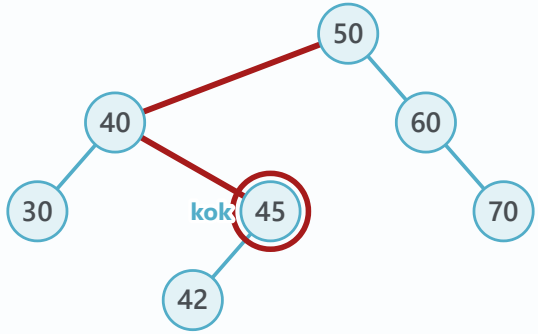
- `sil(40, 45)`
- `sil(50, 45)`
- `main()`

Kod İzleme: `sil(kok, 45)` — tek çocuk

```
1  IkiliAgacDugum<E> sil(
2          IkiliAgacDugum<E> kok, E veri) {
▶      if (kok == null) return null;   // yok
▶      int k = veri.compareTo(kok.veri);
5      if (k < 0)
6          kok.solCocuk = sil(kok.solCocuk, veri);
7      else if (k > 0)
8          kok.sagCocuk = sil(kok.sagCocuk, veri);
9      else if (kok.solCocuk == null)
10         return kok.sagCocuk;  // yaprak/sağ çocuk
11     else if (kok.sagCocuk == null)
12         return kok.solCocuk;  // yalnız sol çocuk
13     else {                    // iki çocuk
14         kok.veri = minDeger(kok.sagCocuk).veri;
15         kok.sagCocuk = sil(kok.sagCocuk, kok.veri);
16     }
17     return kok;
18 }
```

DEĞİŞKENLER

`kok = düğüm(45)` `veri = 45` `k = 0`



kok ≠ null. **k = 0** çünkü 45 = 45.

Adım 6 / 11

ÇAĞRI YIĞINI

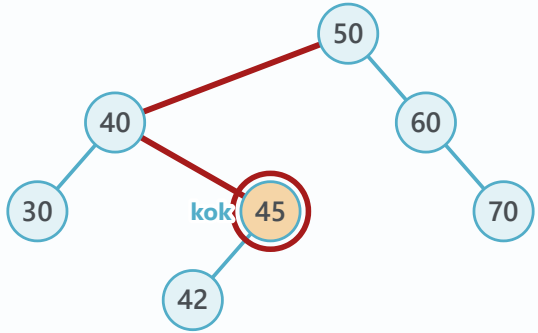
- `sil(45, 45)`
- `sil(40, 45)`
- `sil(50, 45)`
- `main()`

Kod İzleme: `sil(kok, 45)` — tek çocuk

```
1  IkiliAgacDugum<E> sil(
2          IkiliAgacDugum<E> kok, E veri) {
3      if (kok == null) return null;   // yok
4      int k = veri.compareTo(kok.veri);
5      if (k < 0)
6          kok.solCocuk = sil(kok.solCocuk, veri);
7      else if (k > 0)
8          kok.sagCocuk = sil(kok.sagCocuk, veri);
▶      else if (kok.solCocuk == null)
10         return kok.sagCocuk;  // yaprak/sağ çocuk
▶      else if (kok.sagCocuk == null)
▶          return kok.solCocuk;  // yalnız sol çocuk
13     else {                    // iki çocuk
14         kok.veri = minDeger(kok.sagCocuk).veri;
15         kok.sagCocuk = sil(kok.sagCocuk, kok.veri);
16     }
17     return kok;
18 }
```

DEĞİŞKENLER

`kok = düğüm(45)` `veri = 45` `k = 0`



k = 0 → bulundu. Sol çocuk var, **sağ çocuk yok** → **return kok.solCocuk** = düğüm(42): tek çocuk ebeveyne verilir.

Adım 7 / 11

ÇAĞRI YIĞINI

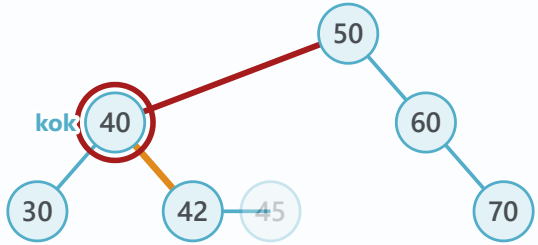
- `sil(45, 45)`
- `sil(40, 45)`
- `sil(50, 45)`
- `main()`

Kod İzleme: `sil(kok, 45)` — tek çocuk

```
1  IkiliAgacDugum<E> sil(
2          IkiliAgacDugum<E> kok, E veri) {
3      if (kok == null) return null;   // yok
4      int k = veri.compareTo(kok.veri);
5      if (k < 0)
6          kok.solCocuk = sil(kok.solCocuk, veri);
7      else if (k > 0)
▶          kok.sagCocuk = sil(kok.sagCocuk, veri);
9      else if (kok.solCocuk == null)
10         return kok.sagCocuk;  // yaprak/sağ çocuk
11     else if (kok.sagCocuk == null)
12         return kok.solCocuk;  // yalnız sol çocuk
13     else {                    // iki çocuk
14         kok.veri = minDeger(kok.sagCocuk).veri;
15         kok.sagCocuk = sil(kok.sagCocuk, kok.veri);
16     }
17     return kok;
18 }
```

DEĞİŞKENLER

`kok` = **düğüm(40)** | `veri` = **45** | `k` = **1** | `dönen` = **düğüm(42)**



Değişen bağ: **kok.sagCocuk = 42**. Ebeveyn, silinen düğümü atlayıp doğrudan onun çocuğuna bağlandı.

Adım 8 / 11

ÇAĞRI YIĞINI

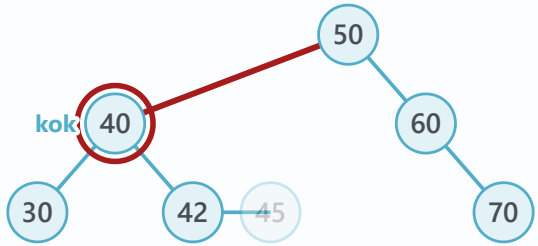
- `sil(40, 45)`
- `sil(50, 45)`
- `main()`

Kod İzleme: `sil(kok, 45)` — tek çocuk

```
1  IkiliAgacDugum<E> sil(
2          IkiliAgacDugum<E> kok, E veri) {
3      if (kok == null) return null;   // yok
4      int k = veri.compareTo(kok.veri);
5      if (k < 0)
6          kok.solCocuk = sil(kok.solCocuk, veri);
7      else if (k > 0)
8          kok.sagCocuk = sil(kok.sagCocuk, veri);
9      else if (kok.solCocuk == null)
10         return kok.sagCocuk;  // yaprak/sağ çocuk
11     else if (kok.sagCocuk == null)
12         return kok.solCocuk;  // yalnız sol çocuk
13     else {                    // iki çocuk
14         kok.veri = minDeger(kok.sagCocuk).veri;
15         kok.sagCocuk = sil(kok.sagCocuk, kok.veri);
16     }
▶      return kok;
18 }
```

DEĞİŞKENLER

`kok = düğüm(40)` `veri = 45` `k = 1`



return kok: 40 kendi ebeveynine döndürülür.

Adım 9 / 11

ÇAĞRI YIĞINI

- `sil(40, 45)`
- `sil(50, 45)`
- `main()`

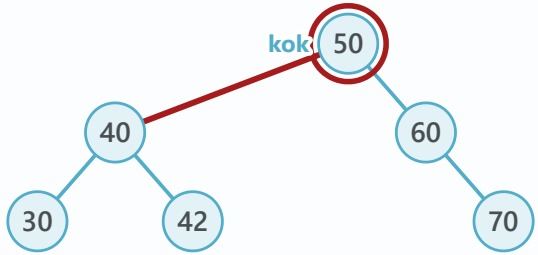
Kod İzleme: `sil(kok, 45)` — tek çocuk

```
1  IkiliAgacDugum<E> sil(
2          IkiliAgacDugum<E> kok, E veri) {
3      if (kok == null) return null;   // yok
4      int k = veri.compareTo(kok.veri);
5      if (k < 0)
▶          kok.solCocuk = sil(kok.solCocuk, veri);
7      else if (k > 0)
8          kok.sagCocuk = sil(kok.sagCocuk, veri);
9      else if (kok.solCocuk == null)
10         return kok.sagCocuk;  // yaprak/sağ çocuk
11     else if (kok.sagCocuk == null)
12         return kok.solCocuk;  // yalnız sol çocuk
13     else {                    // iki çocuk
14         kok.veri = minDeger(kok.sagCocuk).veri;
15         kok.sagCocuk = sil(kok.sagCocuk, kok.veri);
16     }
▶      return kok;
18 }
```

DEĞİŞKENLER

`kok = düğüm(50)` `veri = 45` `k = -1`

`dönen = düğüm(40)`



`kok.solCocuk = düğüm(40)`: bağ zaten onu gösteriyordu, **değişmez**. Sonra **return kok** (50).

Adım 10 / 11

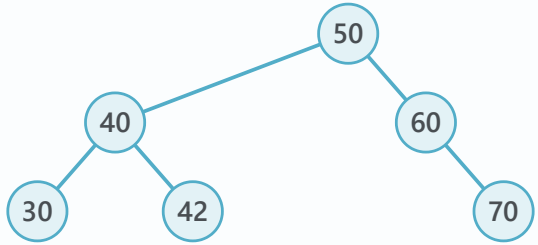
ÇAĞRI YIĞINI

`sil(50, 45)`

`main()`

Kod İzleme: `sil(kok, 45)` — tek çocuk

```
1  IkiliAgacDugum<E> sil(
2          IkiliAgacDugum<E> kok, E veri) {
3      if (kok == null) return null;   // yok
4      int k = veri.compareTo(kok.veri);
5      if (k < 0)
6          kok.solCocuk = sil(kok.solCocuk, veri);
7      else if (k > 0)
8          kok.sagCocuk = sil(kok.sagCocuk, veri);
9      else if (kok.solCocuk == null)
10         return kok.sagCocuk;  // yaprak/sağ çocuk
11     else if (kok.sagCocuk == null)
12         return kok.solCocuk;  // yalnız sol çocuk
13     else {                    // iki çocuk
14         kok.veri = minDeger(kok.sagCocuk).veri;
15         kok.sagCocuk = sil(kok.sagCocuk, kok.veri);
16     }
17     return kok;
18 }
```



main: `kok = 50`. 45 silindi: tek çocuğu 42, 40'ın sağ çocuğu oldu. İnorder sıra korunur.

Adım 11 / 11

DEĞİŞKENLER

`kok = düğüm(50)`

ÇAĞRI YIĞINI

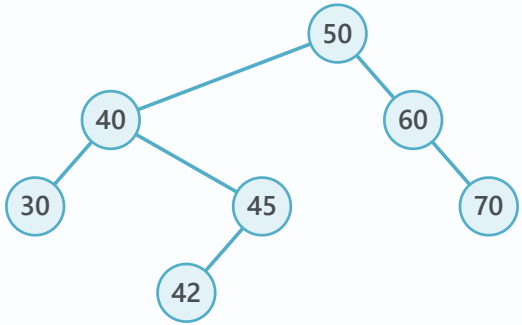
`main()`

Kod İzleme: `sil(kok, 40)` — iki çocuk

```
▶ IkiliAgacDugum<E> sil(
2         IkiliAgacDugum<E> kok, E veri) {
3     if (kok == null) return null;   // yok
4     int k = veri.compareTo(kok.veri);
5     if (k < 0)
6         kok.solCocuk = sil(kok.solCocuk, veri);
7     else if (k > 0)
8         kok.sagCocuk = sil(kok.sagCocuk, veri);
9     else if (kok.solCocuk == null)
10        return kok.sagCocuk;  // yaprak/sağ çocuk
11    else if (kok.sagCocuk == null)
12        return kok.solCocuk;  // yalnız sol çocuk
13    else {                    // iki çocuk
14        kok.veri = minDeger(kok.sagCocuk).veri;
15        kok.sagCocuk = sil(kok.sagCocuk, kok.veri);
16    }
17    return kok;
18 }
```

DEĞİŞKENLER

`veri = 40`



main, `kok = sil(kok, 40)` çağırır. Önce silinecek düğüm aranır.

Adım 1 / 17

ÇAĞRI YIĞINI

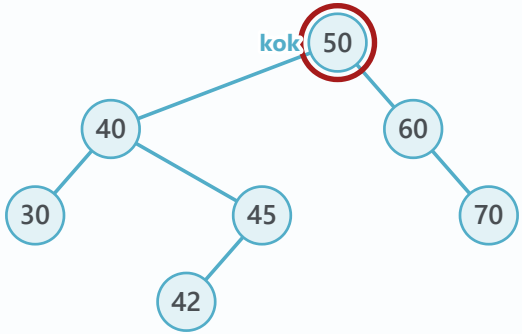
`main()`

Kod İzleme: `sil(kok, 40)` — iki çocuk

```
1  IkiliAgacDugum<E> sil(
2          IkiliAgacDugum<E> kok, E veri) {
▶      if (kok == null) return null;   // yok
▶      int k = veri.compareTo(kok.veri);
5      if (k < 0)
6          kok.solCocuk = sil(kok.solCocuk, veri);
7      else if (k > 0)
8          kok.sagCocuk = sil(kok.sagCocuk, veri);
9      else if (kok.solCocuk == null)
10         return kok.sagCocuk;  // yaprak/sağ çocuk
11     else if (kok.sagCocuk == null)
12         return kok.solCocuk;  // yalnız sol çocuk
13     else {                    // iki çocuk
14         kok.veri = minDeger(kok.sagCocuk).veri;
15         kok.sagCocuk = sil(kok.sagCocuk, kok.veri);
16     }
17     return kok;
18 }
```

DEĞİŞKENLER

`kok = düğüm(50)` `veri = 40` `k = -1`



kok ≠ null. **k = -1** çünkü 40 < 50.

Adım 2 / 17

ÇAĞRI YIĞINI

`sil(50, 40)`

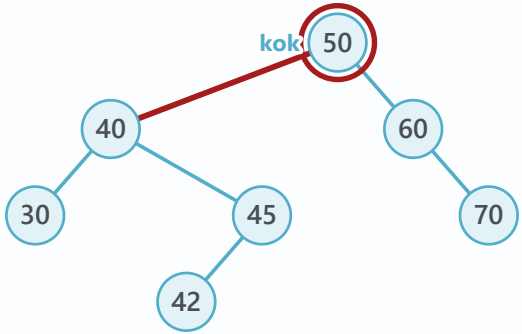
`main()`

Kod İzleme: `sil(kok, 40)` — iki çocuk

```
1  IkiliAgacDugum<E> sil(
2          IkiliAgacDugum<E> kok, E veri) {
3      if (kok == null) return null;   // yok
4      int k = veri.compareTo(kok.veri);
▶      if (k < 0)
▶          kok.solCocuk = sil(kok.solCocuk, veri);
7      else if (k > 0)
8          kok.sagCocuk = sil(kok.sagCocuk, veri);
9      else if (kok.solCocuk == null)
10         return kok.sagCocuk;  // yaprak/sağ çocuk
11     else if (kok.sagCocuk == null)
12         return kok.solCocuk;  // yalnız sol çocuk
13     else {                    // iki çocuk
14         kok.veri = minDeger(kok.sagCocuk).veri;
15         kok.sagCocuk = sil(kok.sagCocuk, kok.veri);
16     }
17     return kok;
18 }
```

DEĞİŞKENLER

`kok` = **düğüm(50)** | `veri` = **40** | `k` = **−1**



Sol alt ağaçta sil: `sil(kok.solCocuk, 40)`.
Dönen değer **kok.solCocuk**'a atanacak.

Adım 3 / 17

ÇAĞRI YIĞINI

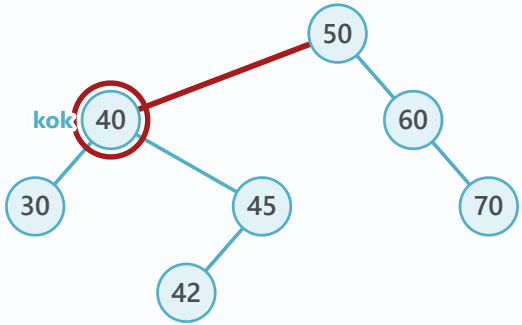
- `sil(50, 40)`
- `main()`

Kod İzleme: `sil(kok, 40)` — iki çocuk

```
1  IkiliAgacDugum<E> sil(
2          IkiliAgacDugum<E> kok, E veri) {
▶      if (kok == null) return null;   // yok
▶      int k = veri.compareTo(kok.veri);
5      if (k < 0)
6          kok.solCocuk = sil(kok.solCocuk, veri);
7      else if (k > 0)
8          kok.sagCocuk = sil(kok.sagCocuk, veri);
9      else if (kok.solCocuk == null)
10         return kok.sagCocuk;  // yaprak/sağ çocuk
11     else if (kok.sagCocuk == null)
12         return kok.solCocuk;  // yalnız sol çocuk
13     else {                    // iki çocuk
14         kok.veri = minDeger(kok.sagCocuk).veri;
15         kok.sagCocuk = sil(kok.sagCocuk, kok.veri);
16     }
17     return kok;
18 }
```

DEĞİŞKENLER

`kok = düğüm(40)` `veri = 40` `k = 0`



kok ≠ null. **k = 0** çünkü 40 = 40.

ÇAĞRI YIĞINI

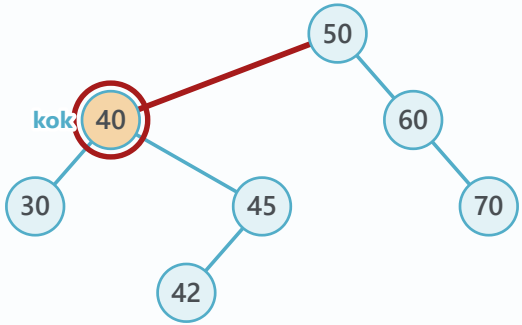
- `sil(40, 40)`
- `sil(50, 40)`
- `main()`

Kod İzleme: `sil(kok, 40)` — iki çocuk

```
1  IkiliAgacDugum<E> sil(
2          IkiliAgacDugum<E> kok, E veri) {
3      if (kok == null) return null;   // yok
4      int k = veri.compareTo(kok.veri);
5      if (k < 0)
6          kok.solCocuk = sil(kok.solCocuk, veri);
7      else if (k > 0)
8          kok.sagCocuk = sil(kok.sagCocuk, veri);
▶      else if (kok.solCocuk == null)
10         return kok.sagCocuk;  // yaprak/sağ çocuk
▶      else if (kok.sagCocuk == null)
12         return kok.solCocuk;  // yalnız sol çocuk
▶      else {                    // iki çocuk
14         kok.veri = minDeger(kok.sagCocuk).veri;
15         kok.sagCocuk = sil(kok.sagCocuk, kok.veri);
16     }
17     return kok;
18 }
```

DEĞİŞKENLER

`kok = düğüm(40)` `veri = 40` `k = 0`



k = 0 → bulundu, ama **iki çocuğu var** (30 ve 45): doğrudan çıkarılamaz.

Adım 5 / 17

ÇAĞRI YIĞINI

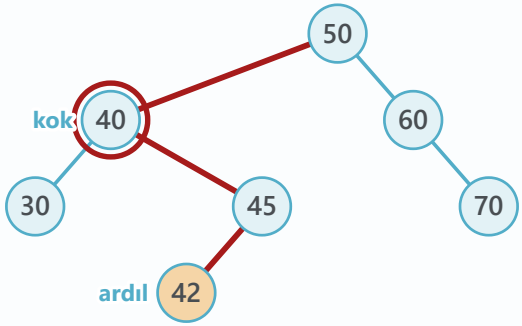
- `sil(40, 40)`
- `sil(50, 40)`
- `main()`

Kod İzleme: `sil(kok, 40)` — iki çocuk

```
1  IkiliAgacDugum<E> sil(
2          IkiliAgacDugum<E> kok, E veri) {
3      if (kok == null) return null;   // yok
4      int k = veri.compareTo(kok.veri);
5      if (k < 0)
6          kok.solCocuk = sil(kok.solCocuk, veri);
7      else if (k > 0)
8          kok.sagCocuk = sil(kok.sagCocuk, veri);
9      else if (kok.solCocuk == null)
10         return kok.sagCocuk;  // yaprak/sağ çocuk
11     else if (kok.sagCocuk == null)
12         return kok.solCocuk;  // yalnız sol çocuk
13     else {                    // iki çocuk
▶          kok.veri = minDeger(kok.sagCocuk).veri;
15         kok.sagCocuk = sil(kok.sagCocuk, kok.veri);
16     }
17     return kok;
18 }
```

DEĞİŞKENLER

`kok` = **düğüm(40)** | `veri` = **40** | `ardıl` = **düğüm(42)**



`minDeger(kok.sagCocuk)` : sağa bir adım (45), sonra hep sola → **ardıl = 42** (sağ alt ağacın en küçüğü).

Adım 6 / 17

ÇAĞRI YIĞINI

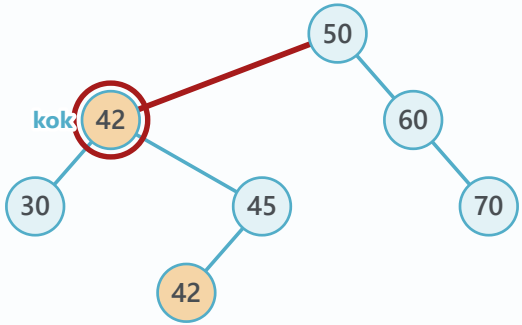
- `minDeger(45)`
- `sil(40, 40)`
- `sil(50, 40)`
- `main()`

Kod İzleme: `sil(kok, 40)` — iki çocuk

```
1  IkiliAgacDugum<E> sil(
2          IkiliAgacDugum<E> kok, E veri) {
3      if (kok == null) return null;   // yok
4      int k = veri.compareTo(kok.veri);
5      if (k < 0)
6          kok.solCocuk = sil(kok.solCocuk, veri);
7      else if (k > 0)
8          kok.sagCocuk = sil(kok.sagCocuk, veri);
9      else if (kok.solCocuk == null)
10         return kok.sagCocuk;  // yaprak/sağ çocuk
11     else if (kok.sagCocuk == null)
12         return kok.solCocuk;  // yalnız sol çocuk
13     else {                    // iki çocuk
▶          kok.veri = minDeger(kok.sagCocuk).veri;
15         kok.sagCocuk = sil(kok.sagCocuk, kok.veri);
16     }
17     return kok;
18 }
```

DEĞİŞKENLER

`kok = düğüm(42)` `veri = 40`



kok.veri = 42: ardılın değeri silinecek düğüme kopyalanır. Şimdi 42 iki kez var; alttakini silmeliyiz.

Adım 7 / 17

ÇAĞRI YIĞINI

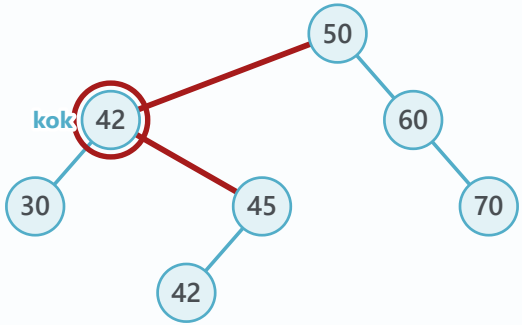
- `sil(40, 40)`
- `sil(50, 40)`
- `main()`

Kod İzleme: `sil(kok, 40)` — iki çocuk

```
1  IkiliAgacDugum<E> sil(
2          IkiliAgacDugum<E> kok, E veri) {
3      if (kok == null) return null;   // yok
4      int k = veri.compareTo(kok.veri);
5      if (k < 0)
6          kok.solCocuk = sil(kok.solCocuk, veri);
7      else if (k > 0)
8          kok.sagCocuk = sil(kok.sagCocuk, veri);
9      else if (kok.solCocuk == null)
10         return kok.sagCocuk;  // yaprak/sağ çocuk
11     else if (kok.sagCocuk == null)
12         return kok.solCocuk;  // yalnız sol çocuk
13     else {                    // iki çocuk
14         kok.veri = minDeger(kok.sagCocuk).veri;
▶          kok.sagCocuk = sil(kok.sagCocuk, kok.veri);
16     }
17     return kok;
18 }
```

DEĞİŞKENLER

`kok` = **düğüm(42)** `veri` = **40**



`sil(kok.sagCocuk, 42)` : eski 42 sağ alt ağaçtan silinir. Ardılın sol çocuğu olmadığından bu hep yaprak/tek çocuk durumudur.

Adım 8 / 17

ÇAĞRI YIĞINI

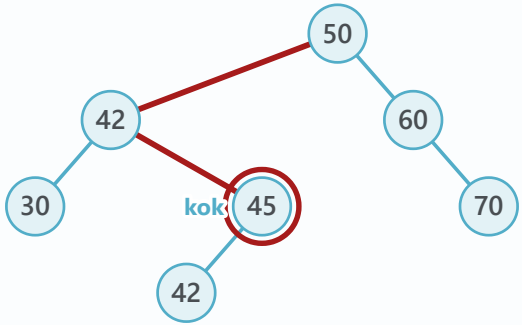
- **`sil(40, 40)`**
- `sil(50, 40)`
- `main()`

Kod İzleme: `sil(kok, 40)` — iki çocuk

```
1  IkiliAgacDugum<E> sil(
2          IkiliAgacDugum<E> kok, E veri) {
▶      if (kok == null) return null;   // yok
▶      int k = veri.compareTo(kok.veri);
5      if (k < 0)
6          kok.solCocuk = sil(kok.solCocuk, veri);
7      else if (k > 0)
8          kok.sagCocuk = sil(kok.sagCocuk, veri);
9      else if (kok.solCocuk == null)
10         return kok.sagCocuk;  // yaprak/sağ çocuk
11     else if (kok.sagCocuk == null)
12         return kok.solCocuk;  // yalnız sol çocuk
13     else {                    // iki çocuk
14         kok.veri = minDeger(kok.sagCocuk).veri;
15         kok.sagCocuk = sil(kok.sagCocuk, kok.veri);
16     }
17     return kok;
18 }
```

DEĞİŞKENLER

`kok = düğüm(45)` `veri = 42` `k = -1`



kok ≠ null. **k = -1** çünkü 42 < 45.

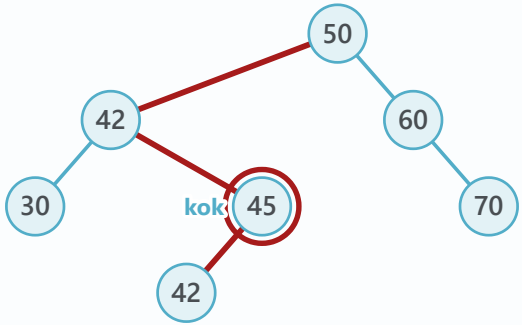
Adım 9 / 17

ÇAĞRI YIĞINI

- `sil(45, 42)`
- `sil(40, 40)`
- `sil(50, 40)`
- `main()`

Kod İzleme: `sil(kok, 40)` — iki çocuk

```
1  IkiliAgacDugum<E> sil(
2          IkiliAgacDugum<E> kok, E veri) {
3      if (kok == null) return null;   // yok
4      int k = veri.compareTo(kok.veri);
▶      if (k < 0)
▶          kok.solCocuk = sil(kok.solCocuk, veri);
7      else if (k > 0)
8          kok.sagCocuk = sil(kok.sagCocuk, veri);
9      else if (kok.solCocuk == null)
10         return kok.sagCocuk;  // yaprak/sağ çocuk
11     else if (kok.sagCocuk == null)
12         return kok.solCocuk;  // yalnız sol çocuk
13     else {                    // iki çocuk
14         kok.veri = minDeger(kok.sagCocuk).veri;
15         kok.sagCocuk = sil(kok.sagCocuk, kok.veri);
16     }
17     return kok;
18 }
```



Sol alt ağaçta sil: `sil(kok.solCocuk, 42)`. Dönen değer **kok.solCocuk**'a atanacak.

Adım 10 / 17

DEĞİŞKENLER

`kok` = **düğüm(45)** | `veri` = **42** | `k` = **−1**

ÇAĞRI YIĞINI

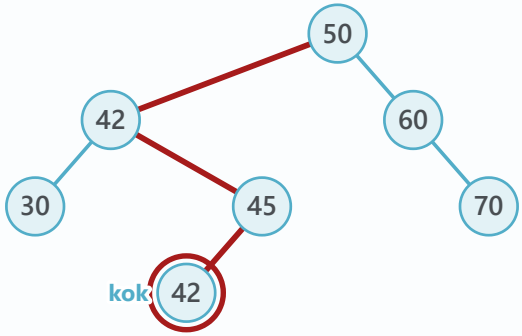
- **`sil(45, 42)`**
- `sil(40, 40)`
- `sil(50, 40)`
- `main()`

Kod İzleme: `sil(kok, 40)` — iki çocuk

```
1  IkiliAgacDugum<E> sil(
2          IkiliAgacDugum<E> kok, E veri) {
▶      if (kok == null) return null;   // yok
▶      int k = veri.compareTo(kok.veri);
5      if (k < 0)
6          kok.solCocuk = sil(kok.solCocuk, veri);
7      else if (k > 0)
8          kok.sagCocuk = sil(kok.sagCocuk, veri);
9      else if (kok.solCocuk == null)
10         return kok.sagCocuk;  // yaprak/sağ çocuk
11     else if (kok.sagCocuk == null)
12         return kok.solCocuk;  // yalnız sol çocuk
13     else {                    // iki çocuk
14         kok.veri = minDeger(kok.sagCocuk).veri;
15         kok.sagCocuk = sil(kok.sagCocuk, kok.veri);
16     }
17     return kok;
18 }
```

DEĞİŞKENLER

`kok = düğüm(42)` `veri = 42` `k = 0`



kok ≠ null. **k = 0** çünkü 42 = 42.

Adım 11 / 17

ÇAĞRI YIĞINI

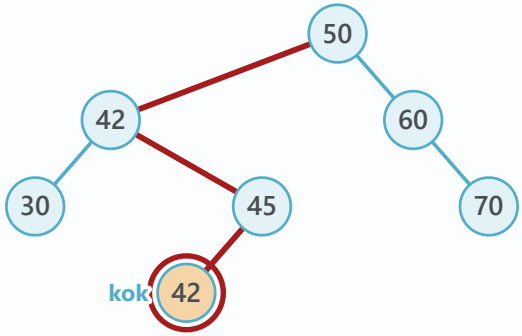
- `sil(42, 42)`
- `sil(45, 42)`
- `sil(40, 40)`
- `sil(50, 40)`
- `main()`

Kod İzleme: `sil(kok, 40)` — iki çocuk

```
1  IkiliAgacDugum<E> sil(
2          IkiliAgacDugum<E> kok, E veri) {
3      if (kok == null) return null;   // yok
4      int k = veri.compareTo(kok.veri);
5      if (k < 0)
6          kok.solCocuk = sil(kok.solCocuk, veri);
7      else if (k > 0)
8          kok.sagCocuk = sil(kok.sagCocuk, veri);
▶      else if (kok.solCocuk == null)
▶          return kok.sagCocuk;  // yaprak/sağ çocuk
11     else if (kok.sagCocuk == null)
12         return kok.solCocuk;  // yalnız sol çocuk
13     else {                    // iki çocuk
14         kok.veri = minDeger(kok.sagCocuk).veri;
15         kok.sagCocuk = sil(kok.sagCocuk, kok.veri);
16     }
17     return kok;
18 }
```

DEĞİŞKENLER

`kok = düğüm(42)` `veri = 42` `k = 0`



k = 0 → silinecek düğüm bulundu. Sol çocuk yok → **return kok.sagCocuk** = null. (Yaprak: null döner.)

Adım 12 / 17

ÇAĞRI YIĞINI

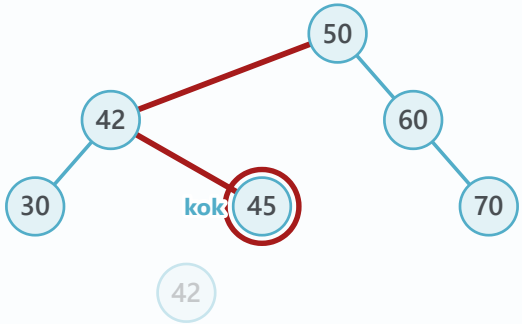
- `sil(42, 42)`
- `sil(45, 42)`
- `sil(40, 40)`
- `sil(50, 40)`
- `main()`

Kod İzleme: `sil(kok, 40)` — iki çocuk

```
1  IkiliAgacDugum<E> sil(
2          IkiliAgacDugum<E> kok, E veri) {
3      if (kok == null) return null;   // yok
4      int k = veri.compareTo(kok.veri);
5      if (k < 0)
▶          kok.solCocuk = sil(kok.solCocuk, veri);
7      else if (k > 0)
8          kok.sagCocuk = sil(kok.sagCocuk, veri);
9      else if (kok.solCocuk == null)
10         return kok.sagCocuk;  // yaprak/sağ çocuk
11     else if (kok.sagCocuk == null)
12         return kok.solCocuk;  // yalnız sol çocuk
13     else {                    // iki çocuk
14         kok.veri = minDeger(kok.sagCocuk).veri;
15         kok.sagCocuk = sil(kok.sagCocuk, kok.veri);
16     }
17     return kok;
18 }
```

DEĞİŞKENLER

`kok = düğüm(45)` `veri = 42` `k = -1` `dönen = null`



Değişen bağ: **kok.solCocuk = null**. Silinen düğüme artık hiçbir bağ yok; çöp toplayıcı temizler.

Adım 13 / 17

ÇAĞRI YIĞINI

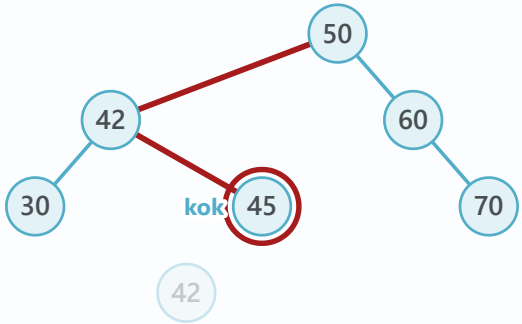
- `sil(45, 42)`
- `sil(40, 40)`
- `sil(50, 40)`
- `main()`

Kod İzleme: `sil(kok, 40)` — iki çocuk

```
1  IkiliAgacDugum<E> sil(
2          IkiliAgacDugum<E> kok, E veri) {
3      if (kok == null) return null;   // yok
4      int k = veri.compareTo(kok.veri);
5      if (k < 0)
6          kok.solCocuk = sil(kok.solCocuk, veri);
7      else if (k > 0)
8          kok.sagCocuk = sil(kok.sagCocuk, veri);
9      else if (kok.solCocuk == null)
10         return kok.sagCocuk;  // yaprak/sağ çocuk
11     else if (kok.sagCocuk == null)
12         return kok.solCocuk;  // yalnız sol çocuk
13     else {                    // iki çocuk
14         kok.veri = minDeger(kok.sagCocuk).veri;
15         kok.sagCocuk = sil(kok.sagCocuk, kok.veri);
16     }
▶      return kok;
18 }
```

DEĞİŞKENLER

`kok` = **düğüm(45)** | `veri` = **42** | `k` = **−1**



return kok: 45 kendi ebeveynine döndürülür.

Adım 14 / 17

ÇAĞRI YIĞINI

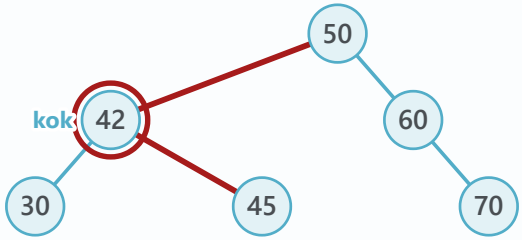
- **sil(45, 42)**
- sil(40, 40)
- sil(50, 40)
- main()

Kod İzleme: `sil(kok, 40)` — iki çocuk

```
1  IkiliAgacDugum<E> sil(
2          IkiliAgacDugum<E> kok, E veri) {
3      if (kok == null) return null;   // yok
4      int k = veri.compareTo(kok.veri);
5      if (k < 0)
6          kok.solCocuk = sil(kok.solCocuk, veri);
7      else if (k > 0)
8          kok.sagCocuk = sil(kok.sagCocuk, veri);
9      else if (kok.solCocuk == null)
10         return kok.sagCocuk;  // yaprak/sağ çocuk
11     else if (kok.sagCocuk == null)
12         return kok.solCocuk;  // yalnız sol çocuk
13     else {                    // iki çocuk
14         kok.veri = minDeger(kok.sagCocuk).veri;
▶          kok.sagCocuk = sil(kok.sagCocuk, kok.veri);
16     }
▶      return kok;
18 }
```

DEĞİŞKENLER

`kok = düğüm(42)` `veri = 40` `dönen = düğüm(45)`



`kok.sagCocuk = düğüm(45)` (aynı bağ), **return kok**. Silinen değer 40'ın yerini 42 aldı.

Adım 15 / 17

ÇAĞRI YIĞINI

- `sil(40, 40)`
- `sil(50, 40)`
- `main()`

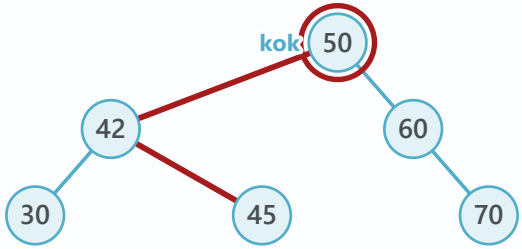
Kod İzleme: `sil(kok, 40)` — iki çocuk

```
1  IkiliAgacDugum<E> sil(
2          IkiliAgacDugum<E> kok, E veri) {
3      if (kok == null) return null;   // yok
4      int k = veri.compareTo(kok.veri);
5      if (k < 0)
▶          kok.solCocuk = sil(kok.solCocuk, veri);
7      else if (k > 0)
8          kok.sagCocuk = sil(kok.sagCocuk, veri);
9      else if (kok.solCocuk == null)
10         return kok.sagCocuk;  // yaprak/sağ çocuk
11     else if (kok.sagCocuk == null)
12         return kok.solCocuk;  // yalnız sol çocuk
13     else {                    // iki çocuk
14         kok.veri = minDeger(kok.sagCocuk).veri;
15         kok.sagCocuk = sil(kok.sagCocuk, kok.veri);
16     }
▶      return kok;
18 }
```

DEĞİŞKENLER

`kok = düğüm(50)` `veri = 40` `k = -1`

`dönen = düğüm(42)`



`kok.solCocuk = düğüm(42)` : bağ zaten onu gösteriyordu, **değişmez**. Sonra **return kok** (50).

Adım 16 / 17

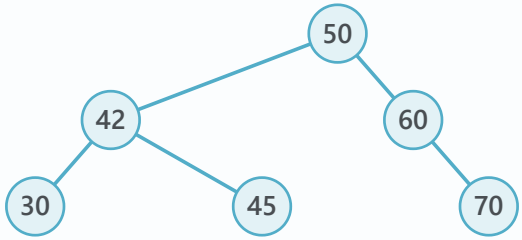
ÇAĞRI YIĞINI

`sil(50, 40)`

`main()`

Kod İzleme: `sil(kok, 40)` — iki çocuk

```
1  IkiliAgacDugum<E> sil(
2          IkiliAgacDugum<E> kok, E veri) {
3      if (kok == null) return null;   // yok
4      int k = veri.compareTo(kok.veri);
5      if (k < 0)
6          kok.solCocuk = sil(kok.solCocuk, veri);
7      else if (k > 0)
8          kok.sagCocuk = sil(kok.sagCocuk, veri);
9      else if (kok.solCocuk == null)
10         return kok.sagCocuk;  // yaprak/sağ çocuk
11     else if (kok.sagCocuk == null)
12         return kok.solCocuk;  // yalnız sol çocuk
13     else {                    // iki çocuk
14         kok.veri = minDeger(kok.sagCocuk).veri;
15         kok.sagCocuk = sil(kok.sagCocuk, kok.veri);
16     }
17     return kok;
18 }
```



main: `kok = 50`. 40 silindi: yerine ardılı 42 geçti; düğüm sayısı azaldı, BST özelliği korundu. Maliyet O(h).

DEĞİŞKENLER

`kok = düğüm(50)`

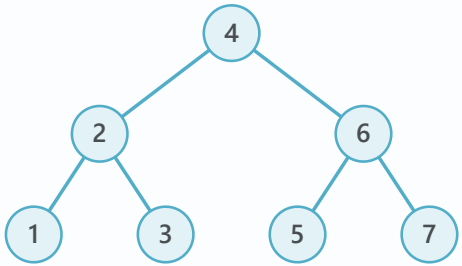
ÇAĞRI YIĞINI

`main()`

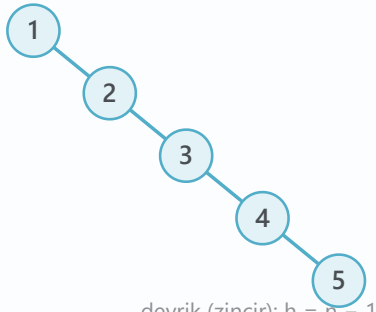
BST İŞLEMLERİ

İkili Arama Ağacı İşlemlerinin Analizi

- Arama, ekleme ve silmenin maliyeti, **erişilen en derin düğümün derinliğiyle** orantılıdır: O(h).



dengeli: h = 2 ≈ $\log_2 n$

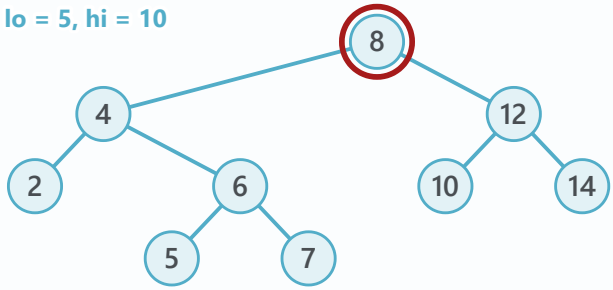


devrik (zincir): h = n − 1

Durum	Yükseklik h	Arama / Ekleme / Silme	n = 10^6 için adım
Dengeli (en iyi)	≈ $\log_2 n$	O(log n)	≈ 20
Rastgele sıra (ortalama)	≈ 2,99 $\log_2 n$	O(log n)	≈ 28 (ortalama)
Devrik (en kötü, sıralı girdi)	n − 1	O(n)	10^6

Rastgele sırada bir düğümün **ortalama derinliği** ≈ 2 ln n ≈ 1,39 $\log_2 n$'dir (n = 10^6 için ≈ 28 adım); **beklenen yükseklik** ise ≈ 4,31 ln n ≈ 2,99 $\log_2 n$'dir (≈ 60).

Kod İzleme: aralık sorgusu `aralik(kok, 5, 10)`



`aralik(8)` : 5 < 8 → solda aralığa giren anahtar olabilir: `aralik(k.sol)` .

Adım 1 / 19

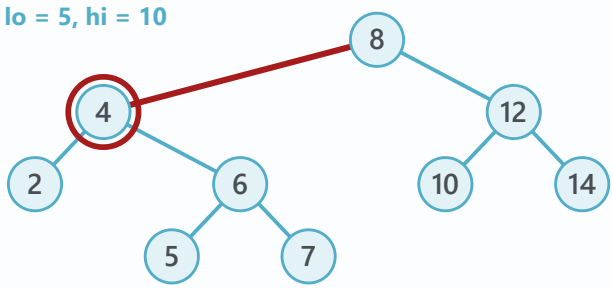
```
1 void aralik(AgacDugumu k, int lo, int hi) {
▶     if (k == null) return;
▶     if (lo < k.veri)           // solda aday var
▶         aralik(k.sol, lo, hi);
5     if (lo <= k.veri && k.veri <= hi)
6         System.out.print(k.veri + " ");
7     if (k.veri < hi)           // sağda aday var
8         aralik(k.sag, lo, hi);
9 }
```

ÇAĞRI YIĞINI

- `aralik(8)`
- `main()`

ÇIKTI

Kod İzleme: aralık sorgusu `aralik(kok, 5, 10)`



`aralik(4)` : 5 < 4 yanlış → sol alt ağaçtaki her anahtar < 4 ≤ lo: **sol alt ağaç budanır** (hiç ziyaret edilmez).

Adım 2 / 19

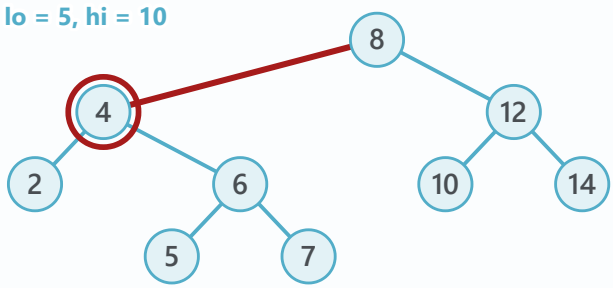
```
1 void aralik(AgacDugumu k, int lo, int hi) {
▶       if (k == null) return;
▶       if (lo < k.veri)           // solda aday var
4           aralik(k.sol, lo, hi);
5       if (lo <= k.veri && k.veri <= hi)
6           System.out.print(k.veri + " ");
7       if (k.veri < hi)           // sağda aday var
8           aralik(k.sag, lo, hi);
9   }
```

ÇAĞRI YIĞINI

- **`aralik(4)`**
- `aralik(8)`
- `main()`

ÇIKTI

Kod İzleme: aralık sorgusu `aralik(kok, 5, 10)`



Sol alt ağaç budandı (4 ≤ lo = 5), çağrı yapılmadı; `aralik(4)` sürer. 4 aralık dışında, yazdırılmaz. 4 < 10 → sağa da bak.

Adım 3 / 19

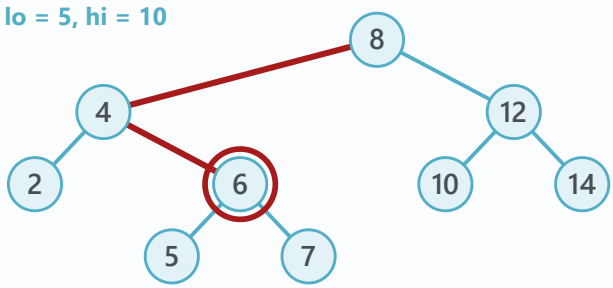
```
1 void aralik(AgacDugumu k, int lo, int hi) {
2     if (k == null) return;
3     if (lo < k.veri)           // solda aday var
4         aralik(k.sol, lo, hi);
▶     if (lo <= k.veri && k.veri <= hi)
6         System.out.print(k.veri + " ");
▶     if (k.veri < hi)           // sağda aday var
▶         aralik(k.sag, lo, hi);
9 }
```

ÇAĞRI YIĞINI

- **`aralik(4)`**
- `aralik(8)`
- `main()`

ÇIKTI

Kod İzleme: aralık sorgusu `aralik(kok, 5, 10)`



`aralik(6)` : 5 < 6 → solda aralığa giren anahtar olabilir: `aralik(k.sol)` .

Adım 4 / 19

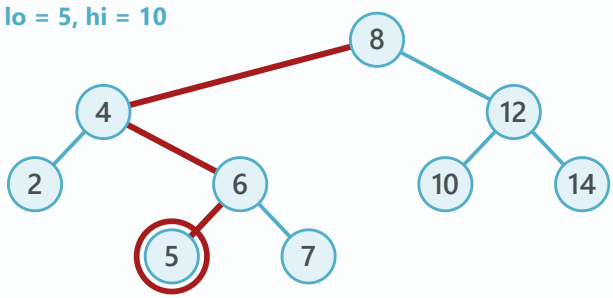
```
1 void aralik(AgacDugumu k, int lo, int hi) {
▶     if (k == null) return;
▶     if (lo < k.veri)           // solda aday var
▶         aralik(k.sol, lo, hi);
5     if (lo <= k.veri && k.veri <= hi)
6         System.out.print(k.veri + " ");
7     if (k.veri < hi)           // sağda aday var
8         aralik(k.sag, lo, hi);
9 }
```

ÇAĞRI YIĞINI

- **aralik(6)**
- aralik(4)
- aralik(8)
- main()

ÇIKTI

Kod İzleme: aralık sorgusu `aralik(kok, 5, 10)`



`aralik(5)` : 5 < 5 yanlış → sol alt ağaçtaki her anahtar < 5 ≤ lo: **sol alt ağaç budanır** (hiç ziyaret edilmez).

Adım 5 / 19

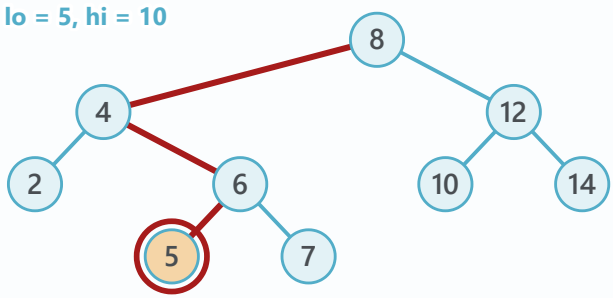
```
1 void aralik(AgacDugumu k, int lo, int hi) {
▶     if (k == null) return;
▶     if (lo < k.veri)           // solda aday var
4         aralik(k.sol, lo, hi);
5     if (lo <= k.veri && k.veri <= hi)
6         System.out.print(k.veri + " ");
7     if (k.veri < hi)           // sağda aday var
8         aralik(k.sag, lo, hi);
9 }
```

ÇAĞRI YIĞINI

- **aralik(5)**
- aralik(6)
- aralik(4)
- aralik(8)
- main()

ÇIKTI

Kod İzleme: aralık sorgusu `aralik(kok, 5, 10)`



Sol alt ağaç budandı (5 ≤ lo = 5), çağrı yapılmadı; `aralik(5)` sürer. 5 ∈ [5, 10] → **yazdır**. 5 < 10 → sağa da bak.

Adım 6 / 19

```
1 void aralik(AgacDugumu k, int lo, int hi) {
2     if (k == null) return;
3     if (lo < k.veri)           // solda aday var
4         aralik(k.sol, lo, hi);
▶     if (lo <= k.veri && k.veri <= hi)
▶         System.out.print(k.veri + " ");
▶     if (k.veri < hi)           // sağda aday var
▶         aralik(k.sag, lo, hi);
9 }
```

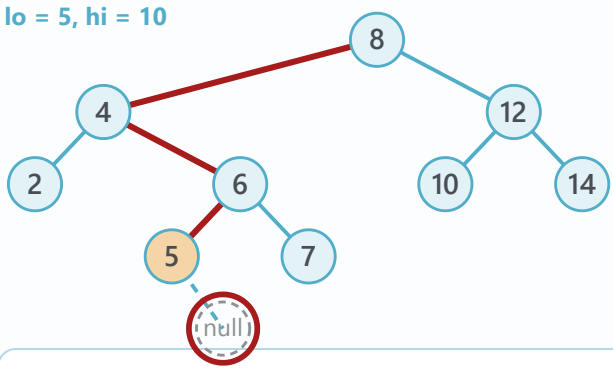
ÇAĞRI YIĞINI

- **aralik(5)**
- aralik(6)
- aralik(4)
- aralik(8)
- main()

ÇIKTI

```
5
```

Kod İzleme: aralık sorgusu `aralik(kok, 5, 10)`



`aralik(null)` : boş alt ağaç → hemen **return**.

Adım 7 / 19

```
1 void aralik(AgacDugumu k, int lo, int hi) {
▶     if (k == null) return;
3     if (lo < k.veri)           // solda aday var
4         aralik(k.sol, lo, hi);
5     if (lo <= k.veri && k.veri <= hi)
6         System.out.print(k.veri + " ");
7     if (k.veri < hi)           // sağda aday var
8         aralik(k.sag, lo, hi);
9 }
```

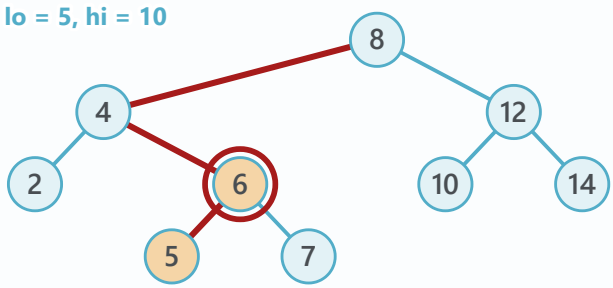
ÇAĞRI YIĞINI

- `aralik(null)`
- `aralik(5)`
- `aralik(6)`
- `aralik(4)`
- `aralik(8)`
- `main()`

ÇIKTI

```
5
```

Kod İzleme: aralık sorgusu `aralik(kok, 5, 10)`



Sol taraf bitti, `aralik(6)`'ya dönüldü. 6 ∈ [5, 10] → **yazdır**. 6 < 10 → sağa da bak.

Adım 8 / 19

```
1 void aralik(AgacDugumu k, int lo, int hi) {
2     if (k == null) return;
3     if (lo < k.veri)           // solda aday var
4         aralik(k.sol, lo, hi);
▶     if (lo <= k.veri && k.veri <= hi)
▶         System.out.print(k.veri + " ");
▶     if (k.veri < hi)           // sağda aday var
▶         aralik(k.sag, lo, hi);
9 }
```

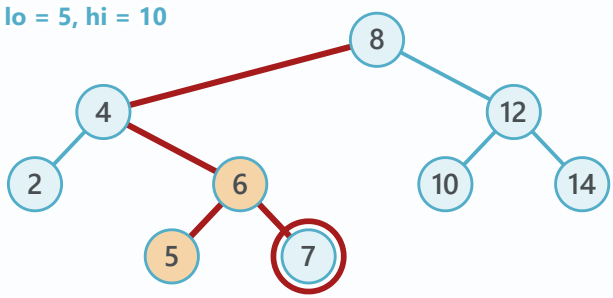
ÇAĞRI YIĞINI

- **`aralik(6)`**
- `aralik(4)`
- `aralik(8)`
- `main()`

ÇIKTI

```
5 6
```

Kod İzleme: aralık sorgusu `aralik(kok, 5, 10)`



`aralik(7)` : 5 < 7 → solda aralığa giren anahtar olabilir: `aralik(k.sol)` .

Adım 9 / 19

```
1  void aralik(AgacDugumu k, int lo, int hi) {
▶      if (k == null) return;
▶      if (lo < k.veri)           // solda aday var
▶          aralik(k.sol, lo, hi);
5      if (lo <= k.veri && k.veri <= hi)
6          System.out.print(k.veri + " ");
7      if (k.veri < hi)           // sağda aday var
8          aralik(k.sag, lo, hi);
9  }
```

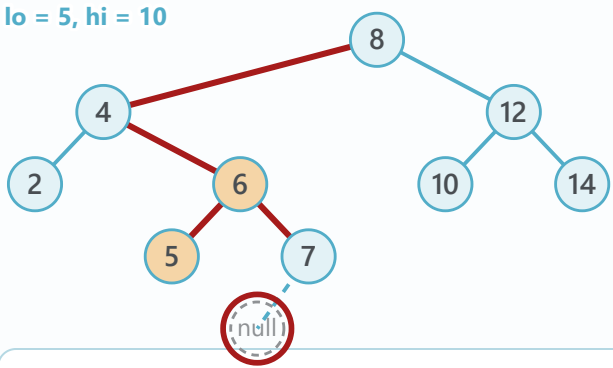
ÇAĞRI YIĞINI

- **aralik(7)**
- aralik(6)
- aralik(4)
- aralik(8)
- main()

ÇIKTI

```
5 6
```

Kod İzleme: aralık sorgusu `aralik(kok, 5, 10)`



`aralik(null)` : boş alt ağaç → hemen **return**.

Adım 10 / 19

```
1 void aralik(AgacDugumu k, int lo, int hi) {
▶       if (k == null) return;
3       if (lo < k.veri)           // solda aday var
4           aralik(k.sol, lo, hi);
5       if (lo <= k.veri && k.veri <= hi)
6           System.out.print(k.veri + " ");
7       if (k.veri < hi)           // sağda aday var
8           aralik(k.sag, lo, hi);
9   }
```

ÇAĞRI YIĞINI

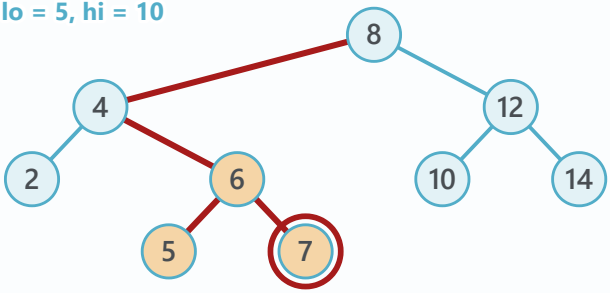
- `aralik(null)`
- `aralik(7)`
- `aralik(6)`
- `aralik(4)`
- `aralik(8)`
- `main()`

ÇIKTI

```
5 6
```

Kod İzleme: aralık sorgusu `aralik(kok, 5, 10)`

lo = 5, hi = 10



Sol taraf bitti, `aralik(7)` 'ye dönüldü. 7 ∈ [5, 10] → **yazdır**. 7 < 10 → sağa da bak.

Adım 11 / 19

```
1 void aralik(AgacDugumu k, int lo, int hi) {
2     if (k == null) return;
3     if (lo < k.veri)           // solda aday var
4         aralik(k.sol, lo, hi);
▶     if (lo <= k.veri && k.veri <= hi)
▶         System.out.print(k.veri + " ");
▶     if (k.veri < hi)           // sağda aday var
▶         aralik(k.sag, lo, hi);
9 }
```

ÇAĞRI YIĞINI

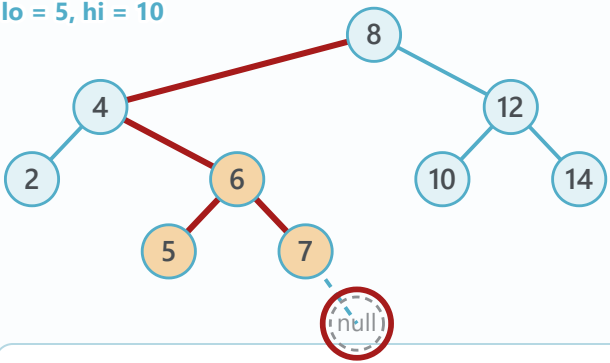
- **aralik(7)**
- aralik(6)
- aralik(4)
- aralik(8)
- main()

ÇIKTI

```
5 6 7
```

Kod İzleme: aralık sorgusu `aralik(kok, 5, 10)`

lo = 5, hi = 10



`aralik(null)` : boş alt ağaç → hemen **return**.

Adım 12 / 19

```
1 void aralik(AgacDugumu k, int lo, int hi) {
▶       if (k == null) return;
3       if (lo < k.veri)           // solda aday var
4           aralik(k.sol, lo, hi);
5       if (lo <= k.veri && k.veri <= hi)
6           System.out.print(k.veri + " ");
7       if (k.veri < hi)           // sağda aday var
8           aralik(k.sag, lo, hi);
9   }
```

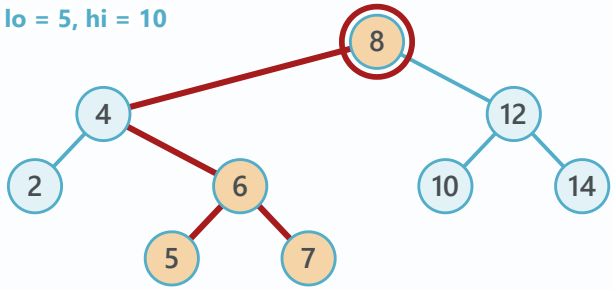
ÇAĞRI YIĞINI

- `aralik(null)`
- `aralik(7)`
- `aralik(6)`
- `aralik(4)`
- `aralik(8)`
- `main()`

ÇIKTI

```
5 6 7
```

Kod İzleme: aralık sorgusu `aralik(kok, 5, 10)`



Sol taraf bitti, `aralik(8)`'e dönüldü. 8 ∈ [5, 10] → **yazdır**. 8 < 10 → sağa da bak.

Adım 13 / 19

```
1 void aralik(AgacDugumu k, int lo, int hi) {
2     if (k == null) return;
3     if (lo < k.veri)           // solda aday var
4         aralik(k.sol, lo, hi);
▶     if (lo <= k.veri && k.veri <= hi)
▶         System.out.print(k.veri + " ");
▶     if (k.veri < hi)           // sağda aday var
▶         aralik(k.sag, lo, hi);
9 }
```

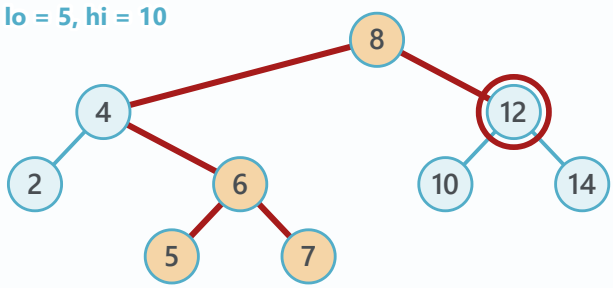
ÇAĞRI YIĞINI

- `aralik(8)`
- `main()`

ÇIKTI

```
5 6 7 8
```

Kod İzleme: aralık sorgusu `aralik(kok, 5, 10)`



`aralik(12)` : 5 < 12 → solda aralığa giren anahtar olabilir: `aralik(k.sol)` .

Adım 14 / 19

```
1 void aralik(AgacDugumu k, int lo, int hi) {
▶     if (k == null) return;
▶     if (lo < k.veri)           // solda aday var
▶         aralik(k.sol, lo, hi);
5     if (lo <= k.veri && k.veri <= hi)
6         System.out.print(k.veri + " ");
7     if (k.veri < hi)           // sağda aday var
8         aralik(k.sag, lo, hi);
9 }
```

ÇAĞRI YIĞINI

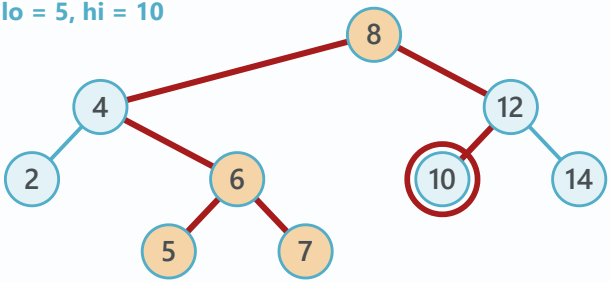
- **aralik(12)**
- aralik(8)
- main()

ÇIKTI

```
5 6 7 8
```

Kod İzleme: aralık sorgusu `aralik(kok, 5, 10)`

lo = 5, hi = 10



`aralik(10)` : 5 < 10 → solda aralığa giren anahtar olabilir: `aralik(k.sol)` .

Adım 15 / 19

```
1 void aralik(AgacDugumu k, int lo, int hi) {
▶     if (k == null) return;
▶     if (lo < k.veri)           // solda aday var
▶         aralik(k.sol, lo, hi);
5     if (lo <= k.veri && k.veri <= hi)
6         System.out.print(k.veri + " ");
7     if (k.veri < hi)           // sağda aday var
8         aralik(k.sag, lo, hi);
9 }
```

ÇAĞRI YIĞINI

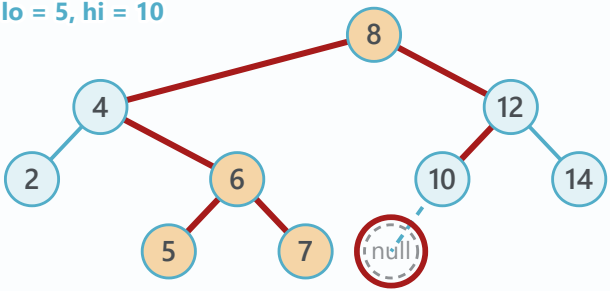
- **aralik(10)**
- aralik(12)
- aralik(8)
- main()

ÇIKTI

```
5 6 7 8
```

Kod İzleme: aralık sorgusu `aralik(kok, 5, 10)`

lo = 5, hi = 10



`aralik(null)` : boş alt ağaç → hemen **return**.

Adım 16 / 19

```
1 void aralik(AgacDugumu k, int lo, int hi) {
▶       if (k == null) return;
3       if (lo < k.veri)           // solda aday var
4           aralik(k.sol, lo, hi);
5       if (lo <= k.veri && k.veri <= hi)
6           System.out.print(k.veri + " ");
7       if (k.veri < hi)           // sağda aday var
8           aralik(k.sag, lo, hi);
9   }
```

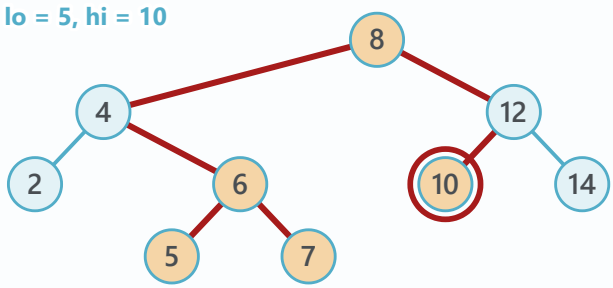
ÇAĞRI YIĞINI

- **aralik(null)**
- aralik(10)
- aralik(12)
- aralik(8)
- main()

ÇIKTI

```
5 6 7 8
```

Kod İzleme: aralık sorgusu `aralik(kok, 5, 10)`



Sol taraf bitti, `aralik(10)`'a dönüldü. 10 ∈ [5, 10] → **yazdır**. 10 ≥ 10 → sağ alt ağaç budanır.

Adım 17 / 19

```
1 void aralik(AgacDugumu k, int lo, int hi) {
2     if (k == null) return;
3     if (lo < k.veri)           // solda aday var
4         aralik(k.sol, lo, hi);
▶     if (lo <= k.veri && k.veri <= hi)
▶         System.out.print(k.veri + " ");
▶     if (k.veri < hi)           // sağda aday var
8         aralik(k.sag, lo, hi);
9 }
```

ÇAĞRI YIĞINI

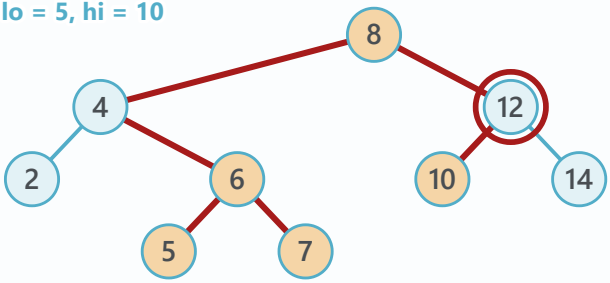
- **`aralik(10)`**
- `aralik(12)`
- `aralik(8)`
- `main()`

ÇIKTI

```
5 6 7 8 10
```

Kod İzleme: aralık sorgusu `aralik(kok, 5, 10)`

lo = 5, hi = 10



Sol taraf bitti, `aralik(12)`'ye dönüldü. 12 aralık dışında, yazdırılmaz. 12 ≥ 10 → sağ alt ağaç budanır.

Adım 18 / 19

```
1 void aralik(AgacDugumu k, int lo, int hi) {
2     if (k == null) return;
3     if (lo < k.veri)           // solda aday var
4         aralik(k.sol, lo, hi);
▶     if (lo <= k.veri && k.veri <= hi)
6         System.out.print(k.veri + " ");
▶     if (k.veri < hi)           // sağda aday var
8         aralik(k.sag, lo, hi);
9 }
```

ÇAĞRI YIĞINI

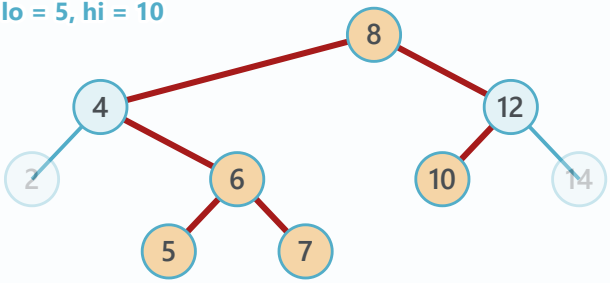
- **aralik(12)**
- aralik(8)
- main()

ÇIKTI

```
5 6 7 8 10
```

Kod İzleme: aralık sorgusu `aralik(kok, 5, 10)`

lo = 5, hi = 10



Bitti. Çıktı sıralı (kök ortada sırası): **5 6 7 8 10**. 2 ve 14 **hiç ziyaret edilmedi** (soluk). Maliyet **O(h + s)**, s: yazdırılan anahtar sayısı.

Adım 19 / 19

```
▶ void aralik(AgacDugumu k, int lo, int hi) {
2     if (k == null) return;
3     if (lo < k.veri)           // solda aday var
4         aralik(k.sol, lo, hi);
5     if (lo <= k.veri && k.veri <= hi)
6         System.out.print(k.veri + " ");
7     if (k.veri < hi)           // sağda aday var
8         aralik(k.sag, lo, hi);
9 }
```

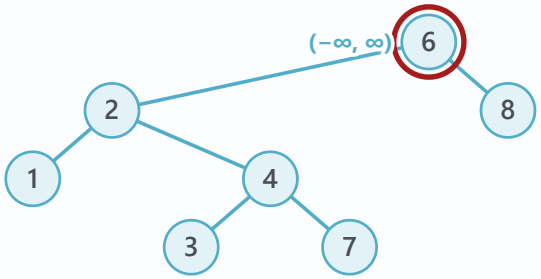
ÇAĞRI YIĞINI

`main()`

ÇIKTI

```
5 6 7 8 10
```

Kod İzleme: bir ağaç BST mi? `bstMi()`



`bstMi(6)` : 6 ∈ (−∞, ∞) ✓. Sol alt ağaç için üst sınır **6** olur: `bstMi(k.sol, −∞, 6)` .

Adım 1 / 12

```
1  boolean bstMi(AgacDugumu k, long min, long max) {
2      if (k == null) return true;
▶      if (k.veri <= min || k.veri >= max)
4          return false;          // aralık dışında
▶      return bstMi(k.sol, min, k.veri)
6          && bstMi(k.sag, k.veri, max);
7  }
8  // ilk çağrı: bstMi(kok, Long.MIN_VALUE,
9  //                  Long.MAX_VALUE)
```

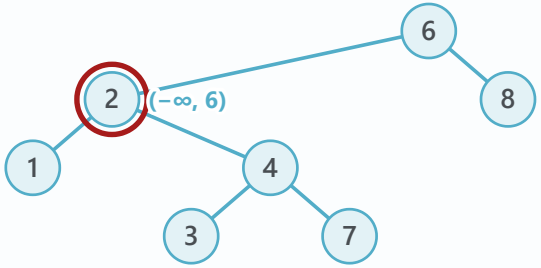
ÇAĞRI YIĞINI

- **`bstMi(6)`**
- `main()`

DEĞİŞKENLER

`k = düğüm(6)` `min = −∞` `max = ∞`

Kod İzleme: bir ağaç BST mi? `bstMi()`



`bstMi(2)` : 2 ∈ (−∞, 6) ✓. Sol alt ağaç için üst sınır **2** olur: `bstMi(k.sol, −∞, 2)` .

Adım 2 / 12

```
1 boolean bstMi(AgacDugumu k, long min, long max) {
2     if (k == null) return true;
▶     if (k.veri <= min || k.veri >= max)
4         return false;          // aralık dışında
▶     return bstMi(k.sol, min, k.veri)
6         && bstMi(k.sag, k.veri, max);
7 }
8 // ilk çağrı: bstMi(kok, Long.MIN_VALUE,
9 //                  Long.MAX_VALUE)
```

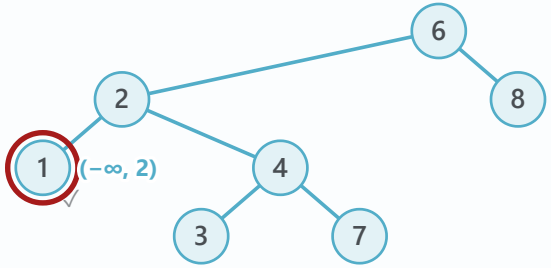
ÇAĞRI YIĞINI

- **bstMi(2)**
- bstMi(6)
- main()

DEĞİŞKENLER

`k = düğüm(2)` `min = −∞` `max = 6`

Kod İzleme: bir ağaç BST mi? `bstMi()`



`bstMi(1)` : 1 ∈ (−∞, 2) ✓. Yaprak: iki çocuğu da null → ikisi de **true** döner → **return true**.

Adım 3 / 12

```
1  boolean bstMi(AgacDugumu k, long min, long max) {
▶      if (k == null) return true;
▶      if (k.veri <= min || k.veri >= max)
4          return false;          // aralık dışında
▶      return bstMi(k.sol, min, k.veri)
▶          && bstMi(k.sag, k.veri, max);
7  }
8  // ilk çağrı: bstMi(kok, Long.MIN_VALUE,
9  //                   Long.MAX_VALUE)
```

ÇAĞRI YIĞINI

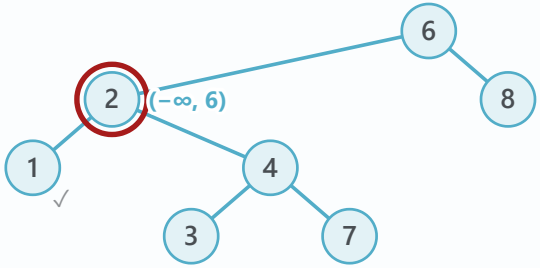
- **bstMi(1)**
- bstMi(2)
- bstMi(6)
- main()

DEĞİŞKENLER

`k = düğüm(1)` `min = −∞` `max = 2`

`dönüş = true`

Kod İzleme: bir ağaç BST mi? `bstMi()`



Sol taraf **true**. Sağ alt ağaç için alt sınır **2** olur: `bstMi(k.sag, 2, 6)`.


```
1 boolean bstMi(AgacDugumu k, long min, long max) {
2     if (k == null) return true;
3     if (k.veri <= min || k.veri >= max)
4         return false;          // aralık dışında
5     return bstMi(k.sol, min, k.veri)
▶         && bstMi(k.sag, k.veri, max);
7 }
8 // ilk çağrı: bstMi(kok, Long.MIN_VALUE,
9 //                  Long.MAX_VALUE)
```

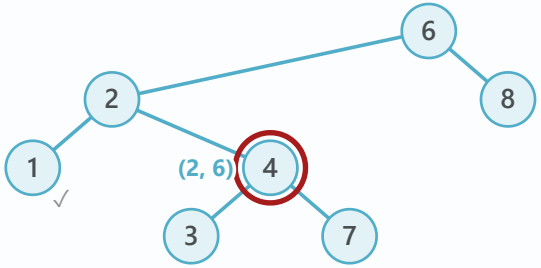
ÇAĞRI YIĞINI

- **bstMi(2)**
- bstMi(6)
- main()

DEĞİŞKENLER

`k = düğüm(2)` `min = −∞` `max = 6`

Kod İzleme: bir ağaç BST mi? `bstMi()`



`bstMi(4)` : 4 ∈ (2, 6) ✓. Sol alt ağaç için üst sınır **4** olur: `bstMi(k.sol, 2, 4)` .

Adım 5 / 12

```
1 boolean bstMi(AgacDugumu k, long min, long max) {
2     if (k == null) return true;
▶     if (k.veri <= min || k.veri >= max)
4         return false;          // aralık dışında
▶     return bstMi(k.sol, min, k.veri)
6         && bstMi(k.sag, k.veri, max);
7 }
8 // ilk çağrı: bstMi(kok, Long.MIN_VALUE,
9 //                  Long.MAX_VALUE)
```

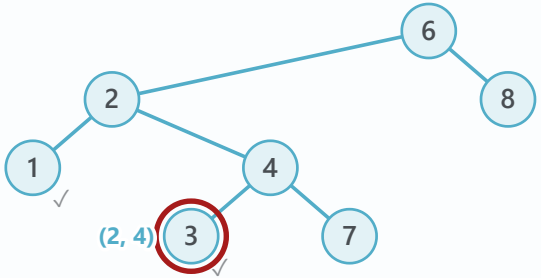
ÇAĞRI YIĞINI

- **`bstMi(4)`**
- `bstMi(2)`
- `bstMi(6)`
- `main()`

DEĞİŞKENLER

`k = düğüm(4)` `min = 2` `max = 6`

Kod İzleme: bir ağaç BST mi? `bstMi()`



`bstMi(3)` : 3 ∈ (2, 4) ✓. Yaprak: iki çocuğu da null → ikisi de **true** döner → **return true**.

Adım 6 / 12

```
1  boolean bstMi(AgacDugumu k, long min, long max) {
▶      if (k == null) return true;
▶      if (k.veri <= min || k.veri >= max)
4          return false;          // aralık dışında
▶      return bstMi(k.sol, min, k.veri)
▶          && bstMi(k.sag, k.veri, max);
7  }
8  // ilk çağrı: bstMi(kok, Long.MIN_VALUE,
9  //                  Long.MAX_VALUE)
```

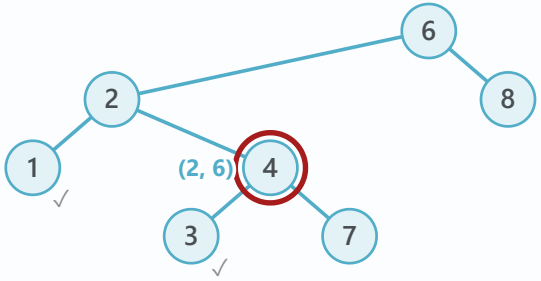
ÇAĞRI YIĞINI

- **bstMi(3)**
- bstMi(4)
- bstMi(2)
- bstMi(6)
- main()

DEĞİŞKENLER

`k = düğüm(3)` `min = 2` `max = 4` `dönüş = true`

Kod İzleme: bir ağaç BST mi? `bstMi()`



Sol taraf **true**. Sağ alt ağaç için alt sınır **4** olur: `bstMi(k.sag, 4, 6)`.

Adım 7 / 12

```
1 boolean bstMi(AgacDugumu k, long min, long max) {
2     if (k == null) return true;
3     if (k.veri <= min || k.veri >= max)
4         return false;          // aralık dışında
5     return bstMi(k.sol, min, k.veri)
▶         && bstMi(k.sag, k.veri, max);
7 }
8 // ilk çağrı: bstMi(kok, Long.MIN_VALUE,
9 //                  Long.MAX_VALUE)
```

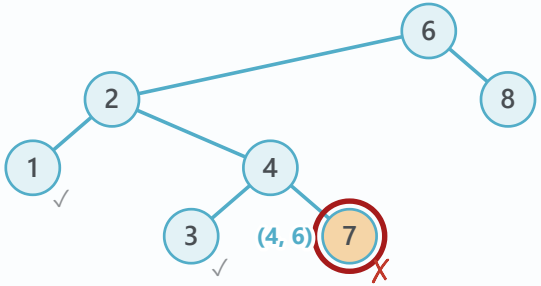
ÇAĞRI YIĞINI

- **bstMi(4)**
- bstMi(2)
- bstMi(6)
- main()

DEĞİŞKENLER

`k = düğüm(4)` `min = 2` `max = 6`

Kod İzleme: bir ağaç BST mi? `bstMi()`



`bstMi(7)`, izinli aralık **(4, 6)**: 7 ≥ 6 → **return false**. 7, 4'ün sağında olduğu için 4'ten büyük olmalı (✓), ama 6'nın sol alt ağacında olduğu için 6'dan **küçük** olmalıydı.


```
1  boolean bstMi(AgacDugumu k, long min, long max) {
2      if (k == null) return true;
▶      if (k.veri <= min || k.veri >= max)
▶          return false;          // aralık dışında
5      return bstMi(k.sol, min, k.veri)
6          && bstMi(k.sag, k.veri, max);
7  }
8  // ilk çağrı: bstMi(kok, Long.MIN_VALUE,
9  //                   Long.MAX_VALUE)
```

ÇAĞRI YIĞINI

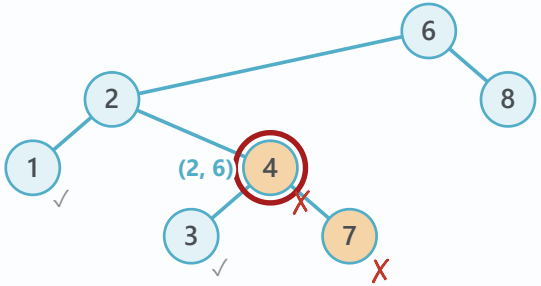
- **bstMi(7)**
- bstMi(4)
- bstMi(2)
- bstMi(6)
- main()

DEĞİŞKENLER

`k = düğüm(7)` `min = 4` `max = 6`

`dönüş = false`

Kod İzleme: bir ağaç BST mi? `bstMi()`



Sağdan **false** geldi → `bstMi(4)` = true && false = **false**.

Adım 9 / 12

```
1 boolean bstMi(AgacDugumu k, long min, long max) {
2     if (k == null) return true;
3     if (k.veri <= min || k.veri >= max)
4         return false;          // aralık dışında
▶     return bstMi(k.sol, min, k.veri)
▶         && bstMi(k.sag, k.veri, max);
7 }
8 // ilk çağrı: bstMi(kok, Long.MIN_VALUE,
9 //                  Long.MAX_VALUE)
```

ÇAĞRI YIĞINI

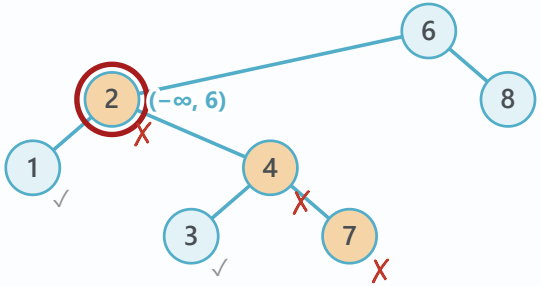
- **bstMi(4)**
- bstMi(2)
- bstMi(6)
- main()

DEĞİŞKENLER

`k = düğüm(4)` `min = 2` `max = 6`

`dönüş = false`

Kod İzleme: bir ağaç BST mi? `bstMi()`



Sağdan **false** geldi → `bstMi(2)` = true && false = **false**.

Adım 10 / 12

```
1 boolean bstMi(AgacDugumu k, long min, long max) {
2     if (k == null) return true;
3     if (k.veri <= min || k.veri >= max)
4         return false;          // aralık dışında
▶     return bstMi(k.sol, min, k.veri)
▶         && bstMi(k.sag, k.veri, max);
7 }
8 // ilk çağrı: bstMi(kok, Long.MIN_VALUE,
9 //                  Long.MAX_VALUE)
```

ÇAĞRI YIĞINI

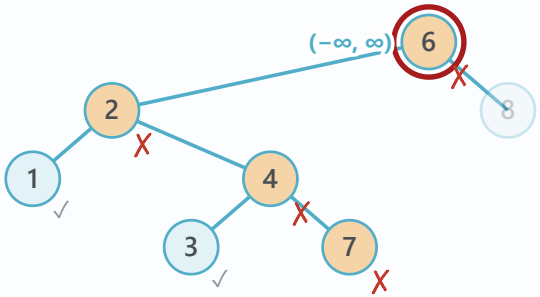
- **bstMi(2)**
- bstMi(6)
- main()

DEĞİŞKENLER

`k = düğüm(2)` `min = −∞` `max = 6`

`dönüş = false`

Kod İzleme: bir ağaç BST mi? `bstMi()`



Sol sonuç **false** → `&&` kısa devre yapar: sağ alt ağaç hiç denetlenmez, `bstMi(6)` **false** döner.

Adım 11 / 12

```
1 boolean bstMi(AgacDugumu k, long min, long max) {
2     if (k == null) return true;
3     if (k.veri <= min || k.veri >= max)
4         return false;          // aralık dışında
▶     return bstMi(k.sol, min, k.veri)
▶         && bstMi(k.sag, k.veri, max);
7 }
8 // ilk çağrı: bstMi(kok, Long.MIN_VALUE,
9 //                  Long.MAX_VALUE)
```

ÇAĞRI YIĞINI

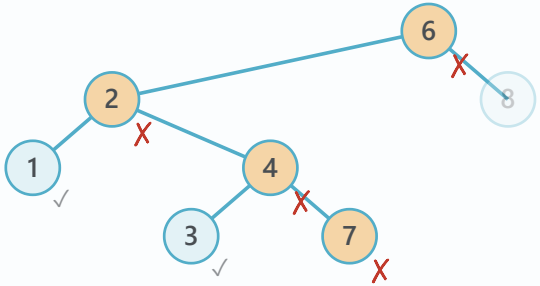
- **bstMi(6)**
- main()

DEĞİŞKENLER

`k = düğüm(6)` `min = −∞` `max = ∞`

`dönüş = false`

Kod İzleme: bir ağaç BST mi? `bstMi()`



Sonuç **false**: ağaç BST değil. Yalnızca **ebeveyn–çocuk** çiftlerine bakan kontrol (4 < 7 ✓) bu ağaca yanlışlıkla **true** derdi; aralığı aşağı taşımak şart. Maliyet O(n).

Adım 12 / 12

```
1 boolean bstMi(AgacDugumu k, long min, long max) {
2     if (k == null) return true;
3     if (k.veri <= min || k.veri >= max)
4         return false;          // aralık dışında
5     return bstMi(k.sol, min, k.veri)
6         && bstMi(k.sag, k.veri, max);
7 }
▶ // ilk çağrı: bstMi(kok, Long.MIN_VALUE,
▶ //                  Long.MAX_VALUE)
```

ÇAĞRI YIĞINI

`main()`

DEĞİŞKENLER

`sonuç = false`

Uygulama: Öğrenci Kayıtları

Anahtar olarak **öğrenci numarası** kullanılır; düğüm, öğrenci nesnesinin tamamını taşır.

```
public class Ogrenci {
    int ogrenciNo;      // anahtar
    String adSoyad;

    public Ogrenci(int ogrenciNo,
                   String adSoyad) {
        this.ogrenciNo = ogrenciNo;
        this.adSoyad = adSoyad;
    }
}
```

```
public class OgrenciDugum {
    Ogrenci ogr;            // veri
    OgrenciDugum solCocuk;
    OgrenciDugum sagCocuk;

    public OgrenciDugum(Ogrenci ogr) {
        this.ogr = ogr;
        // çocuklar başlangıçta null
    }
}
```

Aynı fikir Java'daki `TreeMap<Integer, Ogrenci>` ile hazır gelir: anahtar → değer eşlemesi, içeride dengeli bir arama ağacı (kırmızı-siyah).

BST İŞLEMLERİ

Öğrenci Ekleme ve Arama

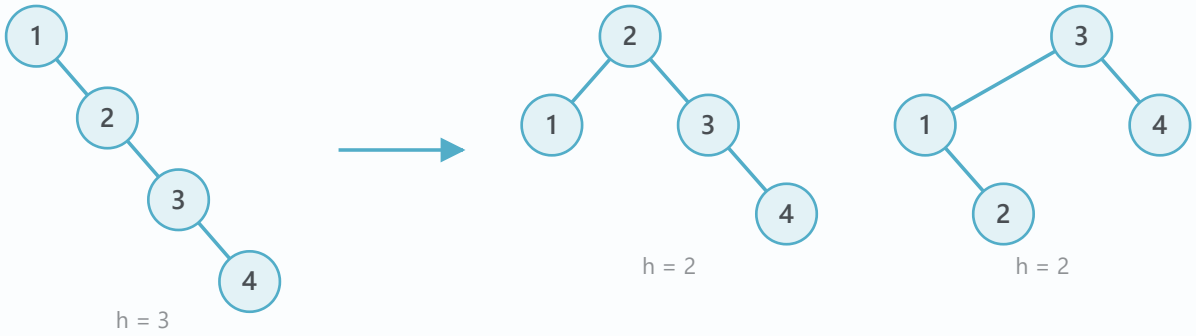
```
OgrenciDugum ekle(OgrenciDugum kok, Ogrenci ogr) {
    if (kok == null) return new OgrenciDugum(ogr);
    if (ogr.ogrenciNo < kok.ogr.ogrenciNo)
        kok.solCocuk = ekle(kok.solCocuk, ogr);   // soldan devam
    else if (ogr.ogrenciNo > kok.ogr.ogrenciNo)
        kok.sagCocuk = ekle(kok.sagCocuk, ogr);   // sağdan devam
    return kok;                                    // aynı numara eklenmez
}
```

```
Ogrenci ara(OgrenciDugum kok, int numara) {
    if (kok == null) return null;                // kayıt yok
    if (numara == kok.ogr.ogrenciNo) return kok.ogr; // bulundu
    if (numara < kok.ogr.ogrenciNo)
        return ara(kok.solCocuk, numara);         // soldan devam
    return ara(kok.sagCocuk, numara);             // sağdan devam
}
```

`ara`, boolean yerine öğrenci nesnesini döndürür: bulunursa ad-soyad gibi bilgilere de erişilir. `int` anahtar olduğu için `==` ve `<` doğrudan kullanılabilir.

Denge İhtiyacı

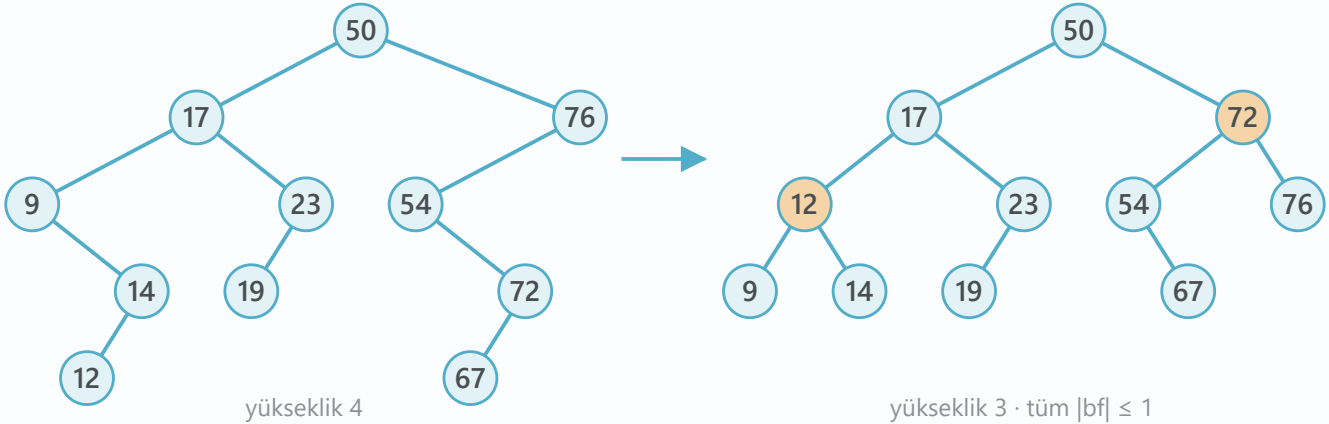
- BST'nin şekli **ekleme sırasına** bağlıdır: 1, 2, 3, 4 sırayla eklenirse ağaç bir bağlı listeye dönüşür.
- Aynı anahtarlar, farklı yapılarla daha kısa bir ağaçta tutulabilir (sağda iki seçenek).



Dengeli (self-balancing) arama ağaçları, her ekleme ve silmeden sonra yapıyı küçük yerel değişikliklerle (döndürme, renk değiştirme) düzelterek h = O(log n) garantisi verir.

Dengeleme Örneği

Soldaki ağaçta 9 → 14 → 12 ve 76 → 54 → 72 → 67 kolları uzamış. Bu alt ağaçlar yeniden düzenlenince (turuncu yeni alt ağaç kökleri) aynı 11 anahtar daha kısa bir ağaçta durur:



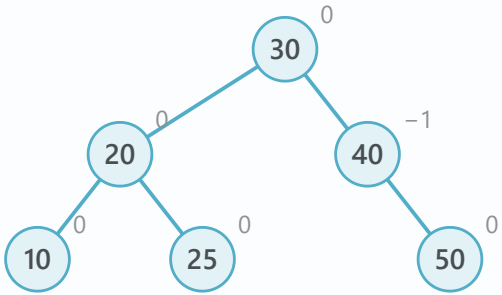
- Yeniden düzenleme sırasında **kök ortada sıra değişmez**: 9 12 14 17 19 23 50 54 67 72 76.
- Ağacı her seferinde baştan kurmak O(n) olurdu; AVL ve kırmızı-siyah ağaçlar bunu yalnızca eklenen/silinen yol üzerinde, O(log n) sürede yapar.

AVL Ağacı

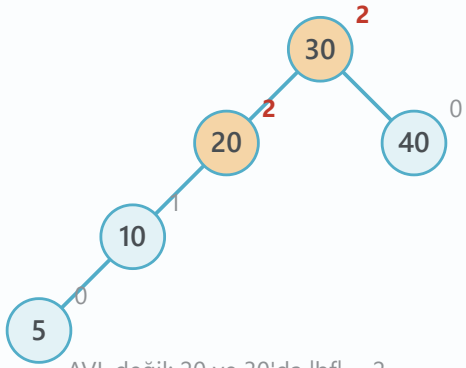
- Adelson-Velsky ve Landis (1962): ilk dengeli BST.
- **Denge faktörü**: bf(x) = h(sol) − h(sağ). Boş alt ağacın yüksekliği −1.
- AVL koşulu: **her düğümde bf ∈ {−1, 0, +1}**.
- Sonuç: $h < 1{,}44 \log_2(n + 2)$ → tüm işlemler O(log n).
- Her düğüm yüksekliğini saklar; ekleme/silmeden sonra köke doğru yukarı çıkılırken bf kontrol edilir.

Nasıl okunur?

Düğümlerin sağ üstündeki sayı bf'dir. Pozitif: sol ağır, negatif: sağ ağır. |bf| = 2 olan ilk (en alttaki) düğüm döndürülür.



AVL ✓



AVL değil: 20 ve 30'da |bf| = 2

AVL Yüksekliği Neden O(log n)?

Soru: yüksekliği h olan bir AVL ağacında **en az** kaç düğüm vardır? En seyrek AVL ağacında her iç düğümün iki alt ağacının yükseklik farkı tam 1'dir.

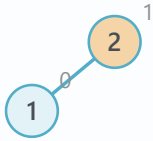


h = 0: tek düğüm. **N(0) = 1**.

N(h) = 1 + N(h−1) + N(h−2) > 2·N(h−2) → N(h) > $2^{h/2}$ → **h < 2 $\log_2$ n**. Fibonacci ile daha sıkı sınır: h < 1,44 $\log_2(n + 2)$.

AVL Yüksekliği Neden O(log n)?

Soru: yüksekliği h olan bir AVL ağacında **en az** kaç düğüm vardır? En seyrek AVL ağacında her iç düğümün iki alt ağacının yükseklik farkı tam 1'dir.



N(0) = 1

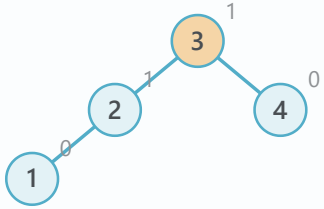
N(1) = 2

h = 1: kök + tek çocuk. **N(1) = 2**.

$N(h) = 1 + N(h-1) + N(h-2) > 2 \cdot N(h-2) \rightarrow N(h) > 2^{h/2} \rightarrow$ **$h < 2 \log_2 n$**. Fibonacci ile daha sıkı sınır: $h < 1{,}44 \log_2(n + 2)$.

AVL Yüksekliği Neden O(log n)?

Soru: yüksekliği h olan bir AVL ağacında **en az** kaç düğüm vardır? En seyrek AVL ağacında her iç düğümün iki alt ağacının yükseklik farkı tam 1'dir.



N(0) = 1

N(1) = 2

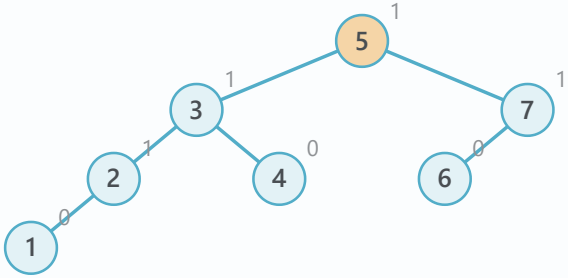
N(2) = 4

h = 2: kök (turuncu) + yüksekliği 1 olan en seyrek ağaç (sol) + yüksekliği 0 olan en seyrek ağaç (sağ). **N(2) = 1 + N(1) + N(0) = 1 + 2 + 1 = 4**.

$N(h) = 1 + N(h-1) + N(h-2) > 2 \cdot N(h-2) \rightarrow N(h) > 2^{h/2} \rightarrow$ **$h < 2 \log_2 n$**. Fibonacci ile daha sıkı sınır: $h < 1{,}44 \log_2(n + 2)$.

AVL Yüksekliği Neden O(log n)?

Soru: yüksekliği h olan bir AVL ağacında **en az** kaç düğüm vardır? En seyrek AVL ağacında her iç düğümün iki alt ağacının yükseklik farkı tam 1'dir.



N(0) = 1

N(1) = 2

N(2) = 4

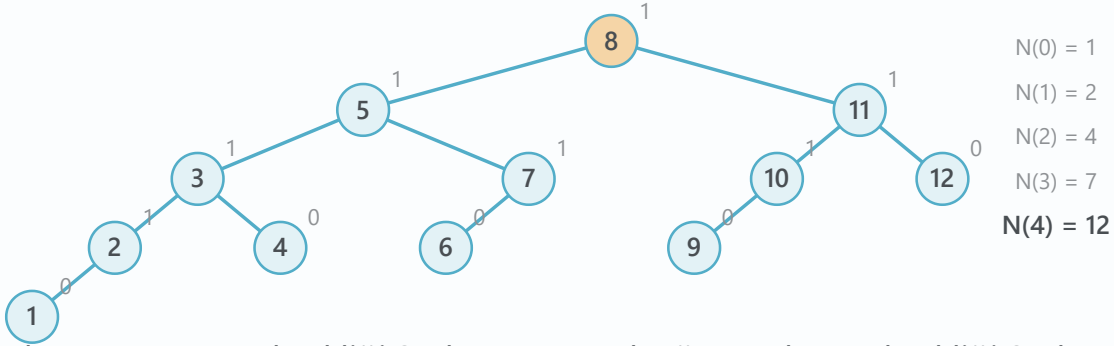
N(3) = 7

h = 3: kök (turuncu) + yüksekliği 2 olan en seyrek ağaç (sol) + yüksekliği 1 olan en seyrek ağaç (sağ). **N(3) = 1 + N(2) + N(1) = 1 + 4 + 2 = 7**.

$N(h) = 1 + N(h-1) + N(h-2) > 2 \cdot N(h-2) \rightarrow N(h) > 2^{h/2} \rightarrow$ **$h < 2 \log_2 n$**. Fibonacci ile daha sıkı sınır: $h < 1{,}44 \log_2(n + 2)$.

AVL Yüksekliği Neden O(log n)?

Soru: yüksekliği h olan bir AVL ağacında **en az** kaç düğüm vardır? En seyrek AVL ağacında her iç düğümün iki alt ağacının yükseklik farkı tam 1'dir.



h = 4: kök (turuncu) + yüksekliği 3 olan en seyrek ağaç (sol) + yüksekliği 2 olan en seyrek ağaç (sağ). **N(4) = 1 + N(3) + N(2) = 1 + 7 + 4 = 12**. Sayılar Fibonacci gibi büyür: N(h) = F(h+3) − 1.

N(h) = 1 + N(h−1) + N(h−2) > 2·N(h−2) → N(h) > $2^{h/2}$ → **h < 2 $\log_2$ n**. Fibonacci ile daha sıkı sınır: h < 1,44 $\log_2(n + 2)$.

Döndürme (Rotation)



kök ortada sıra her iki tarafta aynı: T1 · x · T2 · y · T3

Sağa döndürme (sol ağır y): x yukarı çıkar, y x'in sağ çocuğu olur; x'in sağ alt ağacı T2, y'nin soluna geçer.

Adım 1 / 2

Döndürme BST özelliğini korur (kök ortada sıra aynı kalır), ama alt ağaçların yüksekliklerini değiştirir: bir taraf bir düzey kısalır, diğeri uzar.

Döndürme (Rotation)

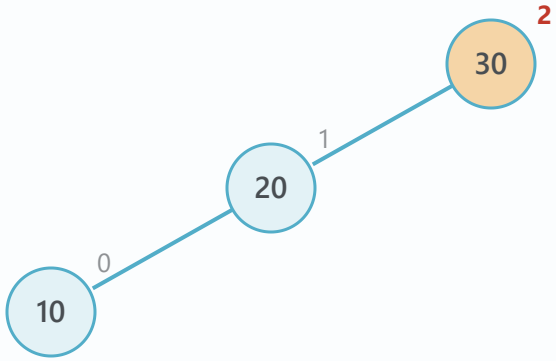


kök ortada sıra her iki tarafta aynı: T1 · x · T2 · y · T3

Sola döndürme (sağ ağır x): simetrik. Yalnızca 3 bağ değişir (kalın çizgiler) → O(1).

Döndürme BST özelliğini korur (kök ortada sıra aynı kalır), ama alt ağaçların yüksekliklerini değiştirir: bir taraf bir düzey kısalır, diğeri uzar.

Tek Döndürmeler: LL ve RR

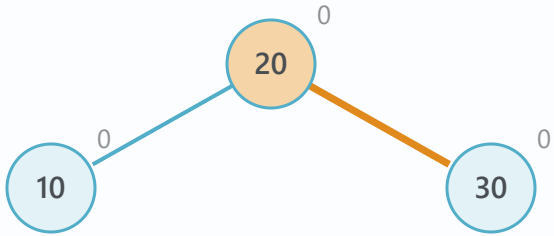


LL: 10, sol çocuğun (20) **soluna** eklendi. bf(30) = 2 *X* ve bf(20) = 1: iki düğüm de aynı yöne (sola) eğik.

Adım 1 / 4

Düğümlerin sağ üstündeki sayı bf'dir. LL durumunda sağa, RR durumunda sola **tek** döndürme yapılır; turuncu bağ, döndürmede değişen bağdır.

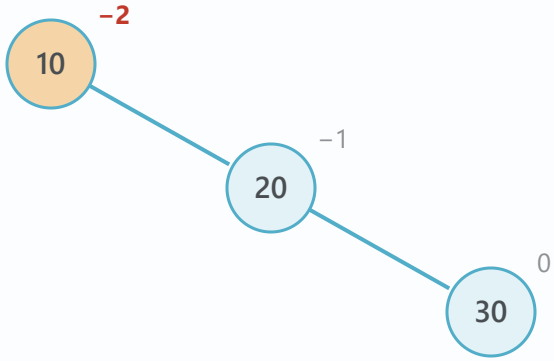
Tek Döndürmeler: LL ve RR



30'da **sağa döndür**: 20 yukarı çıkar, 30 onun sağ çocuğu olur (turuncu bağ). Tüm bf = 0.

Düğümlerin sağ üstündeki sayı bf'dir. LL durumunda sağa, RR durumunda sola **tek** döndürme yapılır; turuncu bağ, döndürmede değişen bağdır.

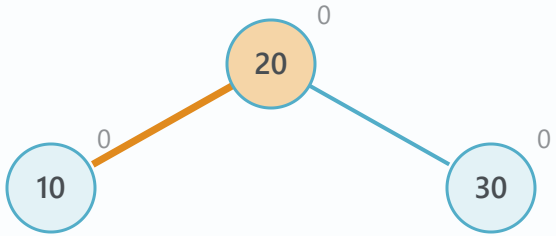
Tek Döndürmeler: LL ve RR



RR: 30, sağ çocuğun (20) **sağına** eklendi. bf(10) = −2 ✗ ve bf(20) = −1: ikisi de sağa eğik.

Düğümlerin sağ üstündeki sayı bf'dir. LL durumunda sağa, RR durumunda sola **tek** döndürme yapılır; turuncu bağ, döndürmede değişen bağdır.

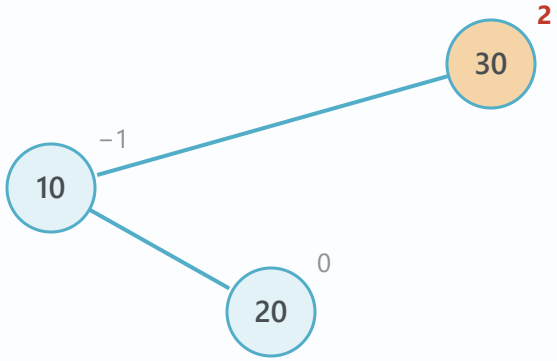
Tek Döndürmeler: LL ve RR



10'da **sola döndür**: 20 yukarı çıkar, 10 onun sol çocuğu olur. Tüm bf = 0.

Düğümlerin sağ üstündeki sayı bf'dir. LL durumunda sağa, RR durumunda sola **tek** döndürme yapılır; turuncu bağ, döndürmede değişen bağdır.

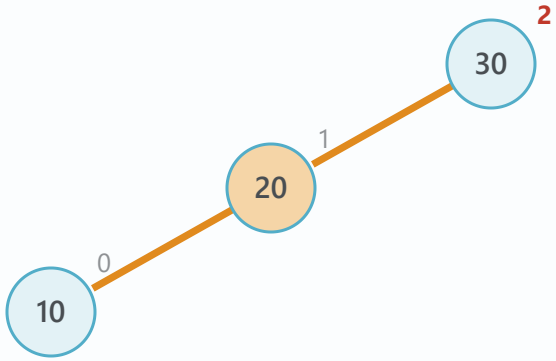
Çift Döndürmeler: LR ve RL



LR: 20, sol çocuğun (10) **sağına** eklendi. bf(30) = 2 ama bf(10) = −1: zıt yöne eğik (üçgen). Tek sağa döndürme yine üçgen bırakır.

LR: önce çocukta sola, sonra dengesiz düğümde sağa. RL: önce çocukta sağa, sonra sola.

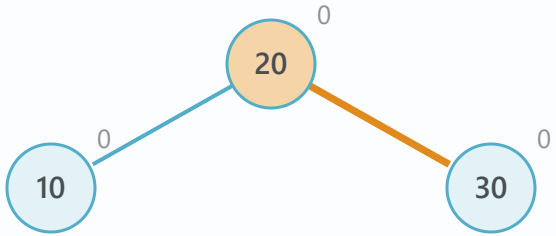
Çift Döndürmeler: LR ve RL



1. döndürme: 10'da **sola döndür** → 20 yukarı; üçgen düz çizgiye (LL) döndü. bf(30) hâlâ 2.

LR: önce çocukta sola, sonra dengesiz düğümde sağa. RL: önce çocukta sağa, sonra sola.

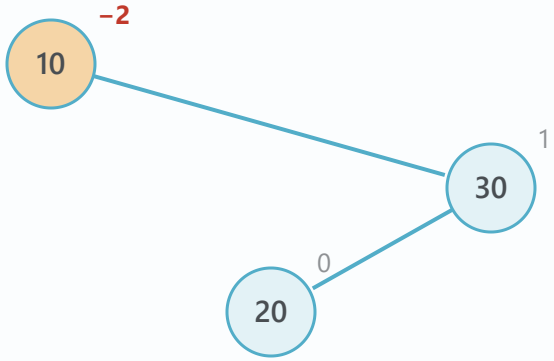
Çift Döndürmeler: LR ve RL



2. döndürme: 30'da **sağa döndür** → ortanca anahtar 20 kök oldu; tüm bf = 0.

LR: önce çocukta sola, sonra dengesiz düğümde sağa. RL: önce çocukta sağa, sonra sola.

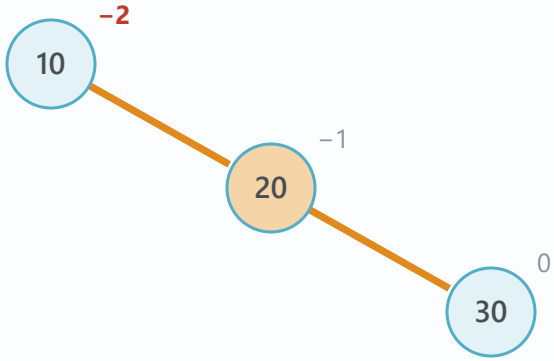
Çift Döndürmeler: LR ve RL



RL: 20, sağ çocuğun (30) **soluna** eklendi. bf(10) = −2, bf(30) = 1: üçgen.

LR: önce çocukta sola, sonra dengesiz düğümde sağa. RL: önce çocukta sağa, sonra sola.

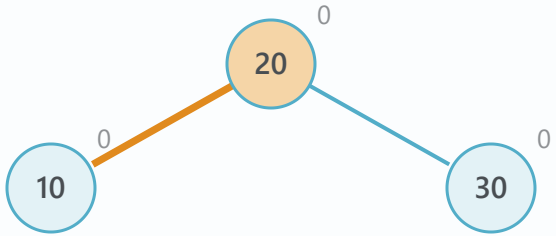
Çift Döndürmeler: LR ve RL



1. döndürme: 30'da **sağa döndür** → düz çizgi (RR). bf(10) hâlâ −2.

LR: önce çocukta sola, sonra dengesiz düğümde sağa. RL: önce çocukta sağa, sonra sola.

Çift Döndürmeler: LR ve RL



2. döndürme: 10'da **sola döndür** → 20 kök; tüm bf = 0.

LR: önce çocukta sola, sonra dengesiz düğümde sağa. RL: önce çocukta sağa, sonra sola.

AVL Döndürme ve Dengeleme Kodu

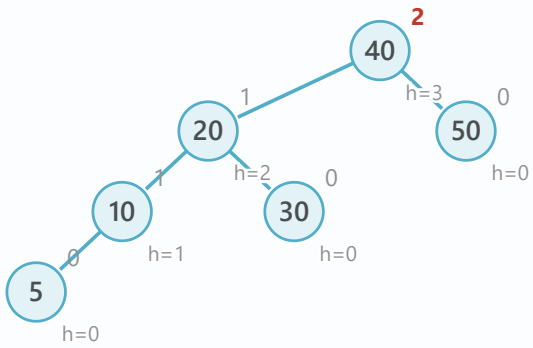
```
int h(AvlDugum d) {
    return d == null ? -1 : d.yukseklik;
}
int bf(AvlDugum d) {
    return h(d.sol) - h(d.sag);
}
void guncelle(AvlDugum d) {
    d.yukseklik = 1 +
        Math.max(h(d.sol), h(d.sag));
}
AvlDugum sagaDondur(AvlDugum y) {
    AvlDugum x = y.sol;
    y.sol = x.sag;  // T2 → y.sol
    x.sag = y;      // y → x.sag
    guncelle(y);    // önce alttaki
    guncelle(x);
    return x;       // yeni kök
}
```

```
AvlDugum dengele(AvlDugum d) {
    guncelle(d);
    if (bf(d) > 1) {       // sol ağır
        if (bf(d.sol) < 0)  // LR
            d.sol = solaDondur(d.sol);
        return sagaDondur(d); // LL
    }
    if (bf(d) < -1) {      // sağ ağır
        if (bf(d.sag) > 0)  // RL
            d.sag = sagaDondur(d.sag);
        return solaDondur(d); // RR
    }
    return d;
}
// solaDondur: sol ⇔ sag simetriği
// ekle/sil BST gibi; dönüşte:
//   return dengele(kok);
```

Özyineleme geri dönerken yol üzerindeki her düğümde `dengele` çağrılır; silmede birden fazla düğümde döndürme gerekebilir (en fazla O(log n)).

Kod İzleme: `sagaDondur(40)` — LL durumu

```
AvlDugum sagaDondur(AvlDugum y) {
    AvlDugum x = y.sol;
    y.sol = x.sag;  // T2 → y.sol
    x.sag = y;      // y → x.sag
    guncelle(y);    // önce alttaki
    guncelle(x);
    return x;       // yeni kök
}
void guncelle(AvlDugum d) {
    d.yukseklik = 1 +
        Math.max(h(d.sol), h(d.sag));
}
```



5 eklendi; dengele(40) bf = 2 buldu, sol çocuğu 20'de bf = 1 → **LL**. `sagaDondur(y)` çağrılır.

Adım 1 / 7

DEĞİŞKENLER

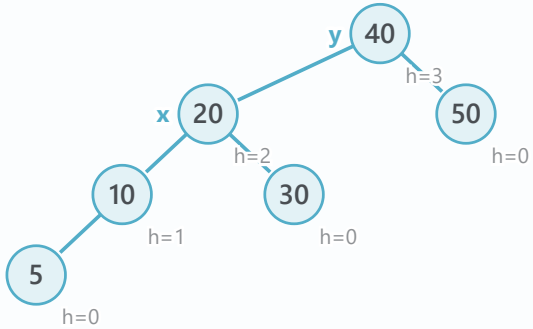
`y = düğüm(40)`

ÇAĞRI YIĞINI

- `sagaDondur(40)`
- `dengele(40)`
- `ekle(40, 5)`

Kod İzleme: `sagaDondur(40)` — LL durumu

```
1  AvlDugum sagaDondur(AvlDugum y) {
▶      AvlDugum x = y.sol;
3      y.sol = x.sag;   // T2 → y.sol
4      x.sag = y;       // y → x.sag
5      guncelle(y);     // önce alttaki
6      guncelle(x);
7      return x;        // yeni kök
8  }
9  void guncelle(AvlDugum d) {
10     d.yukseklik = 1 +
11         Math.max(h(d.sol), h(d.sag));
12 }
```



x = y.sol: yukarı çıkacak çocuk 20. Henüz hiçbir bağ değişmedi.

Adım 2 / 7

DEĞİŞKENLER

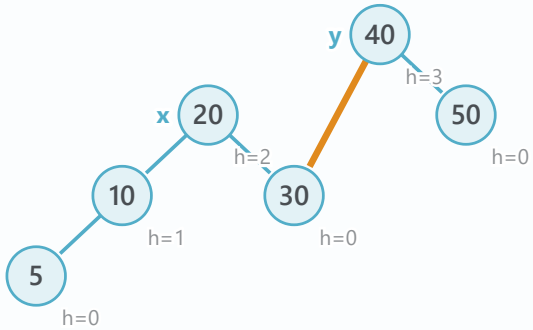
- `y = düğüm(40)`
- `x = düğüm(20)`
- `y.sol = düğüm(20)`
- `x.sag = düğüm(30)`
- `y.yukseklik = 3`
- `x.yukseklik = 2`

ÇAĞRI YIĞINI

- `sagaDondur(40)`
- `dengele(40)`
- `ekle(40, 5)`

Kod İzleme: `sagaDondur(40)` — LL durumu

```
1  AvlDugum sagaDondur(AvlDugum y) {
2      AvlDugum x = y.sol;
▶      y.sol = x.sag;   // T2 → y.sol
4      x.sag = y;       // y → x.sag
5      guncelle(y);     // önce alttaki
6      guncelle(x);
7      return x;        // yeni kök
8  }
9  void guncelle(AvlDugum d) {
10     d.yukseklik = 1 +
11         Math.max(h(d.sol), h(d.sag));
12 }
```



y.sol = x.sag: 40'ın sol bağı artık T2'yi (30) gösterir. 20 ağaçtan koptu; ona yalnızca **x** değişkeni ulaşıyor.

Adım 3 / 7

DEĞİŞKENLER

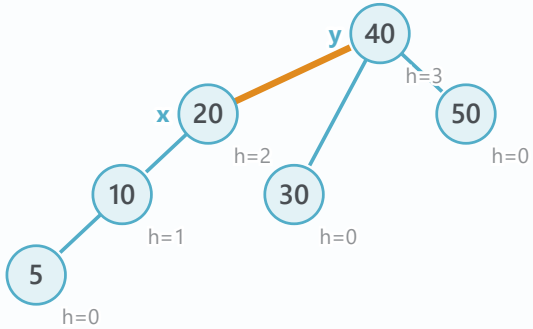
`y = düğüm(40)` `x = düğüm(20)` `y.sol = düğüm(30)` `x.sag = düğüm(30)` `y.yukseklik = 3` `x.yukseklik = 2`

ÇAĞRI YIĞINI

- `sagaDondur(40)`
- `dengele(40)`
- `ekle(40, 5)`

Kod İzleme: `sagaDondur(40)` — LL durumu

```
1  AvlDugum sagaDondur(AvlDugum y) {
2      AvlDugum x = y.sol;
3      y.sol = x.sag;   // T2 → y.sol
▶      x.sag = y;       // y → x.sag
5      guncelle(y);     // önce alttaki
6      guncelle(x);
7      return x;        // yeni kök
8  }
9  void guncelle(AvlDugum d) {
10     d.yukseklik = 1 +
11         Math.max(h(d.sol), h(d.sag));
12 }
```



x.sag = y: x.sag: 30 → 40. Artık 40, 20'nin sağ çocuğu (turuncu ok yukarı doğru).

Adım 4 / 7

DEĞİŞKENLER

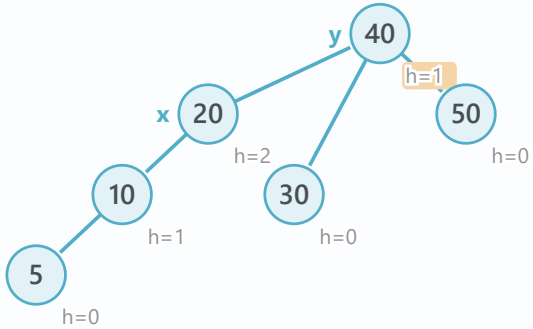
- `y = düğüm(40)`
- `x = düğüm(20)`
- `y.sol = düğüm(30)`
- `x.sag = düğüm(40)`
- `y.yukseklik = 3`
- `x.yukseklik = 2`

ÇAĞRI YIĞINI

- `sagaDondur(40)`
- `dengele(40)`
- `ekle(40, 5)`

Kod İzleme: `sagaDondur(40)` — LL durumu

```
1  AvlDugum sagaDondur(AvlDugum y) {
2      AvlDugum x = y.sol;
3      y.sol = x.sag;   // T2 → y.sol
4      x.sag = y;       // y → x.sag
▶      guncelle(y);     // önce alttaki
6      guncelle(x);
7      return x;        // yeni kök
8  }
9  void guncelle(AvlDugum d) {
▶      d.yukseklik = 1 +
▶          Math.max(h(d.sol), h(d.sag));
12 }
```



`guncelle(y)` : 40.yukseklik = 1 + max(0, 0) = **1** (önceden 3). Önce alttaki düğüm güncellenir; x'in yüksekliği buna bağlı.

Adım 5 / 7

DEĞİŞKENLER

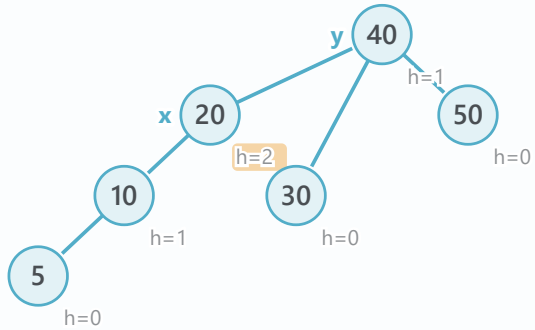
`y` = **düğüm(40)** · `x` = **düğüm(20)** · `y.sol` = **düğüm(30)** · `x.sag` = **düğüm(40)** · `y.yukseklik` = **1** · `x.yukseklik` = **2**

ÇAĞRI YIĞINI

- **`guncelle(40)`**
- `sagaDondur(40)`
- `dengele(40)`
- `ekle(40, 5)`

Kod İzleme: `sagaDondur(40)` — LL durumu

```
1  AvlDugum sagaDondur(AvlDugum y) {
2      AvlDugum x = y.sol;
3      y.sol = x.sag;   // T2 → y.sol
4      x.sag = y;       // y → x.sag
5      guncelle(y);     // önce alttaki
▶      guncelle(x);
7      return x;        // yeni kök
8  }
9  void guncelle(AvlDugum d) {
▶      d.yukseklik = 1 +
▶          Math.max(h(d.sol), h(d.sag));
12 }
```



`guncelle(x)` : 20.yukseklik = 1 + max(1, 1) = **2** (değişmedi, ama yeniden hesaplanmalı).

Adım 6 / 7

DEĞİŞKENLER

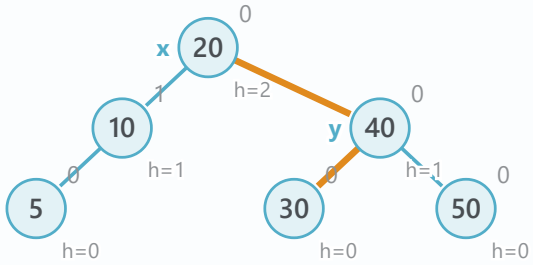
- `y` = **düğüm(40)**
- `x` = **düğüm(20)**
- `y.sol` = **düğüm(30)**
- `x.sag` = **düğüm(40)**
- `y.yukseklik` = **1**
- `x.yukseklik` = **2**

ÇAĞRI YIĞINI

- `guncelle(20)`
- `sagaDondur(40)`
- `dengele(40)`
- `ekle(40, 5)`

Kod İzleme: `sagaDondur(40)` — LL durumu

```
1  AvlDugum sagaDondur(AvlDugum y) {
2      AvlDugum x = y.sol;
3      y.sol = x.sag;   // T2 → y.sol
4      x.sag = y;       // y → x.sag
5      guncelle(y);     // önce alttaki
6      guncelle(x);
▶      return x;        // yeni kök
8  }
9  void guncelle(AvlDugum d) {
10     d.yukseklik = 1 +
11         Math.max(h(d.sol), h(d.sag));
12 }
```



return x: 20 bu alt ağacın yeni kökü (aynı bağlar, yeniden çizildi). Tüm bf ∈ {−1, 0, 1}; kök ortada sıra değişmedi. Metotta 2 bağ değişti, 3. bağı (ebeveynden) çağıran atar → **O(1)**.

Adım 7 / 7

DEĞİŞKENLER

- `y` = **düğüm(40)**
- `x` = **düğüm(20)**
- `y.sol` = **düğüm(30)**
- `x.sag` = **düğüm(40)**
- `y.yukseklik` = **1**
- `x.yukseklik` = **2**

ÇAĞRI YIĞINI

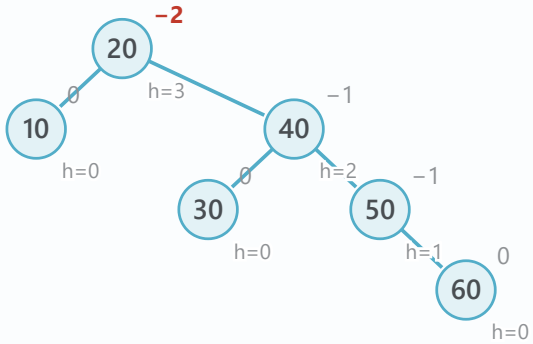
- **dengele(40)**
- ekle(40, 5)

Kod İzleme: `solaDondur(20)` — RR durumu

```
▶ AvlDugum solaDondur(AvlDugum x) {
2     AvlDugum y = x.sag;
3     x.sag = y.sol;   // T2 → x.sag
4     y.sol = x;       // x → y.sol
5     guncelle(x);     // önce alttaki
6     guncelle(y);
7     return y;        // yeni kök
8 }
9 void guncelle(AvlDugum d) {
10    d.yukseklik = 1 +
11        Math.max(h(d.sol), h(d.sag));
12 }
```

DEĞİŞKENLER

`x = düğüm(20)`



60 eklendi; dengele(20) bf = −2 buldu, sağ çocuğu 40'ta bf = −1 → **RR**. `solaDondur(x)` çağrılır.

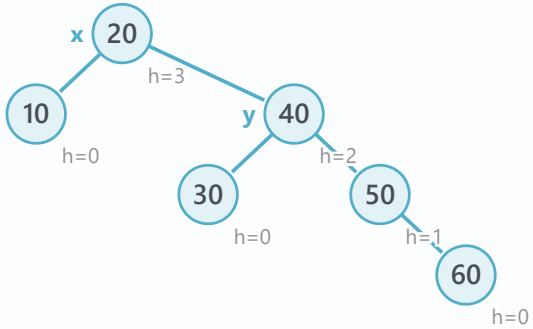
Adım 1 / 7

ÇAĞRI YIĞINI

- **`solaDondur(20)`**
- `dengele(20)`
- `ekle(20, 60)`

Kod İzleme: `solaDondur(20)` — RR durumu

```
1  AvlDugum solaDondur(AvlDugum x) {
▶      AvlDugum y = x.sag;
3      x.sag = y.sol;   // T2 → x.sag
4      y.sol = x;       // x → y.sol
5      guncelle(x);     // önce alttaki
6      guncelle(y);
7      return y;        // yeni kök
8  }
9  void guncelle(AvlDugum d) {
10     d.yukseklik = 1 +
11         Math.max(h(d.sol), h(d.sag));
12 }
```



y = x.sag: yukarı çıkacak çocuk 40. Henüz hiçbir bağ değişmedi.

Adım 2 / 7

DEĞİŞKENLER

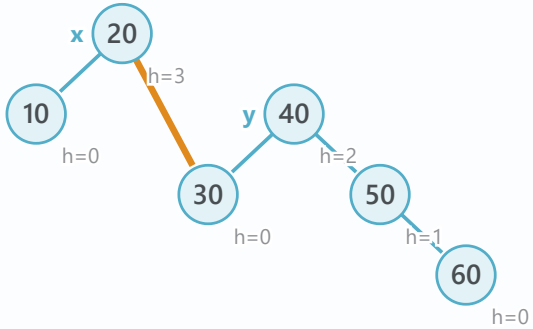
- `x = düğüm(20)`
- `y = düğüm(40)`
- `x.sag = düğüm(40)`
- `y.sol = düğüm(30)`
- `x.yukseklik = 3`
- `y.yukseklik = 2`

ÇAĞRI YIĞINI

- `solaDondur(20)`
- `dengele(20)`
- `ekle(20, 60)`

Kod İzleme: `solaDondur(20)` — RR durumu

```
1  AvlDugum solaDondur(AvlDugum x) {
2      AvlDugum y = x.sag;
▶      x.sag = y.sol;   // T2 → x.sag
4      y.sol = x;       // x → y.sol
5      guncelle(x);     // önce alttaki
6      guncelle(y);
7      return y;        // yeni kök
8  }
9  void guncelle(AvlDugum d) {
10     d.yukseklik = 1 +
11         Math.max(h(d.sol), h(d.sag));
12 }
```



x.sag = y.sol: 20'nin sağ bağı artık T2'yi (30) gösterir. 40 ağaçtan koptu; ona yalnızca **y** değişkeni ulaşıyor.

Adım 3 / 7

DEĞİŞKENLER

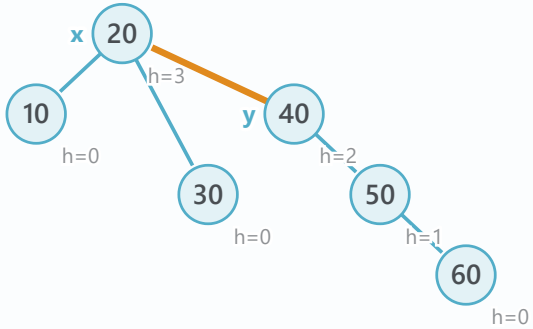
`x = düğüm(20)` `y = düğüm(40)` `x.sag = düğüm(30)` `y.sol = düğüm(30)` `x.yukseklik = 3` `y.yukseklik = 2`

ÇAĞRI YIĞINI

- `solaDondur(20)`
- `dengele(20)`
- `ekle(20, 60)`

Kod İzleme: `solaDondur(20)` — RR durumu

```
1  AvlDugum solaDondur(AvlDugum x) {
2      AvlDugum y = x.sag;
3      x.sag = y.sol;   // T2 → x.sag
▶      y.sol = x;       // x → y.sol
5      guncelle(x);     // önce alttaki
6      guncelle(y);
7      return y;        // yeni kök
8  }
9  void guncelle(AvlDugum d) {
10     d.yukseklik = 1 +
11         Math.max(h(d.sol), h(d.sag));
12 }
```



y.sol = x: y.sol: 30 → 20. Artık 20, 40'ın sol çocuğu (turuncu ok yukarı doğru).

Adım 4 / 7

DEĞİŞKENLER

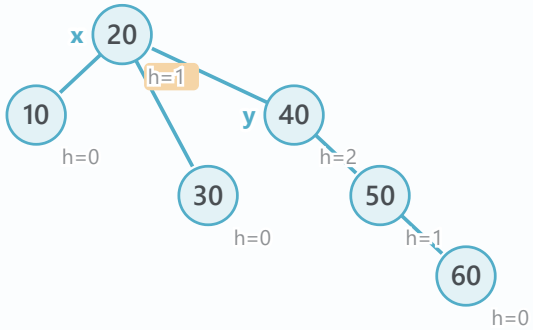
`x = düğüm(20)` `y = düğüm(40)` `x.sag = düğüm(30)` `y.sol = düğüm(20)` `x.yukseklik = 3` `y.yukseklik = 2`

ÇAĞRI YIĞINI

- `solaDondur(20)`
- `dengele(20)`
- `ekle(20, 60)`

Kod İzleme: `solaDondur(20)` — RR durumu

```
1  AvlDugum solaDondur(AvlDugum x) {
2      AvlDugum y = x.sag;
3      x.sag = y.sol;   // T2 → x.sag
4      y.sol = x;       // x → y.sol
▶      guncelle(x);     // önce alttaki
6      guncelle(y);
7      return y;        // yeni kök
8  }
9  void guncelle(AvlDugum d) {
▶      d.yukseklik = 1 +
▶          Math.max(h(d.sol), h(d.sag));
12 }
```



`guncelle(x)` : 20.yukseklik = 1 + max(0, 0) = **1** (önceden 3). Önce alttaki düğüm güncellenir; y'nin yüksekliği buna bağlı.

Adım 5 / 7

DEĞİŞKENLER

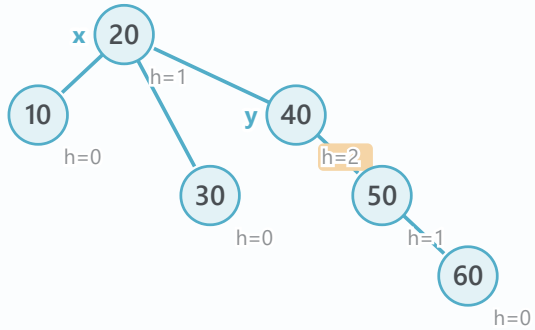
`x` = **düğüm(20)** · `y` = **düğüm(40)** · `x.sag` = **düğüm(30)** · `y.sol` = **düğüm(20)** · `x.yukseklik` = **1** · `y.yukseklik` = **2**

ÇAĞRI YIĞINI

- **`guncelle(20)`**
- `solaDondur(20)`
- `dengele(20)`
- `ekle(20, 60)`

Kod İzleme: `solaDondur(20)` — RR durumu

```
1  AvlDugum solaDondur(AvlDugum x) {
2      AvlDugum y = x.sag;
3      x.sag = y.sol;   // T2 → x.sag
4      y.sol = x;       // x → y.sol
5      guncelle(x);     // önce alttaki
▶      guncelle(y);
7      return y;        // yeni kök
8  }
9  void guncelle(AvlDugum d) {
▶      d.yukseklik = 1 +
▶          Math.max(h(d.sol), h(d.sag));
12 }
```



`guncelle(y)` : 40.yukseklik = 1 + max(1, 1) = **2** (değişmedi, ama yeniden hesaplanmalı).

Adım 6 / 7

DEĞİŞKENLER

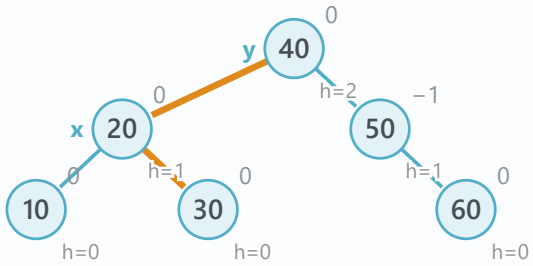
`x` = **düğüm(20)** · `y` = **düğüm(40)** · `x.sag` = **düğüm(30)** · `y.sol` = **düğüm(20)** · `x.yukseklik` = **1** · `y.yukseklik` = **2**

ÇAĞRI YIĞINI

- **`guncelle(40)`**
- `solaDondur(20)`
- `dengele(20)`
- `ekle(20, 60)`

Kod İzleme: `solaDondur(20)` — RR durumu

```
1  AvlDugum solaDondur(AvlDugum x) {
2      AvlDugum y = x.sag;
3      x.sag = y.sol;   // T2 → x.sag
4      y.sol = x;       // x → y.sol
5      guncelle(x);     // önce alttaki
6      guncelle(y);
▶      return y;        // yeni kök
8  }
9  void guncelle(AvlDugum d) {
10     d.yukseklik = 1 +
11         Math.max(h(d.sol), h(d.sag));
12 }
```



return y: 40 bu alt ağacın yeni kökü (aynı bağlar, yeniden çizildi). Tüm bf ∈ {−1, 0, 1}; kök ortada sıra değişmedi. Metotta 2 bağ değişti, 3. bağı (ebeveynden) çağıran atar → **O(1)**.

Adım 7 / 7

DEĞİŞKENLER

- `x` = **düğüm(20)**
- `y` = **düğüm(40)**
- `x.sag` = **düğüm(30)**
- `y.sol` = **düğüm(20)**
- `x.yukseklik` = **1**
- `y.yukseklik` = **2**

ÇAĞRI YIĞINI

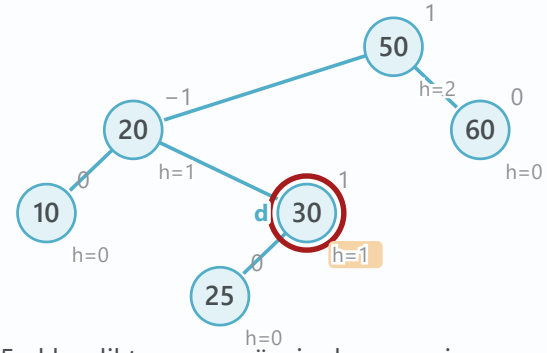
- **`dengele(20)`**
- `ekle(20, 60)`

Kod İzleme: `dengele(50)` — LR durumu

```
1  AvlDugum dengele(AvlDugum d) {
▶      guncelle(d);
3      if (bf(d) > 1) {        // sol ağır
4          if (bf(d.sol) < 0)  // LR
5              d.sol = solaDondur(d.sol);
6          return sagaDondur(d); // LL
7      }
8      if (bf(d) < -1) {       // sağ ağır
9          if (bf(d.sag) > 0)  // RL
10             d.sag = sagaDondur(d.sag);
11         return solaDondur(d); // RR
12     }
13     return d;
14 }
```

DEĞİŞKENLER

`d = düğüm(30)` `d.yukseklik = 1` `bf(d) = 1`



25 eklendikten sonra özyineleme geri dönerken `dengele(30)` çağrılır. guncelle: yükseklik 0 → **1**, bf = 1.

Adım 1 / 10

ÇAĞRI YIĞINI

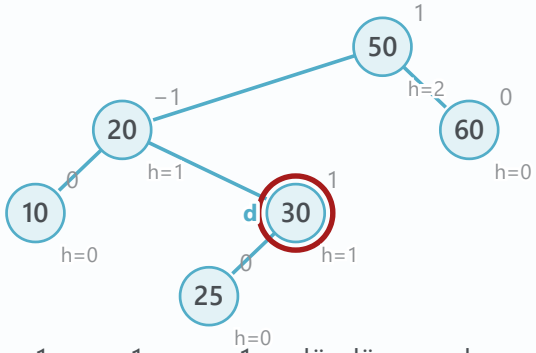
- **`dengele(30)`**
- `ekle(30, 25)`
- `ekle(20, 25)`
- `ekle(50, 25)`
- `main()`

Kod İzleme: `dengele(50)` — LR durumu

```
1  AvlDugum dengele(AvlDugum d) {
2      guncelle(d);
▶      if (bf(d) > 1) {        // sol ağır
4          if (bf(d.sol) < 0)  // LR
5              d.sol = solaDondur(d.sol);
6          return sagaDondur(d); // LL
7      }
▶      if (bf(d) < -1) {       // sağ ağır
9          if (bf(d.sag) > 0)  // RL
10             d.sag = sagaDondur(d.sag);
11         return solaDondur(d); // RR
12     }
▶      return d;
14 }
```

DEĞİŞKENLER

`d = düğüm(30)` `d.yukseklik = 1` `bf(d) = 1`



bf = 1: ne > 1 ne < −1 → döndürme yok, **return d**. Bir üst çağrıya dönülür.

Adım 2 / 10

ÇAĞRI YIĞINI

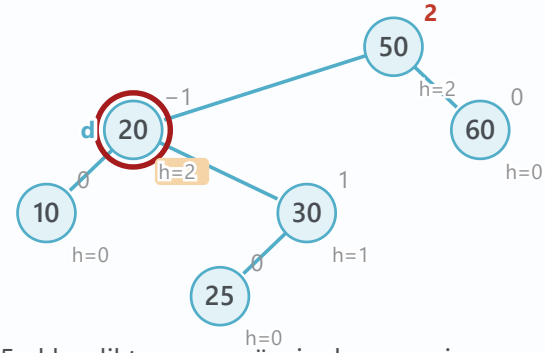
- **`dengele(30)`**
- `ekle(30, 25)`
- `ekle(20, 25)`
- `ekle(50, 25)`
- `main()`

Kod İzleme: `dengele(50)` — LR durumu

```
1  AvlDugum dengele(AvlDugum d) {
▶      guncelle(d);
3      if (bf(d) > 1) {        // sol ağır
4          if (bf(d.sol) < 0)  // LR
5              d.sol = solaDondur(d.sol);
6          return sagaDondur(d); // LL
7      }
8      if (bf(d) < -1) {       // sağ ağır
9          if (bf(d.sag) > 0)  // RL
10             d.sag = sagaDondur(d.sag);
11         return solaDondur(d); // RR
12     }
13     return d;
14 }
```

DEĞİŞKENLER

`d = düğüm(20)` `d.yukseklik = 2` `bf(d) = -1`



25 eklendikten sonra özyineleme geri dönerken `dengele(20)` çağrılır. guncelle: yükseklik 1 → **2**, bf = −1.

Adım 3 / 10

ÇAĞRI YIĞINI

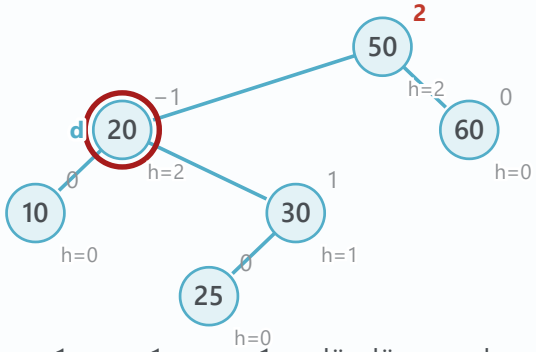
- `dengele(20)`
- `ekle(20, 25)`
- `ekle(50, 25)`
- `main()`

Kod İzleme: `dengele(50)` — LR durumu

```
1  AvlDugum dengele(AvlDugum d) {
2      guncelle(d);
▶      if (bf(d) > 1) {        // sol ağır
4          if (bf(d.sol) < 0)  // LR
5              d.sol = solaDondur(d.sol);
6          return sagaDondur(d); // LL
7      }
▶      if (bf(d) < -1) {       // sağ ağır
9          if (bf(d.sag) > 0)  // RL
10             d.sag = sagaDondur(d.sag);
11         return solaDondur(d); // RR
12     }
▶      return d;
14 }
```

DEĞİŞKENLER

`d = düğüm(20)` `d.yukseklik = 2` `bf(d) = -1`



bf = −1: ne > 1 ne < −1 → döndürme yok, **return d**. Bir üst çağrıya dönülür.

Adım 4 / 10

ÇAĞRI YIĞINI

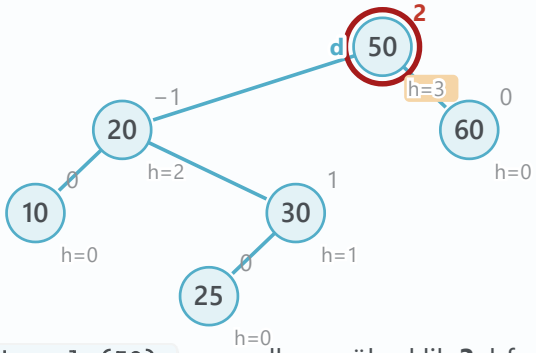
- **`dengele(20)`**
- `ekle(20, 25)`
- `ekle(50, 25)`
- `main()`

Kod İzleme: `dengele(50)` — LR durumu

```
1  AvlDugum dengele(AvlDugum d) {
▶      guncelle(d);
3      if (bf(d) > 1) {        // sol ağır
4          if (bf(d.sol) < 0)  // LR
5              d.sol = solaDondur(d.sol);
6          return sagaDondur(d); // LL
7      }
8      if (bf(d) < -1) {       // sağ ağır
9          if (bf(d.sag) > 0)  // RL
10             d.sag = sagaDondur(d.sag);
11         return solaDondur(d); // RR
12     }
13     return d;
14 }
```

DEĞİŞKENLER

`d = düğüm(50)` `d.yukseklik = 3` `bf(d) = 2`



`dengele(50)` : guncelle → yükseklik **3**, bf = h(20) − h(60) = 2 − 0 = **2**.

Adım 5 / 10

ÇAĞRI YIĞINI

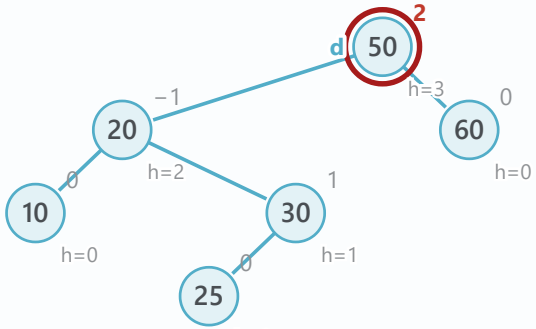
- **`dengele(50)`**
- `ekle(50, 25)`
- `main()`

Kod İzleme: `dengele(50)` — LR durumu

```
1  AvlDugum dengele(AvlDugum d) {
2      guncelle(d);
▶      if (bf(d) > 1) {        // sol ağır
4          if (bf(d.sol) < 0)  // LR
5              d.sol = solaDondur(d.sol);
6          return sagaDondur(d); // LL
7      }
8      if (bf(d) < -1) {       // sağ ağır
9          if (bf(d.sag) > 0)  // RL
10             d.sag = sagaDondur(d.sag);
11         return solaDondur(d); // RR
12     }
13     return d;
14 }
```

DEĞİŞKENLER

`d = düğüm(50)` `d.yukseklik = 3` `bf(d) = 2`



bf(d) = 2 > 1 → **sol ağır**. Hangi döndürme? Sol çocuğun eğimine bakılır.

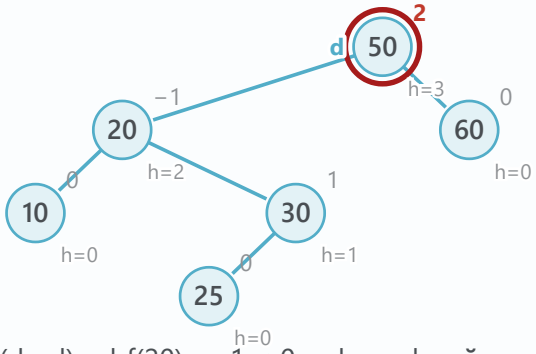
Adım 6 / 10

ÇAĞRI YIĞINI

- `dengele(50)`
- `ekle(50, 25)`
- `main()`

Kod İzleme: `dengele(50)` — LR durumu

```
1  AvlDugum dengele(AvlDugum d) {
2      guncelle(d);
3      if (bf(d) > 1) {        // sol ağır
▶          if (bf(d.sol) < 0)  // LR
5              d.sol = solaDondur(d.sol);
6          return sagaDondur(d); // LL
7      }
8      if (bf(d) < -1) {       // sağ ağır
9          if (bf(d.sag) > 0)  // RL
10             d.sag = sagaDondur(d.sag);
11         return solaDondur(d); // RR
12     }
13     return d;
14 }
```



bf(d.sol) = bf(20) = -1 < 0: sol çocuk **sağa** eğik → **LR** (üçgen). Tek sağa döndürme yetmez.

Adım 7 / 10

DEĞİŞKENLER

`d = düğüm(50)` `d.yukseklik = 3` `bf(d) = 2` `bf(d.sol) = −1`

ÇAĞRI YIĞINI

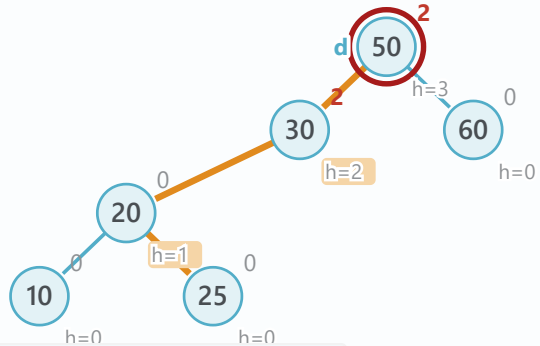
- `dengele(50)`
- `ekle(50, 25)`
- `main()`

Kod İzleme: `dengele(50)` — LR durumu

```
1  AvlDugum dengele(AvlDugum d) {
2      guncelle(d);
3      if (bf(d) > 1) {        // sol ağır
4          if (bf(d.sol) < 0)  // LR
▶              d.sol = solaDondur(d.sol);
6          return sagaDondur(d); // LL
7      }
8      if (bf(d) < -1) {       // sağ ağır
9          if (bf(d.sag) > 0)  // RL
10             d.sag = sagaDondur(d.sag);
11         return solaDondur(d); // RR
12     }
13     return d;
14 }
```

DEĞİŞKENLER

`d = düğüm(50)` `d.yukseklik = 3` `bf(d) = 2` `bf(d.sol) = 2`



`d.sol = solaDondur(20)` : 30 yukarı çıkar, 25 → 20'nin sağına geçer. Üçgen düz çizgiye (LL) döndü; bf(50) hâlâ 2.

Adım 8 / 10

ÇAĞRI YIĞINI

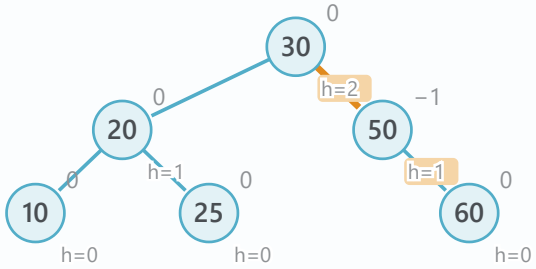
- **`solaDondur(20)`**
- `dengele(50)`
- `ekle(50, 25)`
- `main()`

Kod İzleme: `dengele(50)` — LR durumu

```
1  AvlDugum dengele(AvlDugum d) {
2      guncelle(d);
3      if (bf(d) > 1) {        // sol ağır
4          if (bf(d.sol) < 0)  // LR
5              d.sol = solaDondur(d.sol);
▶          return sagaDondur(d); // LL
7      }
8      if (bf(d) < -1) {       // sağ ağır
9          if (bf(d.sag) > 0)  // RL
10             d.sag = sagaDondur(d.sag);
11         return solaDondur(d); // RR
12     }
13     return d;
14 }
```

DEĞİŞKENLER

`d = düğüm(50)` `d.yukseklik = 1` `bf(d) = -1`



`return sagaDondur(50)` : 30 kök olur, 50 onun sağına iner; 30'un eski sağı (null) 50'nin soluna geçer.

Adım 9 / 10

ÇAĞRI YIĞINI

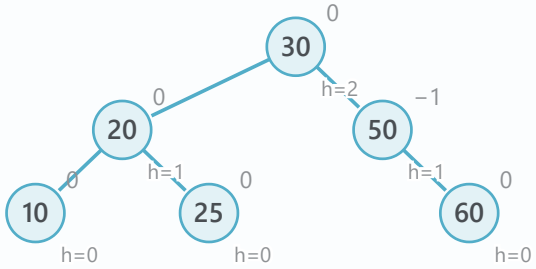
- **`sagaDondur(50)`**
- `dengele(50)`
- `ekle(50, 25)`
- `main()`

Kod İzleme: `dengele(50)` — LR durumu

```
1  AvlDugum dengele(AvlDugum d) {
2      guncelle(d);
3      if (bf(d) > 1) {        // sol ağır
4          if (bf(d.sol) < 0)  // LR
5              d.sol = solaDondur(d.sol);
▶          return sagaDondur(d); // LL
7      }
8      if (bf(d) < -1) {       // sağ ağır
9          if (bf(d.sag) > 0)  // RL
10             d.sag = sagaDondur(d.sag);
11         return solaDondur(d); // RR
12     }
13     return d;
14 }
```

DEĞİŞKENLER

`dönen = düğüm(30)` `bf(30) = 0`



`dengele(50)` yeni alt ağaç kökü **30**'u döndürür; `ekle` onu çağırana (burada `kok`) atar. Yükseklik 3'ten 2'ye indi.

Adım 10 / 10

ÇAĞRI YIĞINI

`ekle(50, 25)`

`main()`

AVL Ekleme İzleme: 10, 20, 30



Ağaç boş: `ekle(null, 10)` → d == null, yeni düğüm döner; **kök = 10**, bf = 0.

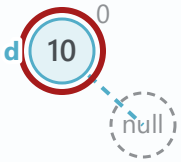
Adım 1 / 8

```
1 AvlDugum ekle(AvlDugum d, int k) {
▶       if (d == null) return new AvlDugum(k);
3       if (k < d.anahtar)
4           d.sol = ekle(d.sol, k);
5       else if (k > d.anahtar)
6           d.sag = ekle(d.sag, k);
7       else return d;          // tekrar yok
8       return dengele(d);
9   }
```

DEĞİŞKENLER

k = **10** | d = **null**

AVL Ekleme İzleme: 10, 20, 30



`ekle(kok, 20)` BST gibi iner: 20 > 10 → sağ.

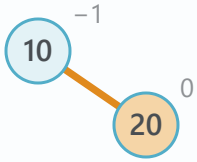
Adım 2 / 8

```
1 AvlDugum ekle(AvlDugum d, int k) {
2     if (d == null) return new AvlDugum(k);
3     if (k < d.anahtar)
4         d.sol = ekle(d.sol, k);
▶     else if (k > d.anahtar)
▶         d.sag = ekle(d.sag, k);
7     else return d;           // tekrar yok
8     return dengele(d);
9 }
```

DEĞİŞKENLER

`k = 20` `d = düğüm(10)`

AVL Ekleme İzleme: 10, 20, 30



d == null → yeni yaprak **20** döner ve 10'un sağ bağına atanır. Dönüşte yol üzerindeki bf'ler değişti.

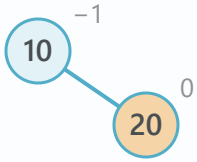
Adım 3 / 8

```
1 AvlDugum ekle(AvlDugum d, int k) {
▶       if (d == null) return new AvlDugum(k);
3       if (k < d.anahtar)
4           d.sol = ekle(d.sol, k);
5       else if (k > d.anahtar)
6           d.sag = ekle(d.sag, k);
7       else return d;          // tekrar yok
8       return dengele(d);
9   }
```

DEĞİŞKENLER

`k = 20` `d = null`

AVL Ekleme İzleme: 10, 20, 30



Dönüşte her atada `return dengele(d)` : bf(10) = −1 → hepsinde |bf| ≤ 1, döndürme yok.

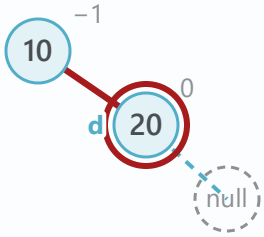
Adım 4 / 8

```
1 AvlDugum ekle(AvlDugum d, int k) {
2     if (d == null) return new AvlDugum(k);
3     if (k < d.anahtar)
4         d.sol = ekle(d.sol, k);
5     else if (k > d.anahtar)
6         d.sag = ekle(d.sag, k);
7     else return d;           // tekrar yok
▶     return dengele(d);
9 }
```

DEĞİŞKENLER

`k = 20` `d = düğüm(10)` `bf(d) = −1`

AVL Ekleme İzleme: 10, 20, 30



`ekle(kok, 30)` BST gibi iner: 30 > 10 → sağ; 30 > 20 → sağ.

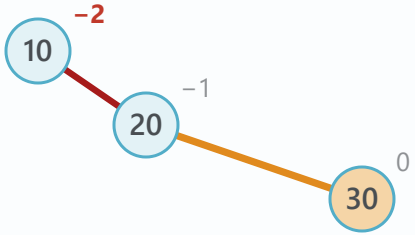
Adım 5 / 8

```
1 AvlDugum ekle(AvlDugum d, int k) {
2     if (d == null) return new AvlDugum(k);
3     if (k < d.anahtar)
4         d.sol = ekle(d.sol, k);
▶     else if (k > d.anahtar)
▶         d.sag = ekle(d.sag, k);
7     else return d;           // tekrar yok
8     return dengele(d);
9 }
```

DEĞİŞKENLER

`k = 30` `d = düğüm(20)`

AVL Ekleme İzleme: 10, 20, 30



d == null → yeni yaprak **30** döner ve 20'nin sağ bağına atanır. Dönüşte yol üzerindeki bf'ler değişti.

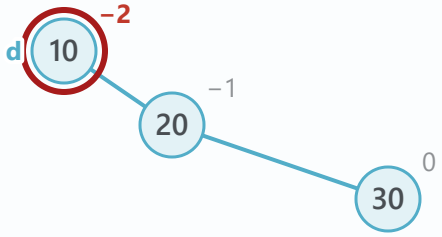
Adım 6 / 8

```
1 AvlDugum ekle(AvlDugum d, int k) {
▶     if (d == null) return new AvlDugum(k);
3     if (k < d.anahtar)
4         d.sol = ekle(d.sol, k);
5     else if (k > d.anahtar)
6         d.sag = ekle(d.sag, k);
7     else return d;          // tekrar yok
8     return dengele(d);
9 }
```

DEĞİŞKENLER

`k = 30` `d = null`

AVL Ekleme İzleme: 10, 20, 30



`dengele(10)` : bf = −2 ✗ (en alttaki dengesiz ata).
Sağ çocuk 20'de bf = −1 → **RR durumu**.

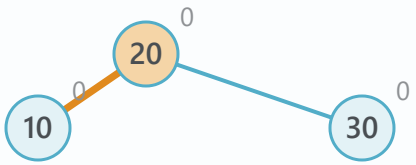
Adım 7 / 8

```
1 AvlDugum ekle(AvlDugum d, int k) {
2     if (d == null) return new AvlDugum(k);
3     if (k < d.anahtar)
4         d.sol = ekle(d.sol, k);
5     else if (k > d.anahtar)
6         d.sag = ekle(d.sag, k);
7     else return d;           // tekrar yok
▶     return dengele(d);
9 }
```

DEĞİŞKENLER

`k = 30` `d = düğüm(10)` `bf(d) = −2` `bf(çocuk) = −1`

AVL Ekleme İzleme: 10, 20, 30



10'da **solaDondur**: 20 yukarı çıkar, 10 onun sol çocuğu olur. Tüm bf ∈ {−1, 0, 1}.

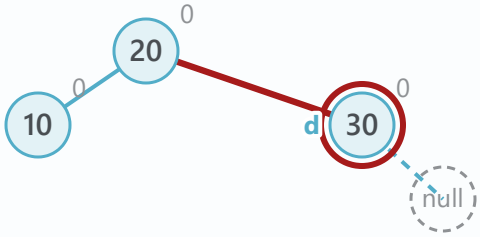
Adım 8 / 8

```
1 AvlDugum ekle(AvlDugum d, int k) {
2     if (d == null) return new AvlDugum(k);
3     if (k < d.anahtar)
4         d.sol = ekle(d.sol, k);
5     else if (k > d.anahtar)
6         d.sag = ekle(d.sag, k);
7     else return d;           // tekrar yok
▶     return dengele(d);
9 }
```

DEĞİŞKENLER

`k = 30` `d = düğüm(20)` `bf(d) = 0`

AVL Ekleme İzleme (devam): 40, 50, 25



`ekle(kok, 40)` BST gibi iner: 40 > 20 → sağ; 40 > 30 → sağ.

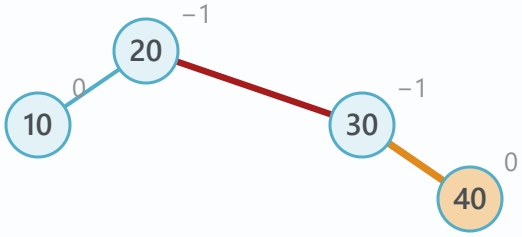
Adım 1 / 12

```
1 AvlDugum ekle(AvlDugum d, int k) {
2     if (d == null) return new AvlDugum(k);
3     if (k < d.anahtar)
4         d.sol = ekle(d.sol, k);
▶     else if (k > d.anahtar)
▶         d.sag = ekle(d.sag, k);
7     else return d;          // tekrar yok
8     return dengele(d);
9 }
```

DEĞİŞKENLER

`k = 40` `d = düğüm(30)`

AVL Ekleme İzleme (devam): 40, 50, 25



d == null → yeni yaprak **40** döner ve 30'un sağ bağına atanır. Dönüşte yol üzerindeki bf'ler değişti.

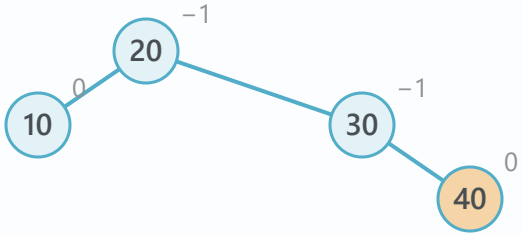
Adım 2 / 12

```
1 AvlDugum ekle(AvlDugum d, int k) {
▶       if (d == null) return new AvlDugum(k);
3       if (k < d.anahtar)
4           d.sol = ekle(d.sol, k);
5       else if (k > d.anahtar)
6           d.sag = ekle(d.sag, k);
7       else return d;          // tekrar yok
8       return dengele(d);
9   }
```

DEĞİŞKENLER

`k = 40` `d = null`

AVL Ekleme İzleme (devam): 40, 50, 25



Dönüşte her atada `return dengele(d)` : bf(30) = −1, bf(20) = −1 → hepsinde |bf| ≤ 1, döndürme yok.

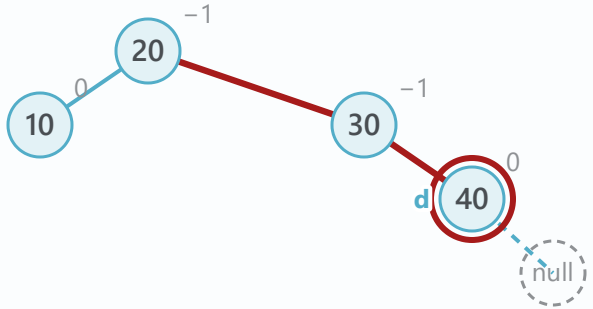
Adım 3 / 12

```
1 AvlDugum ekle(AvlDugum d, int k) {
2     if (d == null) return new AvlDugum(k);
3     if (k < d.anahtar)
4         d.sol = ekle(d.sol, k);
5     else if (k > d.anahtar)
6         d.sag = ekle(d.sag, k);
7     else return d;           // tekrar yok
▶     return dengele(d);
9 }
```

DEĞİŞKENLER

`k = 40` `d = düğüm(20)` `bf(d) = −1`

AVL Ekleme İzleme (devam): 40, 50, 25



`ekle(kok, 50)` BST gibi iner: 50 > 20 → sağ; 50 > 30 → sağ; 50 > 40 → sağ.

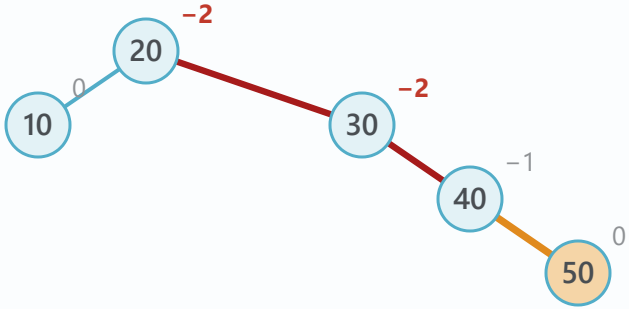
Adım 4 / 12

```
1 AvlDugum ekle(AvlDugum d, int k) {
2     if (d == null) return new AvlDugum(k);
3     if (k < d.anahtar)
4         d.sol = ekle(d.sol, k);
▶     else if (k > d.anahtar)
▶         d.sag = ekle(d.sag, k);
7     else return d;           // tekrar yok
8     return dengele(d);
9 }
```

DEĞİŞKENLER

`k = 50` `d = düğüm(40)`

AVL Ekleme İzleme (devam): 40, 50, 25



d == null → yeni yaprak **50** döner ve 40'ın sağ bağına atanır. Dönüşte yol üzerindeki bf'ler değişti.

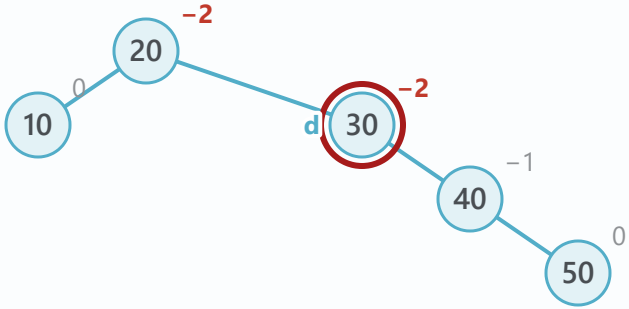
Adım 5 / 12

```
1 AvlDugum ekle(AvlDugum d, int k) {
▶       if (d == null) return new AvlDugum(k);
3       if (k < d.anahtar)
4           d.sol = ekle(d.sol, k);
5       else if (k > d.anahtar)
6           d.sag = ekle(d.sag, k);
7       else return d;          // tekrar yok
8       return dengele(d);
9   }
```

DEĞİŞKENLER

`k = 50` `d = null`

AVL Ekleme İzleme (devam): 40, 50, 25



`dengele(30)` : bf = −2 ✗ (en alttaki dengesiz ata).
Sağ çocuk 40'ta bf = −1 → **RR durumu**.

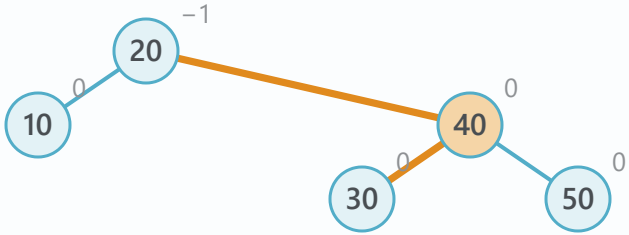
Adım 6 / 12

```
1 AvlDugum ekle(AvlDugum d, int k) {
2     if (d == null) return new AvlDugum(k);
3     if (k < d.anahtar)
4         d.sol = ekle(d.sol, k);
5     else if (k > d.anahtar)
6         d.sag = ekle(d.sag, k);
7     else return d;           // tekrar yok
▶     return dengele(d);
9 }
```

DEĞİŞKENLER

`k = 50` `d = düğüm(30)` `bf(d) = −2` `bf(çocuk) = −1`

AVL Ekleme İzleme (devam): 40, 50, 25



30'da **solaDondur**: 40 yukarı çıkar, 30 onun sol çocuğu olur. Tüm bf ∈ {−1, 0, 1}.

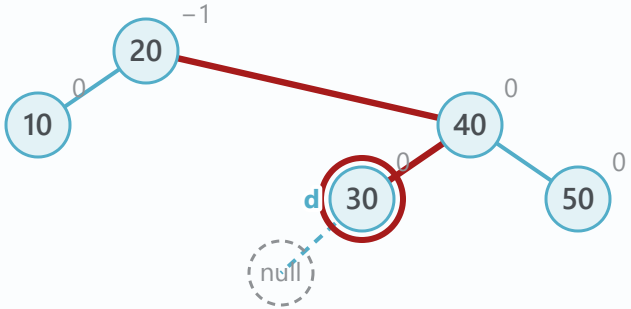
Adım 7 / 12

```
1 AvlDugum ekle(AvlDugum d, int k) {
2     if (d == null) return new AvlDugum(k);
3     if (k < d.anahtar)
4         d.sol = ekle(d.sol, k);
5     else if (k > d.anahtar)
6         d.sag = ekle(d.sag, k);
7     else return d;           // tekrar yok
▶     return dengele(d);
9 }
```

DEĞİŞKENLER

`k = 50` `d = düğüm(40)` `bf(d) = 0`

AVL Ekleme İzleme (devam): 40, 50, 25



`ekle(kok, 25)` BST gibi iner: 25 > 20 → sağ; 25 < 40 → sol; 25 < 30 → sol.

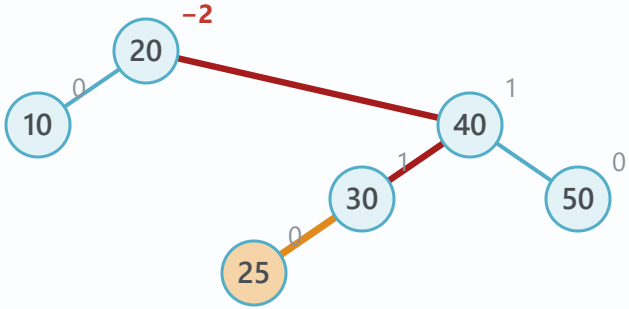
Adım 8 / 12

```
1 AvlDugum ekle(AvlDugum d, int k) {
2     if (d == null) return new AvlDugum(k);
▶     if (k < d.anahtar)
▶         d.sol = ekle(d.sol, k);
5     else if (k > d.anahtar)
6         d.sag = ekle(d.sag, k);
7     else return d;          // tekrar yok
8     return dengele(d);
9 }
```

DEĞİŞKENLER

`k = 25` `d = düğüm(30)`

AVL Ekleme İzleme (devam): 40, 50, 25



d == null → yeni yaprak **25** döner ve 30'un sol bağına atanır. Dönüşte yol üzerindeki bf'ler değişti.

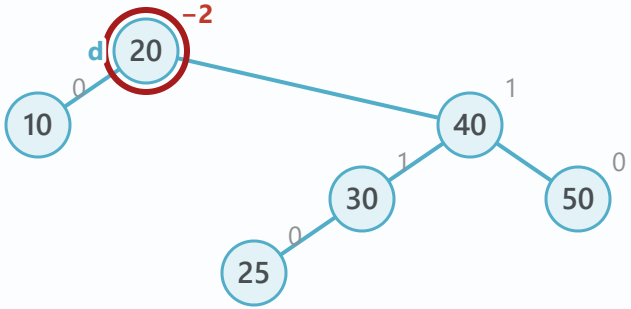
Adım 9 / 12

```
1 AvlDugum ekle(AvlDugum d, int k) {
▶     if (d == null) return new AvlDugum(k);
3     if (k < d.anahtar)
4         d.sol = ekle(d.sol, k);
5     else if (k > d.anahtar)
6         d.sag = ekle(d.sag, k);
7     else return d;          // tekrar yok
8     return dengele(d);
9 }
```

DEĞİŞKENLER

`k = 25` `d = null`

AVL Ekleme İzleme (devam): 40, 50, 25



`dengele(20)` : bf = −2 ✗ (en alttaki dengesiz ata). Sağ çocuk 40'ta bf = 1 → **RL durumu**.

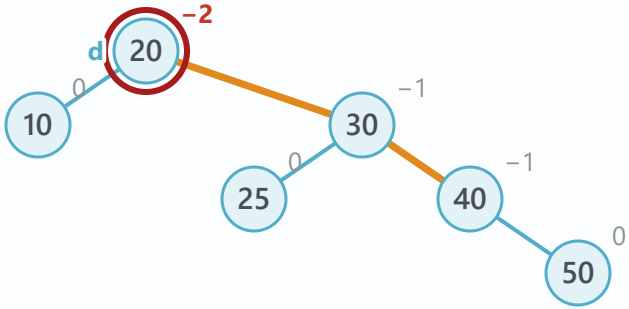
Adım 10 / 12

```
1  AvlDugum ekle(AvlDugum d, int k) {
2      if (d == null) return new AvlDugum(k);
3      if (k < d.anahtar)
4          d.sol = ekle(d.sol, k);
5      else if (k > d.anahtar)
6          d.sag = ekle(d.sag, k);
7      else return d;           // tekrar yok
▶      return dengele(d);
9  }
```

DEĞİŞKENLER

`k = 25` `d = düğüm(20)` `bf(d) = −2` `bf(çocuk) = 1`

AVL Ekleme İzleme (devam): 40, 50, 25



1. döndürme: 40'ta **sagaDondur** → üçgen düz çizgiye (sağ-sağ) döner. bf(20) hâlâ −2.

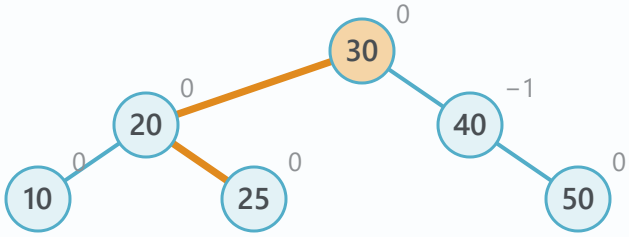
Adım 11 / 12

```
1 AvlDugum ekle(AvlDugum d, int k) {
2     if (d == null) return new AvlDugum(k);
3     if (k < d.anahtar)
4         d.sol = ekle(d.sol, k);
5     else if (k > d.anahtar)
6         d.sag = ekle(d.sag, k);
7     else return d;           // tekrar yok
▶     return dengele(d);
9 }
```

DEĞİŞKENLER

`k = 25` `d = düğüm(20)` `bf(d) = −2`

AVL Ekleme İzleme (devam): 40, 50, 25



2. döndürme: 20'de **solaDondur**: 30 yukarı çıkar; 25, 20'nin sağına geçer. Ağaç yeniden AVL. Tüm bf ∈ {−1, 0, 1}.

Adım 12 / 12

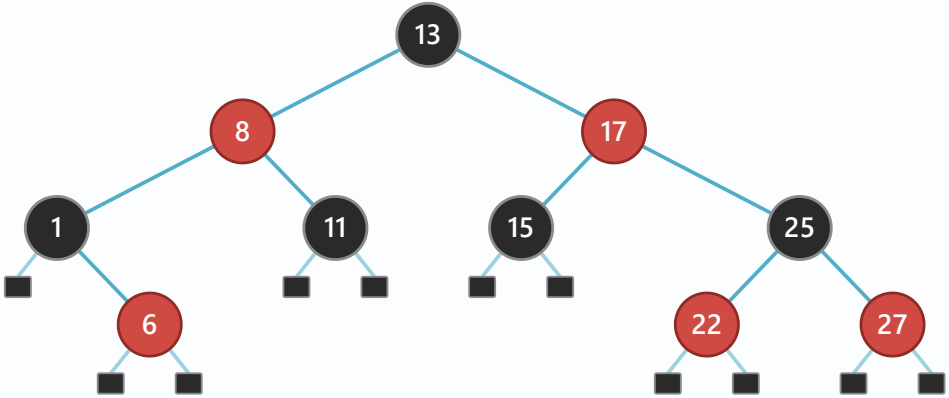
```
1 AvlDugum ekle(AvlDugum d, int k) {
2     if (d == null) return new AvlDugum(k);
3     if (k < d.anahtar)
4         d.sol = ekle(d.sol, k);
5     else if (k > d.anahtar)
6         d.sag = ekle(d.sag, k);
7     else return d;           // tekrar yok
▶     return dengele(d);
9 }
```

DEĞİŞKENLER

`k = 25` `d = düğüm(30)` `bf(d) = 0`

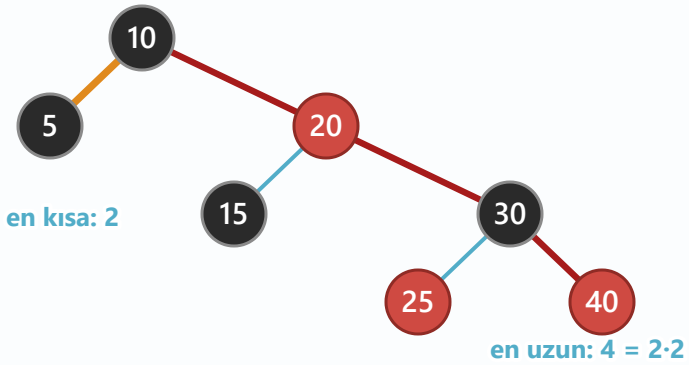
Kırmızı-Siyah Ağaç (Red-Black Tree)

1. Her düğüm **kırmızı** ya da **siyah**tır.
2. **Kök** siyahtır.
3. Tüm yapraklar (**NIL**, boş çocuklar) siyah kabul edilir.
4. Kırmızı düğümün çocukları siyahtır: **ardışık iki kırmızı olmaz**.
5. Bir düğümden NIL yapraklara giden **her yolda aynı sayıda siyah düğüm** vardır (siyah yükseklik).



Örnek: kökten her NIL'e giden yolda 2 siyah düğüm var (kök dahil, NIL hariç)

Kurallar Neden Dengeyi Garanti Eder?



- **En kısa yol:** yalnız siyah düğümlerden oluşabilir → b düğüm (şekilde 10–5: b = 2).
- **En uzun yol:** kırmızılar ardışık olamaz; en fazla her siyahın altına bir kırmızı girer → en çok 2b düğüm (10–20–30–40).
- **Sonuç:** en uzun yol ≤ 2 × en kısa yol; buradan **$h \le 2 \cdot \log_2(n+1)$**.
- **Maliyet:** arama, ekleme, silme O(log n); eklemede en fazla 2, silmede en fazla 3 döndürme, gerisi renk değiştirme.

Kırmızı-siyah ağaç, AVL'ye göre **daha gevşek** dengelidir (biraz daha yüksek olabilir) ama güncellemeleri daha ucuzdur. Java `TreeMap` / `TreeSet` ve C++ `std::map` bu yapıyı kullanır.

Kırmızı-Siyah Ağaca Ekleme

Yeni düğüm BST gibi eklenir ve **kırmızı** boyanır (siyah yükseklik bozulmasın diye). Ebeveyn de kırmızıysa kural 4 bozulur; **amcaya** (ebeveynin kardeşi) bakılır:

Amca kırmızı

Renk değiştir: ebeveyn ve amca siyah, dede kırmızı. Sorun dedeye taşınabilir → oradan devam et.

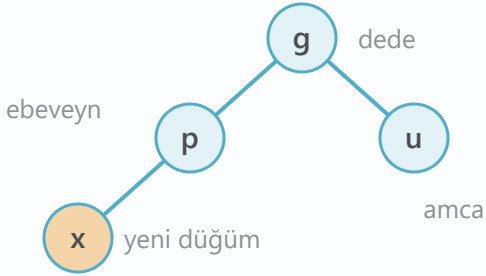
Amca siyah, üçgen

Düğüm-ebeveyn-dede zikzak yapıyorsa ebeveynde döndür → çizgi durumuna dönüşür.

Amca siyah, çizgi

Dededе döndür; eski ebeveyn siyah, eski dede kırmızı olur. Bitti.

- Ebeveyn siyahsa hiçbir şey yapılmaz. En sonda kök her zaman siyaha boyanır.
- NIL amca siyah sayılır.
- Silme benzer ama daha çok durumludur ("çift siyah" düzeltmesi).



Ekleme Sonrası Düzeltme Kodu

```
void duzelt(Dugum x) {      // x: yeni, kırmızı
    while (x != kok && kirmizi(x.ebeveyn)) {
        Dugum p = x.ebeveyn, g = p.ebeveyn;
        Dugum u = (p == g.sol) ? g.sag : g.sol;
        if (kirmizi(u)) {         // amca kırmızı
            p.renk = u.renk = SIYAH;
            g.renk = KIRMIZI;
            x = g;                // sorun dedeye
        } else {                  // amca siyah
            if (ucgen(x, p, g)) { // zikzak
                dondur(p, x);     // çizgiye çevir
                Dugum t = x; x = p; p = t;
            }
            dondur(g, p);         // çizgi
            p.renk = SIYAH; g.renk = KIRMIZI;
        }
    }
    kok.renk = SIYAH;
}
```

```
// Ekleme: BST gibi yaprağa ekle,
// ebeveyni bağla, kırmızı boya:
//   x = bstEkle(anahtar);
//   x.renk = KIRMIZI; duzelt(x);

// NIL (null) siyah sayılır
boolean kirmizi(Dugum n) {
    return n != null
        && n.renk == KIRMIZI;
}
// x–p–g zikzak (üçgen) mı?
boolean ucgen(Dugum x, Dugum p,
              Dugum g) {
    return (p == g.sol)
        != (x == p.sol);
}
// dondur(a, b): b, a'nın çocuğu;
// b'yi a'nın yerine çıkarır (b
// soldaysa sağa, sağdaysa sola
// döndürme; ebeveyn bağları dahil)
```

Kod İzleme: `duzelt` — 10, 20, 30, 15 eklenir

```
▶ void duzelt(Dugum x) {      // x: yeni, kırmızı
2     while (x != kok && kirmizi(x.ebeveyn)) {
3         Dugum p = x.ebeveyn, g = p.ebeveyn;
4         Dugum u = (p == g.sol) ? g.sag : g.sol;
5         if (kirmizi(u)) {         // amca kırmızı
6             p.renk = u.renk = SIYAH;
7             g.renk = KIRMIZI;
8             x = g;                // sorun dedeye
9         } else {                  // amca siyah
10            if (ucgen(x, p, g)) { // zikzak
11                dondur(p, x);     // çizgiye çevir
12                Dugum t = x; x = p; p = t;
13            }
14            dondur(g, p);         // çizgi
15            p.renk = SIYAH; g.renk = KIRMIZI;
16        }
17    }
18    kok.renk = SIYAH;
19 }
```

x 10

10 BST gibi eklendi (kök) ve **kırmızı** boyandı; `duzelt(x)` çağrılır.

Adım 1 / 19

DEĞİŞKENLER

`x = düğüm(10) K`

Kod İzleme: `duzelt` — 10, 20, 30, 15 eklenir

```
1  void duzelt(Dugum x) {      // x: yeni, kırmızı
▶      while (x != kok && kirmizi(x.ebeveyn)) {
3          Dugum p = x.ebeveyn, g = p.ebeveyn;
4          Dugum u = (p == g.sol) ? g.sag : g.sol;
5          if (kirmizi(u)) {         // amca kırmızı
6              p.renk = u.renk = SIYAH;
7              g.renk = KIRMIZI;
8              x = g;                // sorun dedeye
9          } else {                  // amca siyah
10             if (ucgen(x, p, g)) { // zikzak
11                 dondur(p, x);     // çizgiye çevir
12                 Dugum t = x; x = p; p = t;
13             }
14             dondur(g, p);         // çizgi
15             p.renk = SIYAH; g.renk = KIRMIZI;
16         }
17     }
18     kok.renk = SIYAH;
19 }
```

x 10

x kökün kendisi → döngü koşulu yanlış, döngüye girilmez.

Adım 2 / 19

DEĞİŞKENLER

`x = düğüm(10) K`

Kod İzleme: `duzelt` — 10, 20, 30, 15 eklenir

```
1  void duzelt(Dugum x) {      // x: yeni, kırmızı
2      while (x != kok && kirmizi(x.ebeveyn)) {
3          Dugum p = x.ebeveyn, g = p.ebeveyn;
4          Dugum u = (p == g.sol) ? g.sag : g.sol;
5          if (kirmizi(u)) {         // amca kırmızı
6              p.renk = u.renk = SIYAH;
7              g.renk = KIRMIZI;
8              x = g;                // sorun dedeye
9          } else {                  // amca siyah
10             if (ucgen(x, p, g)) { // zikzak
11                 dondur(p, x);     // çizgiye çevir
12                 Dugum t = x; x = p; p = t;
13             }
14             dondur(g, p);         // çizgi
15             p.renk = SIYAH; g.renk = KIRMIZI;
16         }
17     }
▶      kok.renk = SIYAH;
19 }
```



kok.renk = SIYAH: kök her zaman siyah (kural 2).

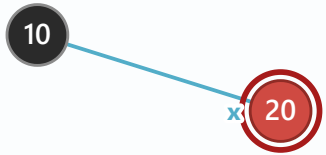
Adım 3 / 19

DEĞİŞKENLER

`x = düğüm(10) S`

Kod İzleme: `duzelt` — 10, 20, 30, 15 eklenir

```
▶ void duzelt(Dugum x) {      // x: yeni, kırmızı
2     while (x != kok && kirmizi(x.ebeveyn)) {
3         Dugum p = x.ebeveyn, g = p.ebeveyn;
4         Dugum u = (p == g.sol) ? g.sag : g.sol;
5         if (kirmizi(u)) {         // amca kırmızı
6             p.renk = u.renk = SIYAH;
7             g.renk = KIRMIZI;
8             x = g;                // sorun dedeye
9         } else {                  // amca siyah
10            if (ucgen(x, p, g)) { // zikzak
11                dondur(p, x);     // çizgiye çevir
12                Dugum t = x; x = p; p = t;
13            }
14            dondur(g, p);         // çizgi
15            p.renk = SIYAH; g.renk = KIRMIZI;
16        }
17    }
18    kok.renk = SIYAH;
19 }
```



20 BST gibi eklendi (10'un sağ çocuğu) ve **kırmızı** boyandı; `duzelt(x)` çağrılır.

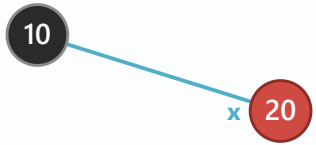
Adım 4 / 19

DEĞİŞKENLER

`x = düğüm(20) K`

Kod İzleme: `duzelt` — 10, 20, 30, 15 eklenir

```
1  void duzelt(Dugum x) {      // x: yeni, kırmızı
▶      while (x != kok && kirmizi(x.ebeveyn)) {
3          Dugum p = x.ebeveyn, g = p.ebeveyn;
4          Dugum u = (p == g.sol) ? g.sag : g.sol;
5          if (kirmizi(u)) {         // amca kırmızı
6              p.renk = u.renk = SIYAH;
7              g.renk = KIRMIZI;
8              x = g;                // sorun dedeye
9          } else {                  // amca siyah
10             if (ucgen(x, p, g)) { // zikzak
11                 dondur(p, x);     // çizgiye çevir
12                 Dugum t = x; x = p; p = t;
13             }
14             dondur(g, p);         // çizgi
15             p.renk = SIYAH; g.renk = KIRMIZI;
16         }
17     }
▶      kok.renk = SIYAH;
19 }
```



x.ebeveyn (10) **siyah** → ardışık kırmızı yok, döngü biter. Satır 18: kök zaten siyah.

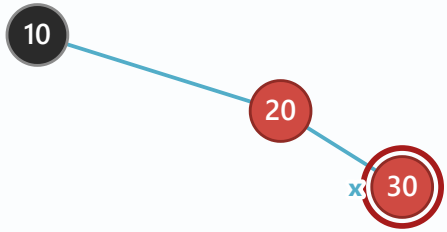
Adım 5 / 19

DEĞİŞKENLER

`x = düğüm(20) K`

Kod İzleme: `duzelt` — 10, 20, 30, 15 eklenir

```
▶ void duzelt(Dugum x) {       // x: yeni, kırmızı
2     while (x != kok && kirmizi(x.ebeveyn)) {
3         Dugum p = x.ebeveyn, g = p.ebeveyn;
4         Dugum u = (p == g.sol) ? g.sag : g.sol;
5         if (kirmizi(u)) {          // amca kırmızı
6             p.renk = u.renk = SIYAH;
7             g.renk = KIRMIZI;
8             x = g;                 // sorun dedeye
9         } else {                   // amca siyah
10            if (ucgen(x, p, g)) { // zikzak
11                dondur(p, x);      // çizgiye çevir
12                Dugum t = x; x = p; p = t;
13            }
14            dondur(g, p);          // çizgi
15            p.renk = SIYAH; g.renk = KIRMIZI;
16        }
17    }
18    kok.renk = SIYAH;
19 }
```



30 BST gibi eklendi (20'nin sağ çocuğu) ve **kırmızı** boyandı; `duzelt(x)` çağrılır.

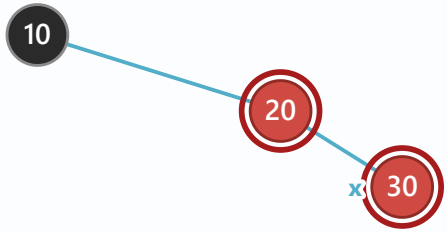
Adım 6 / 19

DEĞİŞKENLER

`x = düğüm(30) K`

Kod İzleme: `duzelt` — 10, 20, 30, 15 eklenir

```
1  void duzelt(Dugum x) {      // x: yeni, kırmızı
▶      while (x != kok && kirmizi(x.ebeveyn)) {
3          Dugum p = x.ebeveyn, g = p.ebeveyn;
4          Dugum u = (p == g.sol) ? g.sag : g.sol;
5          if (kirmizi(u)) {         // amca kırmızı
6              p.renk = u.renk = SIYAH;
7              g.renk = KIRMIZI;
8              x = g;                // sorun dedeye
9          } else {                  // amca siyah
10             if (ucgen(x, p, g)) { // zikzak
11                 dondur(p, x);     // çizgiye çevir
12                 Dugum t = x; x = p; p = t;
13             }
14             dondur(g, p);         // çizgi
15             p.renk = SIYAH; g.renk = KIRMIZI;
16         }
17     }
18     kok.renk = SIYAH;
19 }
```



x.ebeveyn (20) de **kırmızı**: ardışık iki kırmızı *X* (kural 4) → döngüye girilir.

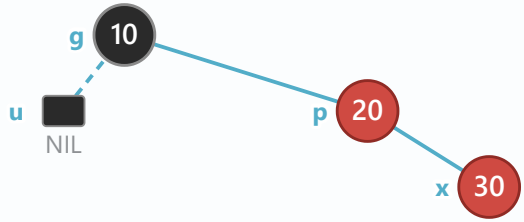
Adım 7 / 19

DEĞİŞKENLER

`x = düğüm(30) K`

Kod İzleme: `duzelt` — 10, 20, 30, 15 eklenir

```
1  void duzelt(Dugum x) {      // x: yeni, kırmızı
2      while (x != kok && kirmizi(x.ebeveyn)) {
▶          Dugum p = x.ebeveyn, g = p.ebeveyn;
▶          Dugum u = (p == g.sol) ? g.sag : g.sol;
5          if (kirmizi(u)) {         // amca kırmızı
6              p.renk = u.renk = SIYAH;
7              g.renk = KIRMIZI;
8              x = g;                // sorun dedeye
9          } else {                  // amca siyah
10             if (ucgen(x, p, g)) { // zikzak
11                 dondur(p, x);     // çizgiye çevir
12                 Dugum t = x; x = p; p = t;
13             }
14             dondur(g, p);         // çizgi
15             p.renk = SIYAH; g.renk = KIRMIZI;
16         }
17     }
18     kok.renk = SIYAH;
19 }
```



p = ebeveyn (20), g = dede (10), u = amca = NIL (boş, siyah sayılır).

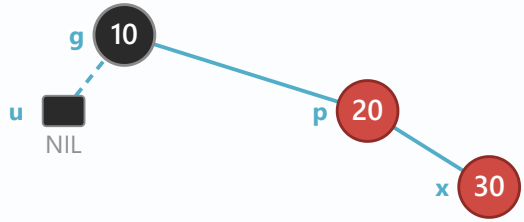
Adım 8 / 19

DEĞİŞKENLER

- x = düğüm(30) K
- p = düğüm(20) K
- g = düğüm(10) S
- u = null (NIL, S)

Kod İzleme: `duzelt` — 10, 20, 30, 15 eklenir

```
1  void duzelt(Dugum x) {      // x: yeni, kırmızı
2      while (x != kok && kirmizi(x.ebeveyn)) {
3          Dugum p = x.ebeveyn, g = p.ebeveyn;
4          Dugum u = (p == g.sol) ? g.sag : g.sol;
▶          if (kirmizi(u)) {          // amca kırmızı
6              p.renk = u.renk = SIYAH;
7              g.renk = KIRMIZI;
8              x = g;                 // sorun dedeye
▶          } else {                   // amca siyah
▶              if (ucgen(x, p, g)) { // zikzak
11                 dondur(p, x);      // çizgiye çevir
12                 Dugum t = x; x = p; p = t;
13             }
14             dondur(g, p);          // çizgi
15             p.renk = SIYAH; g.renk = KIRMIZI;
16         }
17     }
18     kok.renk = SIYAH;
19 }
```



Amca **siyah** → döndürme gerekir. x–p–g **düz çizgi**: ucgen() yanlış, doğrudan satır 14.

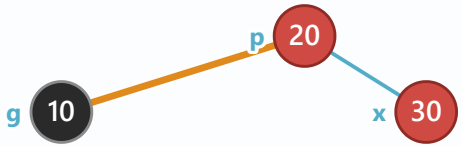
Adım 9 / 19

DEĞİŞKENLER

- x = **düğüm(30) K**
- p = **düğüm(20) K**
- g = **düğüm(10) S**
- u = **null (NIL, S)**

Kod İzleme: `duzelt` — 10, 20, 30, 15 eklenir

```
1  void duzelt(Dugum x) {      // x: yeni, kırmızı
2      while (x != kok && kirmizi(x.ebeveyn)) {
3          Dugum p = x.ebeveyn, g = p.ebeveyn;
4          Dugum u = (p == g.sol) ? g.sag : g.sol;
5          if (kirmizi(u)) {         // amca kırmızı
6              p.renk = u.renk = SIYAH;
7              g.renk = KIRMIZI;
8              x = g;                // sorun dedeye
9          } else {                  // amca siyah
10             if (ucgen(x, p, g)) { // zikzak
11                 dondur(p, x);     // çizgiye çevir
12                 Dugum t = x; x = p; p = t;
13             }
▶              dondur(g, p);         // çizgi
15             p.renk = SIYAH; g.renk = KIRMIZI;
16         }
17     }
18     kok.renk = SIYAH;
19 }
```



`dondur(g, p)`: dede 10'da **sola döndürme** → p (20) yukarı çıkar, g onun çocuğu olur.

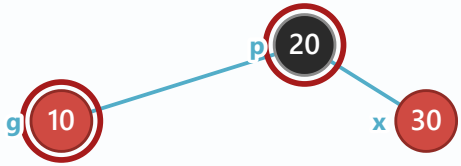
Adım 10 / 19

DEĞİŞKENLER

- x = **düğüm(30) K**
- p = **düğüm(20) K**
- g = **düğüm(10) S**
- u = **null (NIL, S)**

Kod İzleme: `duzelt` — 10, 20, 30, 15 eklenir

```
1  void duzelt(Dugum x) {      // x: yeni, kırmızı
2      while (x != kok && kirmizi(x.ebeveyn)) {
3          Dugum p = x.ebeveyn, g = p.ebeveyn;
4          Dugum u = (p == g.sol) ? g.sag : g.sol;
5          if (kirmizi(u)) {         // amca kırmızı
6              p.renk = u.renk = SIYAH;
7              g.renk = KIRMIZI;
8              x = g;                // sorun dedeye
9          } else {                  // amca siyah
10             if (ucgen(x, p, g)) { // zikzak
11                 dondur(p, x);     // çizgiye çevir
12                 Dugum t = x; x = p; p = t;
13             }
14             dondur(g, p);         // çizgi
▶              p.renk = SIYAH; g.renk = KIRMIZI;
16         }
17     }
18     kok.renk = SIYAH;
19 }
```



Renkler: **p siyah, g kırmızı**. Ardışık kırmızı kalmadı, siyah yükseklik korundu.

Adım 11 / 19

DEĞİŞKENLER

- `x = düğüm(30) K`
- `p = düğüm(20) S`
- `g = düğüm(10) K`
- `u = null (NIL, S)`

Kod İzleme: `duzelt` — 10, 20, 30, 15 eklenir

```
1  void duzelt(Dugum x) {      // x: yeni, kırmızı
▶      while (x != kok && kirmizi(x.ebeveyn)) {
3          Dugum p = x.ebeveyn, g = p.ebeveyn;
4          Dugum u = (p == g.sol) ? g.sag : g.sol;
5          if (kirmizi(u)) {         // amca kırmızı
6              p.renk = u.renk = SIYAH;
7              g.renk = KIRMIZI;
8              x = g;                // sorun dedeye
9          } else {                  // amca siyah
10             if (ucgen(x, p, g)) { // zikzak
11                 dondur(p, x);     // çizgiye çevir
12                 Dugum t = x; x = p; p = t;
13             }
14             dondur(g, p);         // çizgi
15             p.renk = SIYAH; g.renk = KIRMIZI;
16         }
17     }
▶      kok.renk = SIYAH;
19 }
```



x.ebeveyn (20) **siyah** → ardışık kırmızı yok, döngü biter. Satır 18: kök zaten siyah.

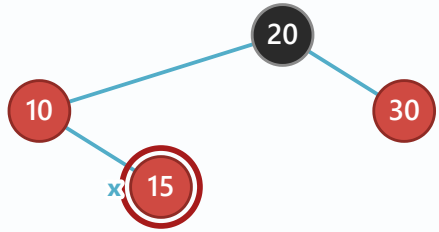
Adım 12 / 19

DEĞİŞKENLER

- x = **düğüm(30) K**
- p = **düğüm(20) S**
- g = **düğüm(10) K**
- u = **null (NIL, S)**

Kod İzleme: `duzelt` — 10, 20, 30, 15 eklenir

```
▶ void duzelt(Dugum x) {       // x: yeni, kırmızı
2     while (x != kok && kirmizi(x.ebeveyn)) {
3         Dugum p = x.ebeveyn, g = p.ebeveyn;
4         Dugum u = (p == g.sol) ? g.sag : g.sol;
5         if (kirmizi(u)) {          // amca kırmızı
6             p.renk = u.renk = SIYAH;
7             g.renk = KIRMIZI;
8             x = g;                 // sorun dedeye
9         } else {                   // amca siyah
10            if (ucgen(x, p, g)) { // zikzak
11                dondur(p, x);      // çizgiye çevir
12                Dugum t = x; x = p; p = t;
13            }
14            dondur(g, p);          // çizgi
15            p.renk = SIYAH; g.renk = KIRMIZI;
16        }
17    }
18    kok.renk = SIYAH;
19 }
```



15 BST gibi eklendi (10'un sağ çocuğu) ve **kırmızı** boyandı; `duzelt(x)` çağrılır.

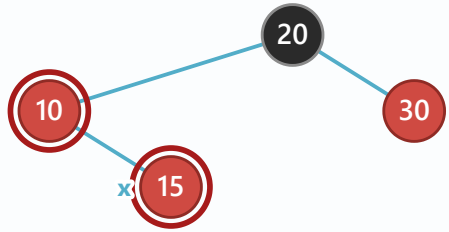
Adım 13 / 19

DEĞİŞKENLER

`x = düğüm(15) K`

Kod İzleme: `duzelt` — 10, 20, 30, 15 eklenir

```
1  void duzelt(Dugum x) {      // x: yeni, kırmızı
▶      while (x != kok && kirmizi(x.ebeveyn)) {
3          Dugum p = x.ebeveyn, g = p.ebeveyn;
4          Dugum u = (p == g.sol) ? g.sag : g.sol;
5          if (kirmizi(u)) {         // amca kırmızı
6              p.renk = u.renk = SIYAH;
7              g.renk = KIRMIZI;
8              x = g;                // sorun dedeye
9          } else {                  // amca siyah
10             if (ucgen(x, p, g)) { // zikzak
11                 dondur(p, x);     // çizgiye çevir
12                 Dugum t = x; x = p; p = t;
13             }
14             dondur(g, p);         // çizgi
15             p.renk = SIYAH; g.renk = KIRMIZI;
16         }
17     }
18     kok.renk = SIYAH;
19 }
```



x.ebeveyn (10) de **kırmızı**: ardışık iki kırmızı *X* (kural 4) → döngüye girilir.

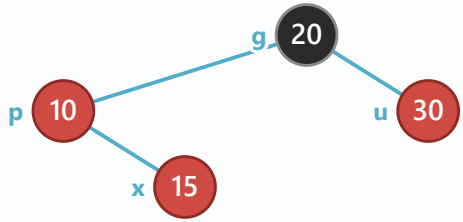
Adım 14 / 19

DEĞİŞKENLER

`x = düğüm(15) K`

Kod İzleme: `duzelt` — 10, 20, 30, 15 eklenir

```
1  void duzelt(Dugum x) {      // x: yeni, kırmızı
2      while (x != kok && kirmizi(x.ebeveyn)) {
▶          Dugum p = x.ebeveyn, g = p.ebeveyn;
▶          Dugum u = (p == g.sol) ? g.sag : g.sol;
5          if (kirmizi(u)) {         // amca kırmızı
6              p.renk = u.renk = SIYAH;
7              g.renk = KIRMIZI;
8              x = g;                // sorun dedeye
9          } else {                  // amca siyah
10             if (ucgen(x, p, g)) { // zikzak
11                 dondur(p, x);     // çizgiye çevir
12                 Dugum t = x; x = p; p = t;
13             }
14             dondur(g, p);         // çizgi
15             p.renk = SIYAH; g.renk = KIRMIZI;
16         }
17     }
18     kok.renk = SIYAH;
19 }
```



p = ebeveyn (10), g = dede (20), u = amca = 30.

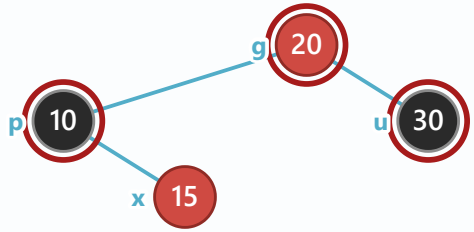
Adım 15 / 19

DEĞİŞKENLER

- x = düğüm(15) K
- p = düğüm(10) K
- g = düğüm(20) S
- u = düğüm(30) K

Kod İzleme: `duzelt` — 10, 20, 30, 15 eklenir

```
1  void duzelt(Dugum x) {      // x: yeni, kırmızı
2      while (x != kok && kirmizi(x.ebeveyn)) {
3          Dugum p = x.ebeveyn, g = p.ebeveyn;
4          Dugum u = (p == g.sol) ? g.sag : g.sol;
▶          if (kirmizi(u)) {          // amca kırmızı
▶              p.renk = u.renk = SIYAH;
▶              g.renk = KIRMIZI;
8              x = g;                 // sorun dedeye
9          } else {                   // amca siyah
10             if (ucgen(x, p, g)) { // zikzak
11                 dondur(p, x);      // çizgiye çevir
12                 Dugum t = x; x = p; p = t;
13             }
14             dondur(g, p);          // çizgi
15             p.renk = SIYAH; g.renk = KIRMIZI;
16         }
17     }
18     kok.renk = SIYAH;
19 }
```



Amca kırmızı → yalnızca renk değiştir: p ve u siyah, g kırmızı. Her yolda siyah sayısı aynı kalır.

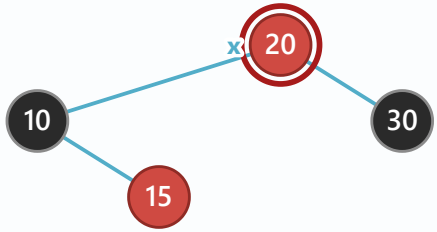
Adım 16 / 19

DEĞİŞKENLER

x = düğüm(15) K	p = düğüm(10) S
g = düğüm(20) K	u = düğüm(30) S

Kod İzleme: `duzelt` — 10, 20, 30, 15 eklenir

```
1  void duzelt(Dugum x) {      // x: yeni, kırmızı
2      while (x != kok && kirmizi(x.ebeveyn)) {
3          Dugum p = x.ebeveyn, g = p.ebeveyn;
4          Dugum u = (p == g.sol) ? g.sag : g.sol;
5          if (kirmizi(u)) {         // amca kırmızı
6              p.renk = u.renk = SIYAH;
7              g.renk = KIRMIZI;
▶              x = g;                // sorun dedeye
9          } else {                  // amca siyah
10             if (ucgen(x, p, g)) { // zikzak
11                 dondur(p, x);     // çizgiye çevir
12                 Dugum t = x; x = p; p = t;
13             }
14             dondur(g, p);         // çizgi
15             p.renk = SIYAH; g.renk = KIRMIZI;
16         }
17     }
18     kok.renk = SIYAH;
19 }
```



x = g (20): g kırmızı oldu; onun ebeveyni de kırmızı olabilir → döngü dededen devam eder.

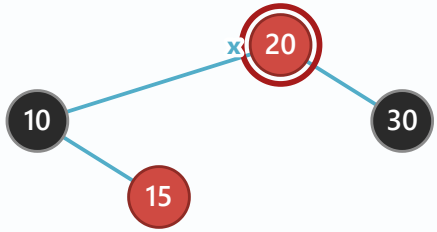
Adım 17 / 19

DEĞİŞKENLER

`x = düğüm(20) K`

Kod İzleme: `duzelt` — 10, 20, 30, 15 eklenir

```
1  void duzelt(Dugum x) {      // x: yeni, kırmızı
▶      while (x != kok && kirmizi(x.ebeveyn)) {
3          Dugum p = x.ebeveyn, g = p.ebeveyn;
4          Dugum u = (p == g.sol) ? g.sag : g.sol;
5          if (kirmizi(u)) {         // amca kırmızı
6              p.renk = u.renk = SIYAH;
7              g.renk = KIRMIZI;
8              x = g;                // sorun dedeye
9          } else {                  // amca siyah
10             if (ucgen(x, p, g)) { // zikzak
11                 dondur(p, x);     // çizgiye çevir
12                 Dugum t = x; x = p; p = t;
13             }
14             dondur(g, p);         // çizgi
15             p.renk = SIYAH; g.renk = KIRMIZI;
16         }
17     }
18     kok.renk = SIYAH;
19 }
```



x kökün kendisi → döngü koşulu yanlış, döngüye girilmez.

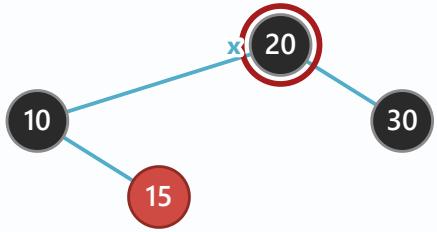
Adım 18 / 19

DEĞİŞKENLER

`x = düğüm(20) K`

Kod İzleme: `duzelt` — 10, 20, 30, 15 eklenir

```
1  void duzelt(Dugum x) {      // x: yeni, kırmızı
2      while (x != kok && kirmizi(x.ebeveyn)) {
3          Dugum p = x.ebeveyn, g = p.ebeveyn;
4          Dugum u = (p == g.sol) ? g.sag : g.sol;
5          if (kirmizi(u)) {         // amca kırmızı
6              p.renk = u.renk = SIYAH;
7              g.renk = KIRMIZI;
8              x = g;                // sorun dedeye
9          } else {                  // amca siyah
10             if (ucgen(x, p, g)) { // zikzak
11                 dondur(p, x);     // çizgiye çevir
12                 Dugum t = x; x = p; p = t;
13             }
14             dondur(g, p);         // çizgi
15             p.renk = SIYAH; g.renk = KIRMIZI;
16         }
17     }
▶      kok.renk = SIYAH;
19 }
```



kok.renk = SIYAH: kök her zaman siyah (kural 2). Siyah yükseklik 1 arttı.

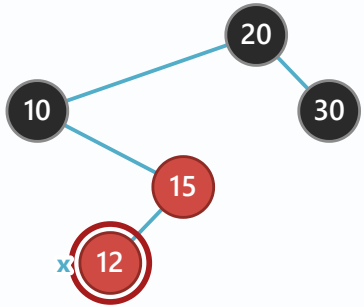
Adım 19 / 19

DEĞİŞKENLER

`x = düğüm(20) S`

Kod İzleme: `duzelt` — 12 ve 5 eklenir

```
▶ void duzelt(Dugum x) {       // x: yeni, kırmızı
2     while (x != kok && kirmizi(x.ebeveyn)) {
3         Dugum p = x.ebeveyn, g = p.ebeveyn;
4         Dugum u = (p == g.sol) ? g.sag : g.sol;
5         if (kirmizi(u)) {          // amca kırmızı
6             p.renk = u.renk = SIYAH;
7             g.renk = KIRMIZI;
8             x = g;                 // sorun dedeye
9         } else {                   // amca siyah
10            if (ucgen(x, p, g)) { // zikzak
11                dondur(p, x);      // çizgiye çevir
12                Dugum t = x; x = p; p = t;
13            }
14            dondur(g, p);          // çizgi
15            p.renk = SIYAH; g.renk = KIRMIZI;
16        }
17    }
18    kok.renk = SIYAH;
19 }
```



12 BST gibi eklendi (15'in sol çocuğu) ve **kırmızı** boyandı; `duzelt(x)` çağrılır.

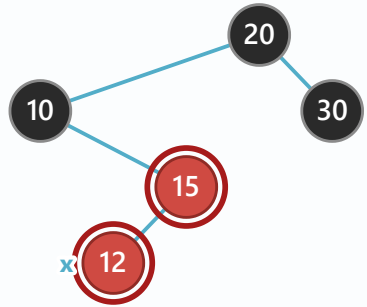
Adım 1 / 15

DEĞİŞKENLER

`x = düğüm(12) K`

Kod İzleme: `duzelt` — 12 ve 5 eklenir

```
1  void duzelt(Dugum x) {      // x: yeni, kırmızı
▶      while (x != kok && kirmizi(x.ebeveyn)) {
3          Dugum p = x.ebeveyn, g = p.ebeveyn;
4          Dugum u = (p == g.sol) ? g.sag : g.sol;
5          if (kirmizi(u)) {         // amca kırmızı
6              p.renk = u.renk = SIYAH;
7              g.renk = KIRMIZI;
8              x = g;                // sorun dedeye
9          } else {                  // amca siyah
10             if (ucgen(x, p, g)) { // zikzak
11                 dondur(p, x);     // çizgiye çevir
12                 Dugum t = x; x = p; p = t;
13             }
14             dondur(g, p);         // çizgi
15             p.renk = SIYAH; g.renk = KIRMIZI;
16         }
17     }
18     kok.renk = SIYAH;
19 }
```



x.ebeveyn (15) de **kırmızı**: ardışık iki kırmızı *X* (kural 4) → döngüye girilir.

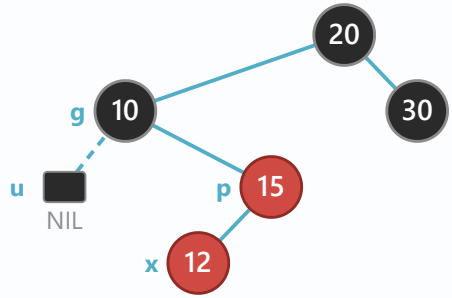
Adım 2 / 15

DEĞİŞKENLER

`x = düğüm(12) K`

Kod İzleme: `duzelt` — 12 ve 5 eklenir

```
1  void duzelt(Dugum x) {      // x: yeni, kırmızı
2      while (x != kok && kirmizi(x.ebeveyn)) {
▶          Dugum p = x.ebeveyn, g = p.ebeveyn;
▶          Dugum u = (p == g.sol) ? g.sag : g.sol;
5          if (kirmizi(u)) {         // amca kırmızı
6              p.renk = u.renk = SIYAH;
7              g.renk = KIRMIZI;
8              x = g;                // sorun dedeye
9          } else {                  // amca siyah
10             if (ucgen(x, p, g)) { // zikzak
11                 dondur(p, x);     // çizgiye çevir
12                 Dugum t = x; x = p; p = t;
13             }
14             dondur(g, p);         // çizgi
15             p.renk = SIYAH; g.renk = KIRMIZI;
16         }
17     }
18     kok.renk = SIYAH;
19 }
```



p = ebeveyn (15), g = dede (10), u = amca = NIL (boş, siyah sayılır).

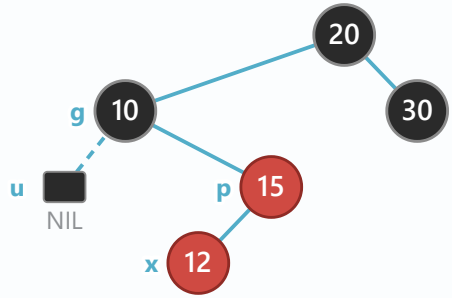
Adım 3 / 15

DEĞİŞKENLER

- `x = düğüm(12) K`
- `p = düğüm(15) K`
- `g = düğüm(10) S`
- `u = null (NIL, S)`

Kod İzleme: `duzelt` — 12 ve 5 eklenir

```
1  void duzelt(Dugum x) {      // x: yeni, kırmızı
2      while (x != kok && kirmizi(x.ebeveyn)) {
3          Dugum p = x.ebeveyn, g = p.ebeveyn;
4          Dugum u = (p == g.sol) ? g.sag : g.sol;
▶          if (kirmizi(u)) {          // amca kırmızı
6              p.renk = u.renk = SIYAH;
7              g.renk = KIRMIZI;
8              x = g;                 // sorun dedeye
▶          } else {                   // amca siyah
▶              if (ucgen(x, p, g)) { // zikzak
11                 dondur(p, x);      // çizgiye çevir
12                 Dugum t = x; x = p; p = t;
13             }
14             dondur(g, p);          // çizgi
15             p.renk = SIYAH; g.renk = KIRMIZI;
16         }
17     }
18     kok.renk = SIYAH;
19 }
```



Amca **siyah** → döndürme gerekir. x–p–g **üçgen** (zikzak): x ile p zıt yönde çocuk.

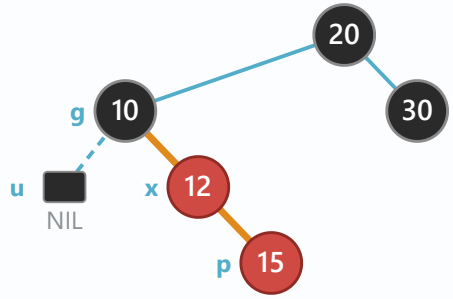
Adım 4 / 15

DEĞİŞKENLER

- x = **düğüm(12) K**
- p = **düğüm(15) K**
- g = **düğüm(10) S**
- u = **null (NIL, S)**

Kod İzleme: `duzelt` — 12 ve 5 eklenir

```
1  void duzelt(Dugum x) {      // x: yeni, kırmızı
2      while (x != kok && kirmizi(x.ebeveyn)) {
3          Dugum p = x.ebeveyn, g = p.ebeveyn;
4          Dugum u = (p == g.sol) ? g.sag : g.sol;
5          if (kirmizi(u)) {          // amca kırmızı
6              p.renk = u.renk = SIYAH;
7              g.renk = KIRMIZI;
8              x = g;                 // sorun dedeye
9          } else {                   // amca siyah
10             if (ucgen(x, p, g)) { // zikzak
▶                  dondur(p, x);     // çizgiye çevir
12                 Dugum t = x; x = p; p = t;
13             }
14             dondur(g, p);          // çizgi
15             p.renk = SIYAH; g.renk = KIRMIZI;
16         }
17     }
18     kok.renk = SIYAH;
19 }
```



`dondur(p, x)`: 15'te sağa döndürme → x yukarı, p aşağı. Üçgen düz çizgiye döndü (hâlâ ardışık kırmızı).

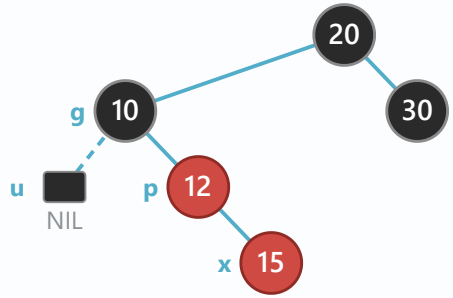
Adım 5 / 15

DEĞİŞKENLER

- x = **düğüm(12) K**
- p = **düğüm(15) K**
- g = **düğüm(10) S**
- u = **null (NIL, S)**

Kod İzleme: `duzelt` — 12 ve 5 eklenir

```
1  void duzelt(Dugum x) {      // x: yeni, kırmızı
2      while (x != kok && kirmizi(x.ebeveyn)) {
3          Dugum p = x.ebeveyn, g = p.ebeveyn;
4          Dugum u = (p == g.sol) ? g.sag : g.sol;
5          if (kirmizi(u)) {         // amca kırmızı
6              p.renk = u.renk = SIYAH;
7              g.renk = KIRMIZI;
8              x = g;                // sorun dedeye
9          } else {                  // amca siyah
10             if (ucgen(x, p, g)) { // zikzak
11                 dondur(p, x);     // çizgiye çevir
▶                  Dugum t = x; x = p; p = t;
13             }
14             dondur(g, p);         // çizgi
15             p.renk = SIYAH; g.renk = KIRMIZI;
16         }
17     }
18     kok.renk = SIYAH;
19 }
```



x ile p rolleri değişir: artık **x = 15** (alttaki), **p = 12**.

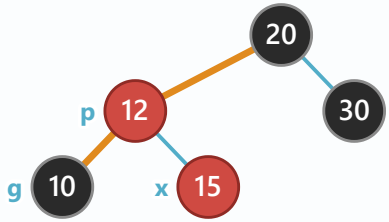
Adım 6 / 15

DEĞİŞKENLER

- x = **düğüm(15) K**
- p = **düğüm(12) K**
- g = **düğüm(10) S**
- u = **null (NIL, S)**

Kod İzleme: `duzelt` — 12 ve 5 eklenir

```
1  void duzelt(Dugum x) {      // x: yeni, kırmızı
2      while (x != kok && kirmizi(x.ebeveyn)) {
3          Dugum p = x.ebeveyn, g = p.ebeveyn;
4          Dugum u = (p == g.sol) ? g.sag : g.sol;
5          if (kirmizi(u)) {         // amca kırmızı
6              p.renk = u.renk = SIYAH;
7              g.renk = KIRMIZI;
8              x = g;                // sorun dedeye
9          } else {                  // amca siyah
10             if (ucgen(x, p, g)) { // zikzak
11                 dondur(p, x);     // çizgiye çevir
12                 Dugum t = x; x = p; p = t;
13             }
▶              dondur(g, p);         // çizgi
15             p.renk = SIYAH; g.renk = KIRMIZI;
16         }
17     }
18     kok.renk = SIYAH;
19 }
```



`dondur(g, p)`: dede 10'da **sola döndürme** → p (12) yukarı çıkar, g onun çocuğu olur.

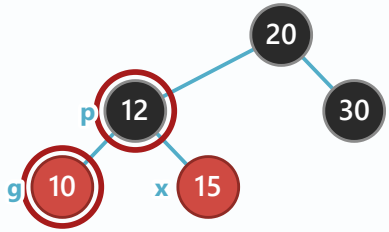
Adım 7 / 15

DEĞİŞKENLER

- `x = düğüm(15) K`
- `p = düğüm(12) K`
- `g = düğüm(10) S`
- `u = null (NIL, S)`

Kod İzleme: `duzelt` — 12 ve 5 eklenir

```
1  void duzelt(Dugum x) {      // x: yeni, kırmızı
2      while (x != kok && kirmizi(x.ebeveyn)) {
3          Dugum p = x.ebeveyn, g = p.ebeveyn;
4          Dugum u = (p == g.sol) ? g.sag : g.sol;
5          if (kirmizi(u)) {         // amca kırmızı
6              p.renk = u.renk = SIYAH;
7              g.renk = KIRMIZI;
8              x = g;                // sorun dedeye
9          } else {                  // amca siyah
10             if (ucgen(x, p, g)) { // zikzak
11                 dondur(p, x);     // çizgiye çevir
12                 Dugum t = x; x = p; p = t;
13             }
14             dondur(g, p);         // çizgi
▶              p.renk = SIYAH; g.renk = KIRMIZI;
16         }
17     }
18     kok.renk = SIYAH;
19 }
```



Renkler: **p siyah, g kırmızı**. Ardışık kırmızı kalmadı, siyah yükseklik korundu.

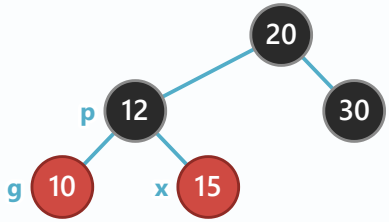
Adım 8 / 15

DEĞİŞKENLER

- `x = düğüm(15) K`
- `p = düğüm(12) S`
- `g = düğüm(10) K`
- `u = null (NIL, S)`

Kod İzleme: `duzelt` — 12 ve 5 eklenir

```
1  void duzelt(Dugum x) {      // x: yeni, kırmızı
▶      while (x != kok && kirmizi(x.ebeveyn)) {
3          Dugum p = x.ebeveyn, g = p.ebeveyn;
4          Dugum u = (p == g.sol) ? g.sag : g.sol;
5          if (kirmizi(u)) {          // amca kırmızı
6              p.renk = u.renk = SIYAH;
7              g.renk = KIRMIZI;
8              x = g;                 // sorun dedeye
9          } else {                   // amca siyah
10             if (ucgen(x, p, g)) { // zikzak
11                 dondur(p, x);      // çizgiye çevir
12                 Dugum t = x; x = p; p = t;
13             }
14             dondur(g, p);          // çizgi
15             p.renk = SIYAH; g.renk = KIRMIZI;
16         }
17     }
▶      kok.renk = SIYAH;
19 }
```



x.ebeveyn (12) **siyah** → ardışık kırmızı yok, döngü biter. Satır 18: kök zaten siyah.

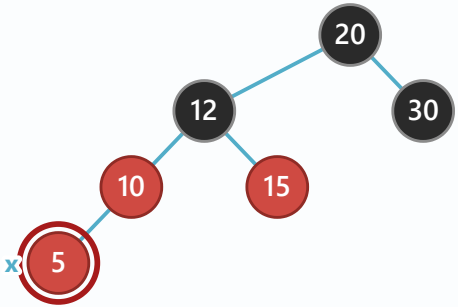
Adım 9 / 15

DEĞİŞKENLER

- x = **düğüm(15) K**
- p = **düğüm(12) S**
- g = **düğüm(10) K**
- u = **null (NIL, S)**

Kod İzleme: `duzelt` — 12 ve 5 eklenir

```
▶ void duzelt(Dugum x) {      // x: yeni, kırmızı
2     while (x != kok && kirmizi(x.ebeveyn)) {
3         Dugum p = x.ebeveyn, g = p.ebeveyn;
4         Dugum u = (p == g.sol) ? g.sag : g.sol;
5         if (kirmizi(u)) {         // amca kırmızı
6             p.renk = u.renk = SIYAH;
7             g.renk = KIRMIZI;
8             x = g;                // sorun dedeye
9         } else {                  // amca siyah
10            if (ucgen(x, p, g)) { // zikzak
11                dondur(p, x);     // çizgiye çevir
12                Dugum t = x; x = p; p = t;
13            }
14            dondur(g, p);         // çizgi
15            p.renk = SIYAH; g.renk = KIRMIZI;
16        }
17    }
18    kok.renk = SIYAH;
19 }
```



5 BST gibi eklendi (10'un sol çocuğu) ve **kırmızı** boyandı; `duzelt(x)` çağrılır.

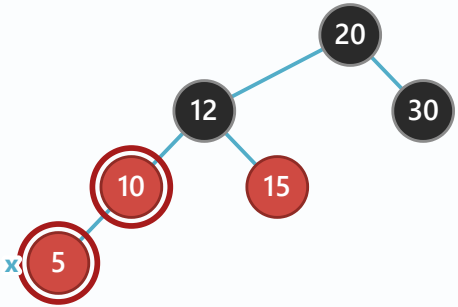
Adım 10 / 15

DEĞİŞKENLER

`x = düğüm(5) K`

Kod İzleme: `duzelt` — 12 ve 5 eklenir

```
1  void duzelt(Dugum x) {      // x: yeni, kırmızı
▶      while (x != kok && kirmizi(x.ebeveyn)) {
3          Dugum p = x.ebeveyn, g = p.ebeveyn;
4          Dugum u = (p == g.sol) ? g.sag : g.sol;
5          if (kirmizi(u)) {         // amca kırmızı
6              p.renk = u.renk = SIYAH;
7              g.renk = KIRMIZI;
8              x = g;                // sorun dedeye
9          } else {                  // amca siyah
10             if (ucgen(x, p, g)) { // zikzak
11                 dondur(p, x);     // çizgiye çevir
12                 Dugum t = x; x = p; p = t;
13             }
14             dondur(g, p);         // çizgi
15             p.renk = SIYAH; g.renk = KIRMIZI;
16         }
17     }
18     kok.renk = SIYAH;
19 }
```



x.ebeveyn (10) de **kırmızı**: ardışık iki kırmızı ✗ (kural 4) → döngüye girilir.

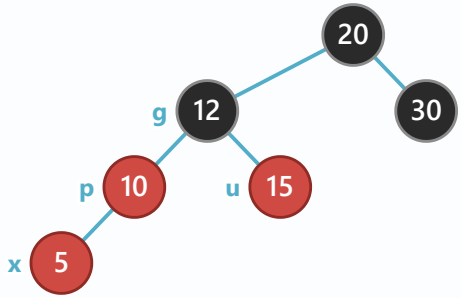
Adım 11 / 15

DEĞİŞKENLER

`x = düğüm(5) K`

Kod İzleme: `duzelt` — 12 ve 5 eklenir

```
1  void duzelt(Dugum x) {      // x: yeni, kırmızı
2      while (x != kok && kirmizi(x.ebeveyn)) {
▶          Dugum p = x.ebeveyn, g = p.ebeveyn;
▶          Dugum u = (p == g.sol) ? g.sag : g.sol;
5          if (kirmizi(u)) {          // amca kırmızı
6              p.renk = u.renk = SIYAH;
7              g.renk = KIRMIZI;
8              x = g;                 // sorun dedeye
9          } else {                   // amca siyah
10             if (ucgen(x, p, g)) { // zikzak
11                 dondur(p, x);      // çizgiye çevir
12                 Dugum t = x; x = p; p = t;
13             }
14             dondur(g, p);          // çizgi
15             p.renk = SIYAH; g.renk = KIRMIZI;
16         }
17     }
18     kok.renk = SIYAH;
19 }
```



p = ebeveyn (10), g = dede (12), u = amca = 15.

Adım 12 / 15

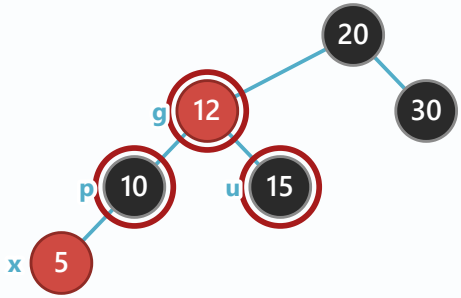
DEĞİŞKENLER

`x = düğüm(5) K` `p = düğüm(10) K`

`g = düğüm(12) S` `u = düğüm(15) K`

Kod İzleme: `duzelt` — 12 ve 5 eklenir

```
1  void duzelt(Dugum x) {      // x: yeni, kırmızı
2      while (x != kok && kirmizi(x.ebeveyn)) {
3          Dugum p = x.ebeveyn, g = p.ebeveyn;
4          Dugum u = (p == g.sol) ? g.sag : g.sol;
▶          if (kirmizi(u)) {          // amca kırmızı
▶              p.renk = u.renk = SIYAH;
▶              g.renk = KIRMIZI;
8              x = g;                 // sorun dedeye
9          } else {                   // amca siyah
10             if (ucgen(x, p, g)) { // zikzak
11                 dondur(p, x);      // çizgiye çevir
12                 Dugum t = x; x = p; p = t;
13             }
14             dondur(g, p);          // çizgi
15             p.renk = SIYAH; g.renk = KIRMIZI;
16         }
17     }
18     kok.renk = SIYAH;
19 }
```



Amca kırmızı → yalnızca renk değiştir: p ve u siyah, g kırmızı. Her yolda siyah sayısı aynı kalır.

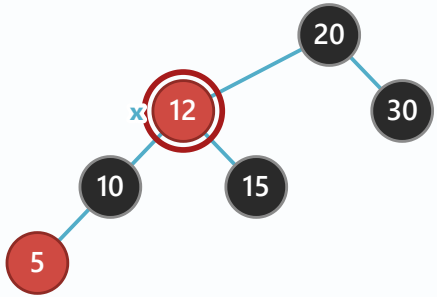
Adım 13 / 15

DEĞİŞKENLER

x = düğüm(5) K | p = düğüm(10) S | g = düğüm(12) K | u = düğüm(15) S

Kod İzleme: `duzelt` — 12 ve 5 eklenir

```
1  void duzelt(Dugum x) {      // x: yeni, kırmızı
2      while (x != kok && kirmizi(x.ebeveyn)) {
3          Dugum p = x.ebeveyn, g = p.ebeveyn;
4          Dugum u = (p == g.sol) ? g.sag : g.sol;
5          if (kirmizi(u)) {          // amca kırmızı
6              p.renk = u.renk = SIYAH;
7              g.renk = KIRMIZI;
▶              x = g;                 // sorun dedeye
9          } else {                   // amca siyah
10             if (ucgen(x, p, g)) { // zikzak
11                 dondur(p, x);      // çizgiye çevir
12                 Dugum t = x; x = p; p = t;
13             }
14             dondur(g, p);          // çizgi
15             p.renk = SIYAH; g.renk = KIRMIZI;
16         }
17     }
18     kok.renk = SIYAH;
19 }
```



x = g (12): g kırmızı oldu; onun ebeveyni de kırmızı olabilir → döngü dededen devam eder.

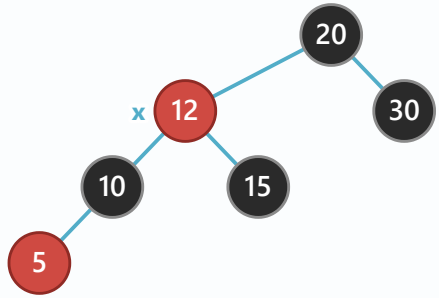
Adım 14 / 15

DEĞİŞKENLER

`x = düğüm(12) K`

Kod İzleme: `duzelt` — 12 ve 5 eklenir

```
1  void duzelt(Dugum x) {      // x: yeni, kırmızı
▶      while (x != kok && kirmizi(x.ebeveyn)) {
3          Dugum p = x.ebeveyn, g = p.ebeveyn;
4          Dugum u = (p == g.sol) ? g.sag : g.sol;
5          if (kirmizi(u)) {          // amca kırmızı
6              p.renk = u.renk = SIYAH;
7              g.renk = KIRMIZI;
8              x = g;                 // sorun dedeye
9          } else {                   // amca siyah
10             if (ucgen(x, p, g)) { // zikzak
11                 dondur(p, x);      // çizgiye çevir
12                 Dugum t = x; x = p; p = t;
13             }
14             dondur(g, p);          // çizgi
15             p.renk = SIYAH; g.renk = KIRMIZI;
16         }
17     }
▶      kok.renk = SIYAH;
19 }
```



x.ebeveyn (20) **siyah** → ardışık kırmızı yok, döngü biter. Satır 18: kök zaten siyah.

Adım 15 / 15

DEĞİŞKENLER

`x = düğüm(12) K`

AVL ve Kırmızı-Siyah Ağaçlarda Silme

Ortak başlangıç

Önce normal BST silmesi uygulanır. İki çocuklu düğüm, ardılı veya öncülüyle değiştirilir; gerçek silme yaprak ya da tek çocuk durumuna iner.

Neden ek işlem gerekir?

Düğüm kaldırmak AVL yükseklik farkını veya kırmızı-siyah ağacın siyah yükseklik kuralını bozabilir.

AVL düzeltmesi

Silinen düğümden köke doğru yükseklikleri güncelle. |bf| = 2 olan her düğümde LL, RR, LR veya RL döndürmesi uygula. Birden fazla atada düzeltme gerekebilir.

Kırmızı-siyah düzeltmesi

Kırmızı düğümün silinmesi kolaydır. Siyah düğüm eksilirse kardeşin rengine ve çocuklarına göre yeniden renklendirme/döndürme yap; eksik siyahlık köke taşınabilir.

Yapı	Yukarı doğru inceleme	Döndürme	Toplam
AVL	O(log n)	O(log n) olabilir	O(log n)
Kırmızı-siyah	O(log n)	en fazla 3	O(log n)

Ekleme ve silme BST sırasını korur; dengeleme yalnızca ağacın biçimini ve yardımcı bilgilerini değiştirir.

AVL mi, Kırmızı-Siyah mı?

Özellik	AVL	Kırmızı-Siyah
Denge ölçütü	$\|bf\| \leq 1$ her düğümde	renk kuralları, eşit siyah yükseklik
En büyük yükseklik	$\approx 1{,}44 \log_2 n$	$2 \log_2(n + 1)$
Arama	O(log n), biraz daha hızlı (daha kısa)	O(log n)
Ekleme döndürmesi	en fazla 2 (bir tek veya bir çift)	en fazla 2
Silme döndürmesi	O(log n) olabilir	en fazla 3
Düğümde ek bilgi	yükseklik (veya bf)	1 bit renk
Tercih	aramanın baskın olduğu işler	sık ekleme/silme, kütüphaneler

Splay Ağacı (Splay Tree)

- Sleator ve Tarjan (1985): **kendini düzenleyen** (self-adjusting) BST.
- Her erişimde (arama, ekleme, silme) ilgili düğüm döndürmelerle **köke taşınır**: *splay* işlemi.
- Düğümlerde denge bilgisi (yükseklik, renk) tutulmaz.
- Tek bir işlem O(n) sürebilir; ama m işlemlik her dizi O(m log n) sürer: **amortize O(log n)**.

Yerellik
Sık erişilen anahtarlar köke yakın durur; önbellek gibi davranır.

Basitlik
Ek alan yok, tek bir splay yordamı tüm işlemlere yeter.

Dezavantaj
Arama bile ağacı değiştirir; tek işlem için garanti yok.

Splay Adımları: Zig, Zig-Zig, Zig-Zag

x: splay edilen düğüm, p: ebeveyni, g: dedesi. x kök olana kadar bu adımlar tekrarlanır.



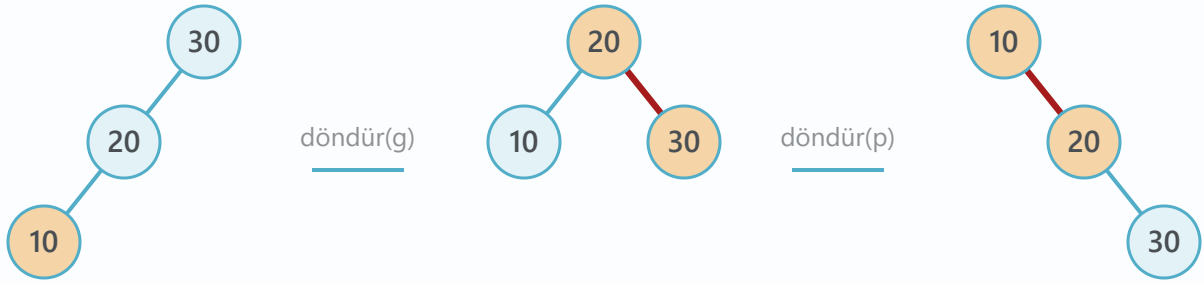
x = 10, ebeveyni p = 20 kök

x kök oldu; 15, p'nin soluna geçti

Zig: x'in ebeveyni p köktür → p'de tek döndürme. (Yalnızca son adımda olabilir.)

Splay Adımları: Zig, Zig-Zig, Zig-Zag

x: splay edilen düğüm, p: ebeveyni, g: dedesi. x kök olana kadar bu adımlar tekrarlanır.

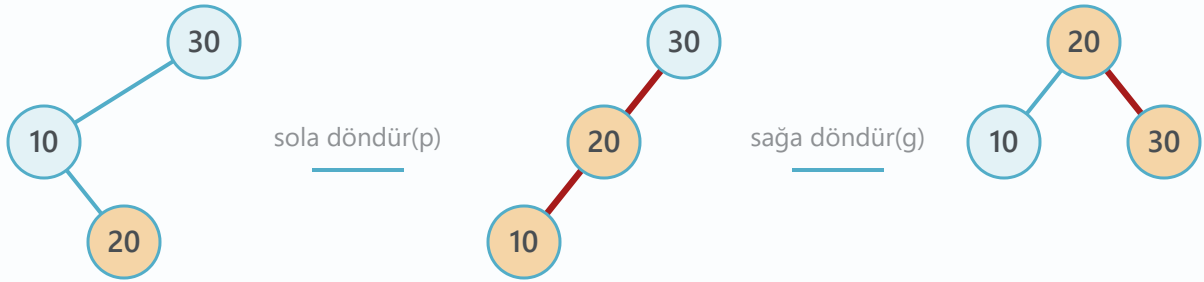


x, p, g aynı yönde (sol-sol) önce dede g = 30 sonra ebeveyn p = 20

Zig-zig: x ve p aynı yönde çocuk → **önce g'de**, sonra p'de döndür. (AVL'deki tek döndürmeden farklı!)

Splay Adımları: Zig, Zig-Zig, Zig-Zag

x: splay edilen düğüm, p: ebeveyni, g: dedesi. x kök olana kadar bu adımlar tekrarlanır.



x sağ, p sol çocuk (zikzak) — önce ebeveyn p = 10 — sonra dede g = 30

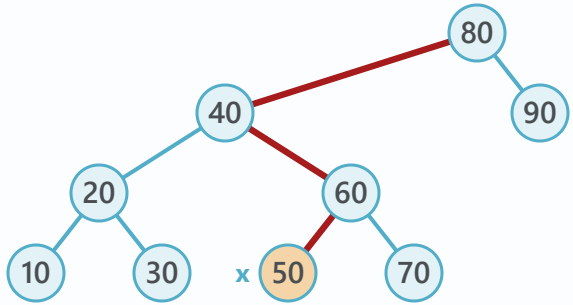
Zig-zag: x ve p zıt yönde → önce p'de, sonra g'de döndür (AVL çift döndürmesiyle aynı).

Kod İzleme: `splay(50)`, sonra `splay(10)`

```
▶ void splay(Dugum x) {
2     while (x != kok) {
3         Dugum p = x.ebeveyn, g = p.ebeveyn;
4         if (g == null)                  // zig
5             dondur(p, x);
6         else if (ayniYon(x, p, g)) {  // zig-zig
7             dondur(g, p);   // önce dede
8             dondur(p, x);
9         } else {                        // zig-zag
10            dondur(p, x);   // önce ebeveyn
11            dondur(g, x);
12        }
13    }
14 }
```

DEĞİŞKENLER

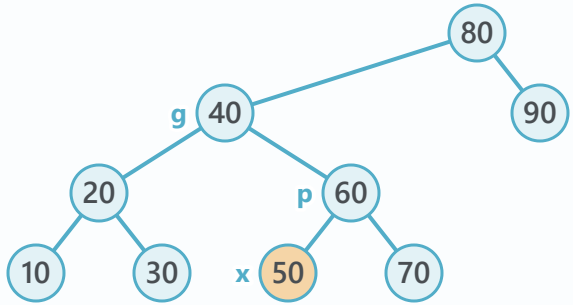
`x = düğüm(50)`



`splay(50)` : önce normal BST araması 80 → 40 → 60 → 50; bulunan düğüm x olur. (ayniYon: x ve p aynı yönde çocuk mu; dondur: kırmızı-siyah kodundakiyle aynı tek döndürme.)

Kod İzleme: `splay(50)`, sonra `splay(10)`

```
1  void splay(Dugum x) {
▶      while (x != kok) {
▶          Dugum p = x.ebeveyn, g = p.ebeveyn;
4          if (g == null)                  // zig
5              dondur(p, x);
6          else if (ayniYon(x, p, g)) {  // zig-zig
7              dondur(g, p);   // önce dede
8              dondur(p, x);
9          } else {                        // zig-zag
10             dondur(p, x);   // önce ebeveyn
11             dondur(g, x);
12         }
13     }
14 }
```



x kök değil. p = 60, g = 40.

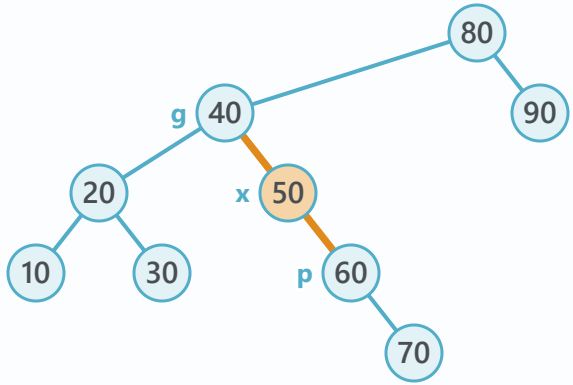
Adım 2 / 14

DEĞİŞKENLER

`x = düğüm(50)` `p = düğüm(60)` `g = düğüm(40)`

Kod İzleme: `splay(50)`, sonra `splay(10)`

```
1  void splay(Dugum x) {
2      while (x != kok) {
3          Dugum p = x.ebeveyn, g = p.ebeveyn;
4          if (g == null)                  // zig
5              dondur(p, x);
▶          else if (ayniYon(x, p, g)) {  // zig-zig
7              dondur(g, p);   // önce dede
8              dondur(p, x);
▶          } else {                        // zig-zag
▶              dondur(p, x);   // önce ebeveyn
11             dondur(g, x);
12         }
13     }
14 }
```



x ve p zıt yönde → **zig-zag**. Önce **ebeveyn**: 60'ta sağa döndür.

Adım 3 / 14

DEĞİŞKENLER

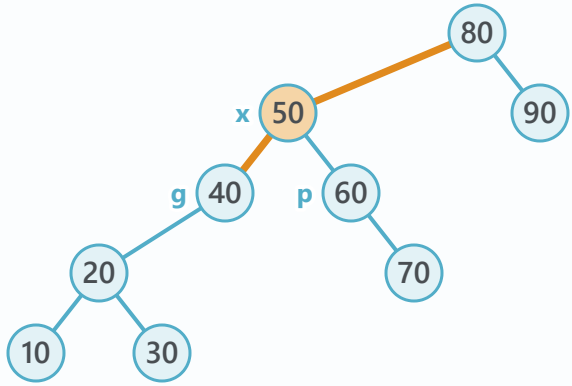
`x = düğüm(50)` `p = düğüm(60)` `g = düğüm(40)`

Kod İzleme: `splay(50)`, sonra `splay(10)`

```
1  void splay(Dugum x) {
2      while (x != kok) {
3          Dugum p = x.ebeveyn, g = p.ebeveyn;
4          if (g == null)                  // zig
5              dondur(p, x);
6          else if (ayniYon(x, p, g)) {  // zig-zig
7              dondur(g, p);   // önce dede
8              dondur(p, x);
9          } else {                        // zig-zag
10             dondur(p, x);   // önce ebeveyn
▶              dondur(g, x);
12         }
13     }
14 }
```

DEĞİŞKENLER

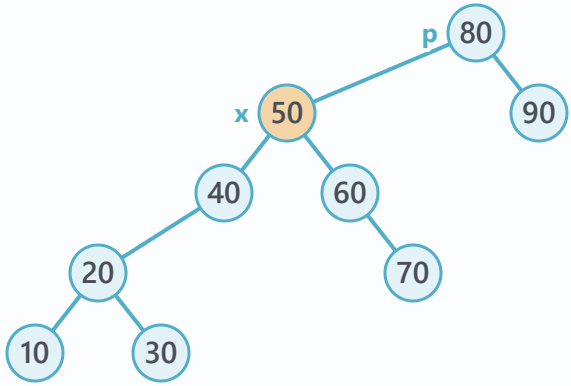
`x = düğüm(50)` `p = düğüm(60)` `g = düğüm(40)`



Sonra dede: 40'ta sola döndür → x, p ile g'nin ebeveyni olur.

Kod İzleme: `splay(50)`, sonra `splay(10)`

```
1  void splay(Dugum x) {
▶      while (x != kok) {
▶          Dugum p = x.ebeveyn, g = p.ebeveyn;
4          if (g == null)                  // zig
5              dondur(p, x);
6          else if (ayniYon(x, p, g)) {  // zig-zig
7              dondur(g, p);   // önce dede
8              dondur(p, x);
9          } else {                        // zig-zag
10             dondur(p, x);   // önce ebeveyn
11             dondur(g, x);
12         }
13     }
14 }
```



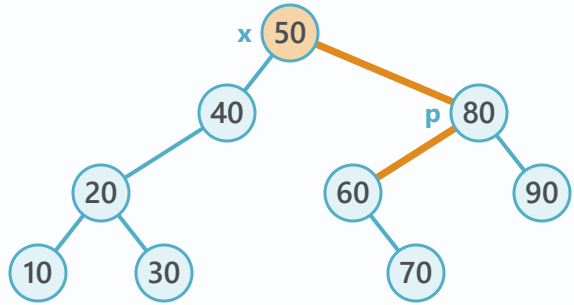
x kök değil. p = 80, g = null (p kök).

DEĞİŞKENLER

`x = düğüm(50)` `p = düğüm(80)` `g = null`

Kod İzleme: `splay(50)`, sonra `splay(10)`

```
1  void splay(Dugum x) {
2      while (x != kok) {
3          Dugum p = x.ebeveyn, g = p.ebeveyn;
▶          if (g == null)                  // zig
▶              dondur(p, x);
6          else if (ayniYon(x, p, g)) {  // zig-zig
7              dondur(g, p);   // önce dede
8              dondur(p, x);
9          } else {                        // zig-zag
10             dondur(p, x);   // önce ebeveyn
11             dondur(g, x);
12         }
13     }
14 }
```



g == null → **zig**: kök 80'de sağa döndürme; x kök olur.

DEĞİŞKENLER

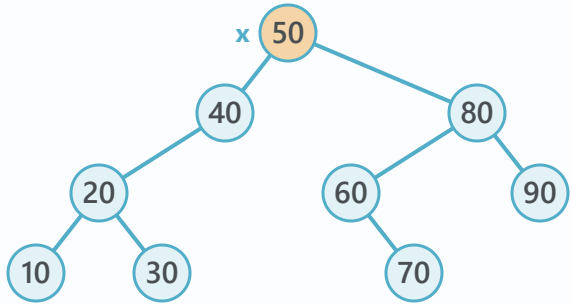
`x = düğüm(50)` `p = düğüm(80)` `g = null`

Kod İzleme: `splay(50)`, sonra `splay(10)`

```
1  void splay(Dugum x) {
▶      while (x != kok) {
3          Dugum p = x.ebeveyn, g = p.ebeveyn;
4          if (g == null)                  // zig
5              dondur(p, x);
6          else if (ayniYon(x, p, g)) {  // zig-zig
7              dondur(g, p);   // önce dede
8              dondur(p, x);
9          } else {                        // zig-zag
10             dondur(p, x);   // önce ebeveyn
11             dondur(g, x);
12         }
13     }
14 }
```

DEĞİŞKENLER

`x = düğüm(50)`



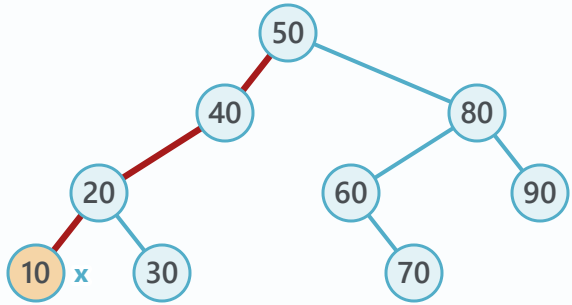
x == kok → döngü biter. 50 artık kök; sık erişilen anahtarlar köke yakın durur.

Kod İzleme: `splay(50)`, sonra `splay(10)`

```
▶ void splay(Dugum x) {
2     while (x != kok) {
3         Dugum p = x.ebeveyn, g = p.ebeveyn;
4         if (g == null)                  // zig
5             dondur(p, x);
6         else if (ayniYon(x, p, g)) {  // zig-zig
7             dondur(g, p);   // önce dede
8             dondur(p, x);
9         } else {                        // zig-zag
10            dondur(p, x);   // önce ebeveyn
11            dondur(g, x);
12        }
13    }
14 }
```

DEĞİŞKENLER

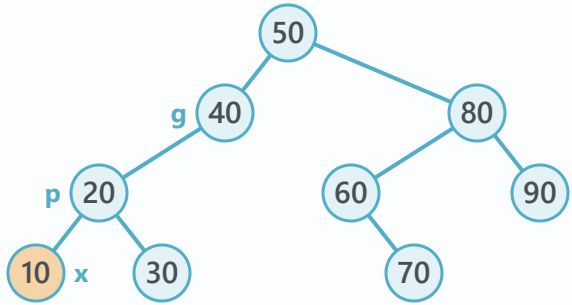
`x = düğüm(10)`



`splay(10)` : önce normal BST araması 50 → 40 → 20 → 10; bulunan düğüm x olur.

Kod İzleme: `splay(50)`, sonra `splay(10)`

```
1  void splay(Dugum x) {
▶      while (x != kok) {
▶          Dugum p = x.ebeveyn, g = p.ebeveyn;
4          if (g == null)                  // zig
5              dondur(p, x);
6          else if (ayniYon(x, p, g)) {  // zig-zig
7              dondur(g, p);   // önce dede
8              dondur(p, x);
9          } else {                        // zig-zag
10             dondur(p, x);   // önce ebeveyn
11             dondur(g, x);
12         }
13     }
14 }
```



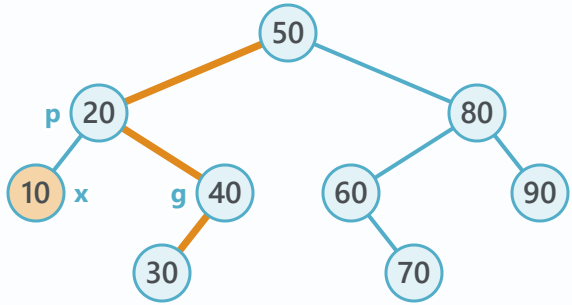
x kök değil. p = 20, g = 40.

DEĞİŞKENLER

`x = düğüm(10)` `p = düğüm(20)` `g = düğüm(40)`

Kod İzleme: `splay(50)`, sonra `splay(10)`

```
1  void splay(Dugum x) {
2      while (x != kok) {
3          Dugum p = x.ebeveyn, g = p.ebeveyn;
4          if (g == null)                  // zig
5              dondur(p, x);
▶          else if (ayniYon(x, p, g)) {  // zig-zig
▶              dondur(g, p);   // önce dede
8              dondur(p, x);
9          } else {                        // zig-zag
10             dondur(p, x);   // önce ebeveyn
11             dondur(g, x);
12         }
13     }
14 }
```



x ve p aynı yönde → **zig-zig**. Önce **dede**: 40'ta sağa döndür (p yukarı).

Adım 10 / 14

DEĞİŞKENLER

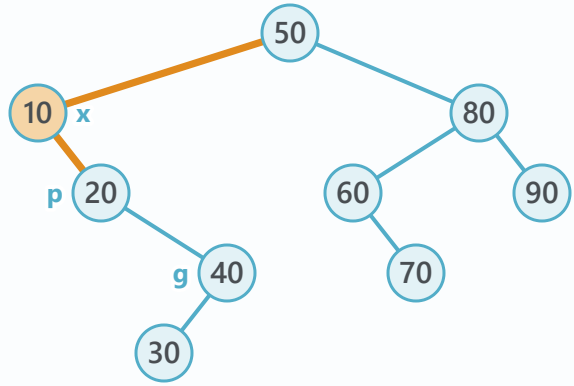
`x = düğüm(10)` `p = düğüm(20)` `g = düğüm(40)`

Kod İzleme: `splay(50)`, sonra `splay(10)`

```
1  void splay(Dugum x) {
2      while (x != kok) {
3          Dugum p = x.ebeveyn, g = p.ebeveyn;
4          if (g == null)                  // zig
5              dondur(p, x);
6          else if (ayniYon(x, p, g)) {  // zig-zig
7              dondur(g, p);   // önce dede
▶              dondur(p, x);
9          } else {                        // zig-zag
10             dondur(p, x);   // önce ebeveyn
11             dondur(g, x);
12         }
13     }
14 }
```

DEĞİŞKENLER

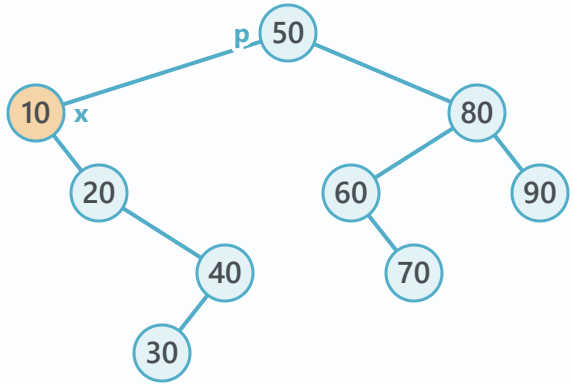
`x = düğüm(10)` `p = düğüm(20)` `g = düğüm(40)`



Sonra ebeveyn: 20'de sağa döndür → x iki düzey yükseldi.

Kod İzleme: `splay(50)`, sonra `splay(10)`

```
1  void splay(Dugum x) {
▶      while (x != kok) {
▶          Dugum p = x.ebeveyn, g = p.ebeveyn;
4          if (g == null)                  // zig
5              dondur(p, x);
6          else if (ayniYon(x, p, g)) {  // zig-zig
7              dondur(g, p);   // önce dede
8              dondur(p, x);
9          } else {                        // zig-zag
10             dondur(p, x);   // önce ebeveyn
11             dondur(g, x);
12         }
13     }
14 }
```



x kök değil. p = 50, g = null (p kök).

Adım 12 / 14

DEĞİŞKENLER

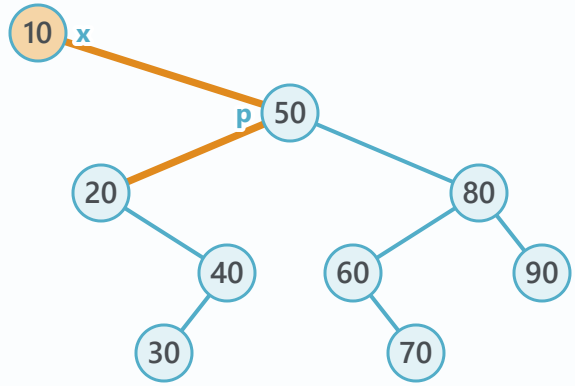
`x = düğüm(10)` `p = düğüm(50)` `g = null`

Kod İzleme: `splay(50)`, sonra `splay(10)`

```
1  void splay(Dugum x) {
2      while (x != kok) {
3          Dugum p = x.ebeveyn, g = p.ebeveyn;
▶          if (g == null)                  // zig
▶              dondur(p, x);
6          else if (ayniYon(x, p, g)) {  // zig-zig
7              dondur(g, p);   // önce dede
8              dondur(p, x);
9          } else {                        // zig-zag
10             dondur(p, x);   // önce ebeveyn
11             dondur(g, x);
12         }
13     }
14 }
```

DEĞİŞKENLER

`x = düğüm(10)` `p = düğüm(50)` `g = null`



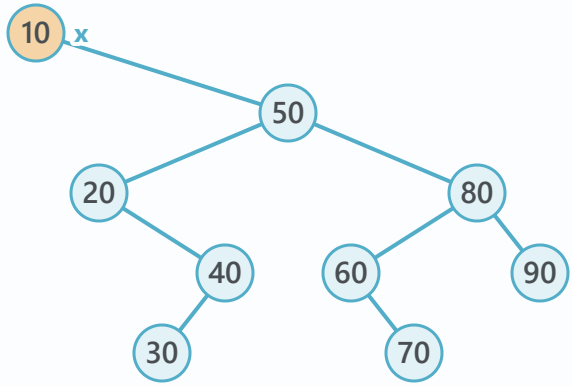
g == null → **zig**: kök 50'de sağa döndürme; x kök olur.

Kod İzleme: `splay(50)`, sonra `splay(10)`

```
1  void splay(Dugum x) {
▶      while (x != kok) {
3          Dugum p = x.ebeveyn, g = p.ebeveyn;
4          if (g == null)                  // zig
5              dondur(p, x);
6          else if (ayniYon(x, p, g)) {  // zig-zig
7              dondur(g, p);   // önce dede
8              dondur(p, x);
9          } else {                        // zig-zag
10             dondur(p, x);   // önce ebeveyn
11             dondur(g, x);
12         }
13     }
14 }
```

DEĞİŞKENLER

`x = düğüm(10)`

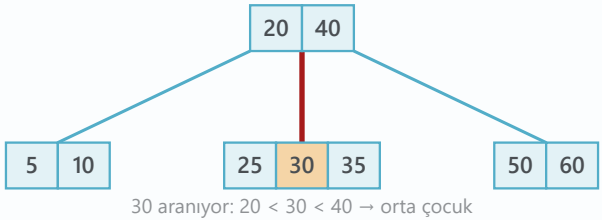


x == kok → döngü biter. 10 kök; az önce erişilen 50 hâlâ köke yakın (derinlik 1). Tek işlem O(n) olabilir, amortize O(log n).

Adım 14 / 14

B-Ağacı (B-Tree)

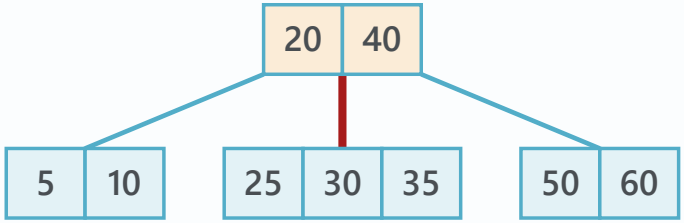
- Bir düğümde **birden çok anahtar** ve çok sayıda çocuk bulunan, dengeli bir çok yollu arama ağacıdır.
- Derecesi m olan B-ağacında bir düğüm en fazla m çocuk, m − 1 anahtar tutar; kök dışındaki **iç düğümlerde** en az ⌈m/2⌉ çocuk bulunur.
- Düğüm içindeki anahtarlar sıralıdır; **tüm yapraklar aynı derinliktedir**.
- Yükseklik O(log n) kalır; ekleme/silmede düğümler bölünür veya birleştirilir.



Neden? Bir düğüm bir disk bloğuna denk getirilir. Geniş ve alçak ağaç = az disk erişimi. Veritabanı ve dosya sistemi indislerinin temelidir.

Derecesi 4 olan B-ağacı (2-3-4 ağacı) kırmızı-siyah ağacın eşdeğeridir: her siyah düğüm kırmızı çocuklarıyla tek düğümde toplanınca 2-3-4 ağacı elde edilir. Adım adım ekleme ve bölünme: B-Ağacı simülatörü.

B-Ağacında Arama ve Ekleme



kök: 20 < 30 < 40 → orta çocuğa in

1 · Düğüm içinde ara. 30 aranıyor: kökteki anahtarlar sıralı karşılaştırılır; 20 < 30 < 40 → bu aralığa karşılık gelen **orta çocuğa** inilir.

Adım 1 / 4

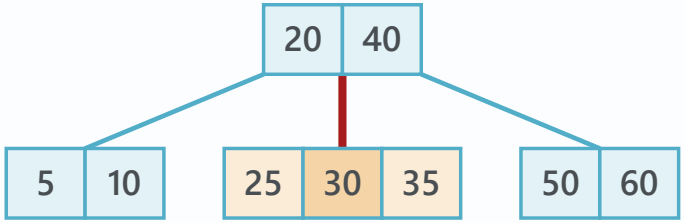
Kök bölünürse

Yeni bir kök oluşur ve ağacın yüksekliği yalnızca bir artar. Tüm yapraklar yine aynı derinlikte kalır.

Silme

Eksik dolan düğüm kardeşten anahtar ödünç alır veya kardeşle birleşir; denge ve doluluk sınırları korunur.

B-Ağacında Arama ve Ekleme



yaprak: 25, 30 → bulundu (2 düğüm okundu)

Yaprakta 30 bulunur. Her düzeyde yalnızca **bir düğüm (bir disk bloğu)** okunur → maliyet yükseklikle orantılı: $O(\log_m n)$.

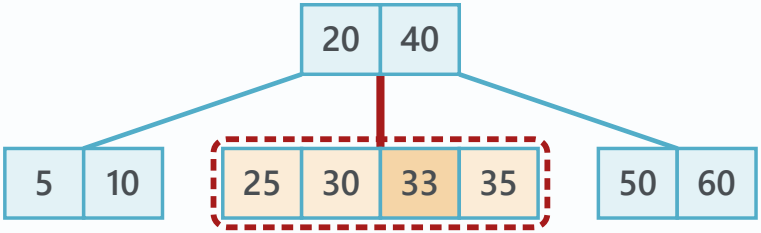
Kök bölünürse

Yeni bir kök oluşur ve ağacın yüksekliği yalnızca bir artar. Tüm yapraklar yine aynı derinlikte kalır.

Silme

Eksik dolan düğüm kardeşten anahtar ödünç alır veya kardeşle birleşir; denge ve doluluk sınırları korunur.

B-Ağacında Arama ve Ekleme



m = 4: bir düğümde en çok 3 anahtar

2 · Yaprağa ekle. 33 için aynı yol izlenir ve 33 yaprakta **sıralı yerine** konur. Artık 4 anahtar var: kapasite (m − 1 = 3) aşıldı.

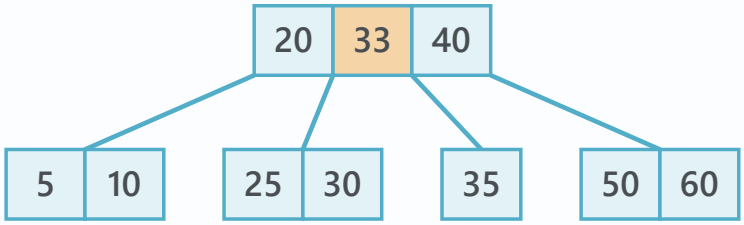
Kök bölünürse

Yeni bir kök oluşur ve ağacın yüksekliği yalnızca bir artar. Tüm yapraklar yine aynı derinlikte kalır.

Silme

Eksik dolan düğüm kardeşten anahtar ödünç alır veya kardeşle birleşir; denge ve doluluk sınırları korunur.

B-Ağacında Arama ve Ekleme



25 30 | 33 ↑ | 35

3 · Taşmayı böl. Ortadaki anahtar (33) **ebeveyne çıkar**, kalanlar iki düğüme ayrılır (25 30 ve 35). Ebeveyn de taşarsa bölünme köke kadar yayılır; kök bölünürse yeni kök oluşur ve yükseklik bir artar.

Adım 4 / 4

Kök bölünürse

Yeni bir kök oluşur ve ağacın yüksekliği yalnızca bir artar. Tüm yapraklar yine aynı derinlikte kalır.

Silme

Eksik dolan düğüm kardeşten anahtar ödünç alır veya kardeşle birleşir; denge ve doluluk sınırları korunur.

B-Ağacına Ekleme ve Bölünme (m = 3)

Sırayla eklenen anahtarlar: **10, 20, 30, 40, 50, 60, 70** · derece m = 3 → düğüm başına en çok 2 anahtar, 3 çocuk.



10 eklenir. kök yaprağa sıralı yerine konur; düğümde 1 anahtar var (≤ 2), bölünme gerekmez.

B-Ağacına Ekleme ve Bölünme (m = 3)

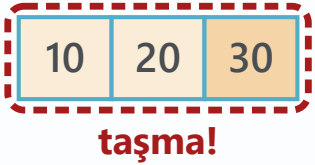
Sırayla eklenen anahtarlar: **10, 20, 30, 40, 50, 60, 70** · derece m = 3 → düğüm başına en çok 2 anahtar, 3 çocuk.



20 eklenir. kök yaprağa sıralı yerine konur; düğümde 2 anahtar var (≤ 2), bölünme gerekmez.

B-Ağacına Ekleme ve Bölünme (m = 3)

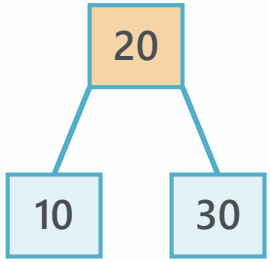
Sırayla eklenen anahtarlar: **10, 20, 30, 40, 50, 60, 70** · derece m = 3 → düğüm başına en çok 2 anahtar, 3 çocuk.



30 eklenir → düğüm [10, 20, 30] oldu: 3 anahtar, kapasite (m − 1 = 2) aşıldı → **bölünmeli**.

B-Ağacına Ekleme ve Bölünme (m = 3)

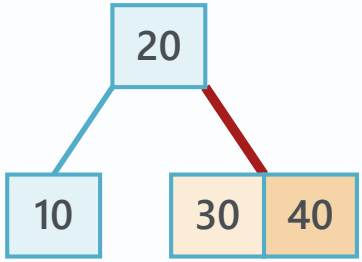
Sırayla eklenen anahtarlar: **10, 20, 30, 40, 50, 60, 70** · derece m = 3 → düğüm başına en çok 2 anahtar, 3 çocuk.



Kök bölündü: ortanca **20** yeni köke çıkar; 10 ve 30 iki ayrı düğüm olur. **Yükseklik bir arttı**, tüm yapraklar yine aynı derinlikte.

B-Ağacına Ekleme ve Bölünme (m = 3)

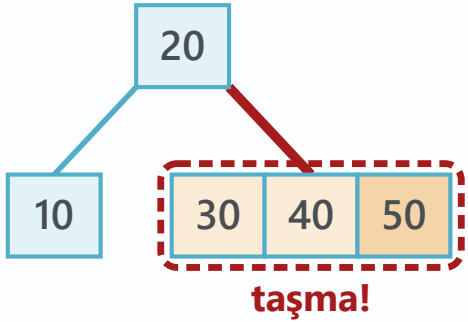
Sırayla eklenen anahtarlar: **10, 20, 30, 40, 50, 60, 70** · derece m = 3 → düğüm başına en çok 2 anahtar, 3 çocuk.



40 eklenir. Kökten sağa inilir; yaprağa sıralı yerine konur; düğümde 2 anahtar var (≤ 2), bölünme gerekmez.

B-Ağacına Ekleme ve Bölünme (m = 3)

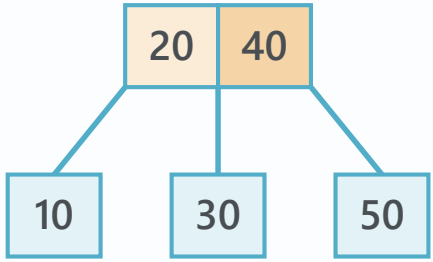
Sırayla eklenen anahtarlar: **10, 20, 30, 40, 50, 60, 70** · derece m = 3 → düğüm başına en çok 2 anahtar, 3 çocuk.



50 eklenir → düğüm [30, 40, 50] oldu: 3 anahtar, kapasite (m − 1 = 2) aşıldı → **bölünmeli**.

B-Ağacına Ekleme ve Bölünme (m = 3)

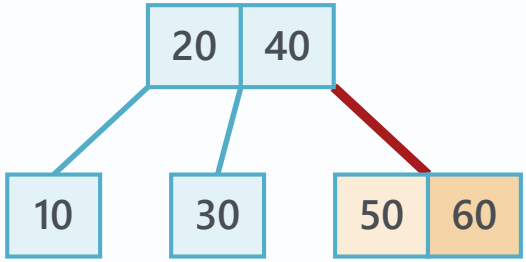
Sırayla eklenen anahtarlar: **10, 20, 30, 40, 50, 60, 70** · derece m = 3 → düğüm başına en çok 2 anahtar, 3 çocuk.



Ortanca **40** ebeveyne çıkar; kalanlar 30 ve 50 olarak iki düğüme ayrılır. Ebeveynde yer var → bölünme burada durur.

B-Ağacına Ekleme ve Bölünme (m = 3)

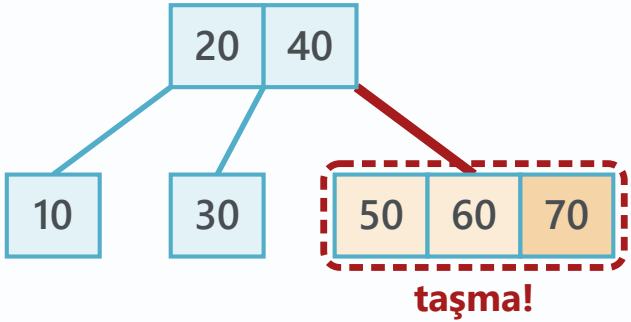
Sırayla eklenen anahtarlar: **10, 20, 30, 40, 50, 60, 70** · derece m = 3 → düğüm başına en çok 2 anahtar, 3 çocuk.



60 eklenir. Kökten sağa inilir; yaprağa sıralı yerine konur; düğümde 2 anahtar var (≤ 2), bölünme gerekmez.

B-Ağacına Ekleme ve Bölünme (m = 3)

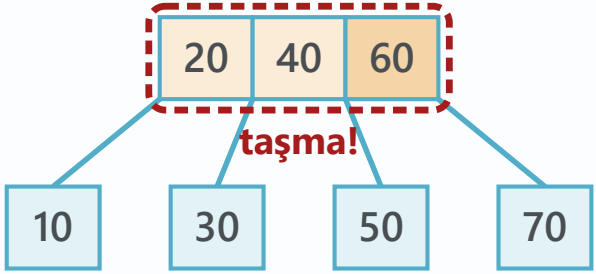
Sırayla eklenen anahtarlar: **10, 20, 30, 40, 50, 60, 70** · derece m = 3 → düğüm başına en çok 2 anahtar, 3 çocuk.



70 eklenir → düğüm [50, 60, 70] oldu: 3 anahtar, kapasite (m − 1 = 2) aşıldı → **bölünmeli**.

B-Ağacına Ekleme ve Bölünme (m = 3)

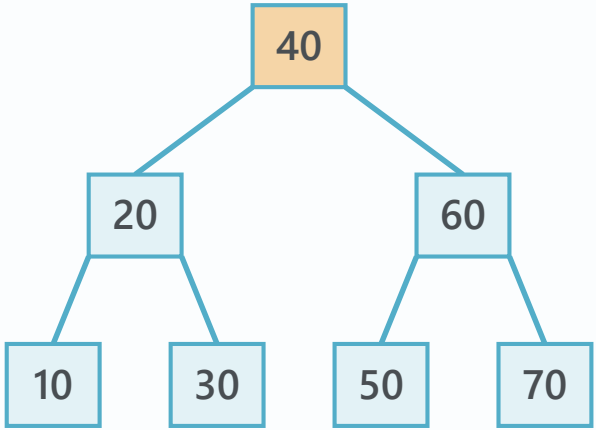
Sırayla eklenen anahtarlar: **10, 20, 30, 40, 50, 60, 70** · derece m = 3 → düğüm başına en çok 2 anahtar, 3 çocuk.



Ortanca **60** ebeveyne çıktı, düğüm 50 | 70 olarak ikiye ayrıldı. Ama ebeveyn [20, 40, 60] şimdi 3 anahtarlı → **bölünme yukarı yayılır**.

B-Ağacına Ekleme ve Bölünme (m = 3)

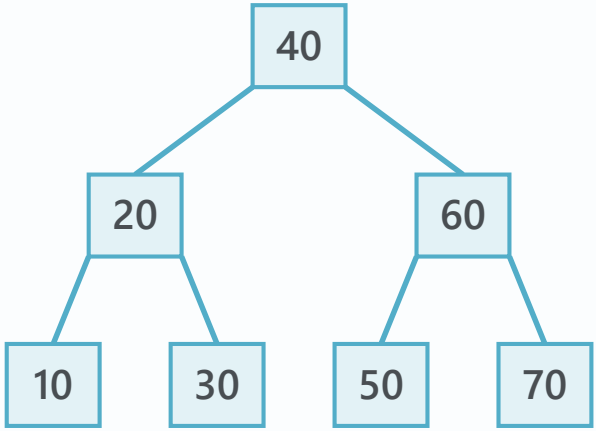
Sırayla eklenen anahtarlar: **10, 20, 30, 40, 50, 60, 70** · derece m = 3 → düğüm başına en çok 2 anahtar, 3 çocuk.



Kök bölündü: ortanca **40** yeni köke çıkar; 20 ve 60 iki ayrı düğüm olur. **Yükseklik bir arttı**, tüm yapraklar yine aynı derinlikte.

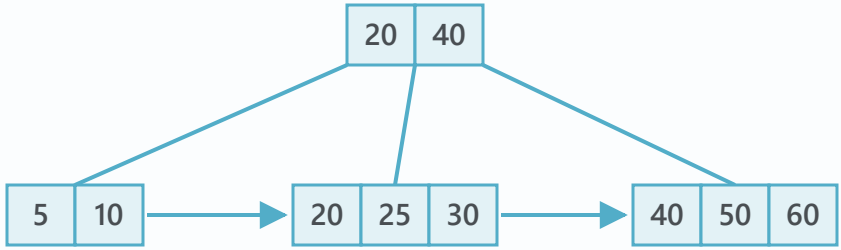
B-Ağacına Ekleme ve Bölünme (m = 3)

Sırayla eklenen anahtarlar: **10, 20, 30, 40, 50, 60, 70** · derece m = 3 → düğüm başına en çok 2 anahtar, 3 çocuk.



Sonuç: 7 anahtar, 3 düzey; her iç düğümün 2 çocuğu var, **tüm yapraklar aynı derinlikte**. B-ağacı aşağıdan yukarı büyür: yükseklik yalnızca kök bölününce artar.

B+ Ağacı (B+ Tree)



yapraklar bağlı liste: aralık sorgusu soldan sağa okunur

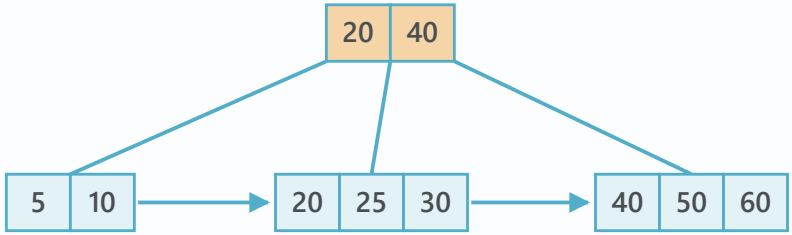
Veri yapraklarda
Tüm anahtarlar (ve kayıtlar) yapraklarda bulunur; iç düğümler yalnızca yönlendirme anahtarı tutar.

Bağlı yapraklar
Yapraklar birbirine bağlıdır; "20 ile 50 arası" gibi aralık sorguları hızlıdır.

Daha geniş iç düğümler
İç düğümler kayıt taşımadığından bir bloğa daha çok anahtar sığar → ağaç daha alçak.

Çoğu ilişkisel veritabanı indisi B+ ağacıdır.

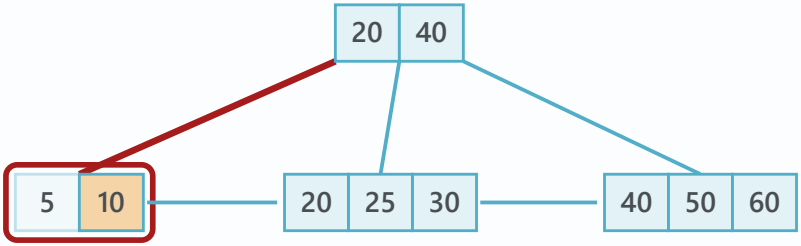
B+ Ağacında Aralık Sorgusu: 10 ile 45 arası



sorgu: 10 ≤ anahtar ≤ 45 **sonuç: —**

Kökten başla: 10 < 20 → alt sınır 10'u içeren yaprak **en soldaki** çocuktadır. (Arama gibi: O(log n) düzey.)

B+ Ağacında Aralık Sorgusu: 10 ile 45 arası

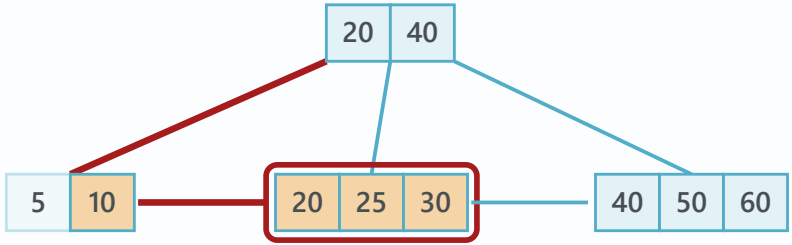


sorgu: 10 ≤ anahtar ≤ 45

sonuç: 10

İlk yaprak [5 10]: 5 < 10 atlanır, **10** alınır. Yaprağın sonuna gelindi → **sonraki yaprak** bağına geç (yukarı çıkmadan).

B+ Ağacında Aralık Sorgusu: 10 ile 45 arası

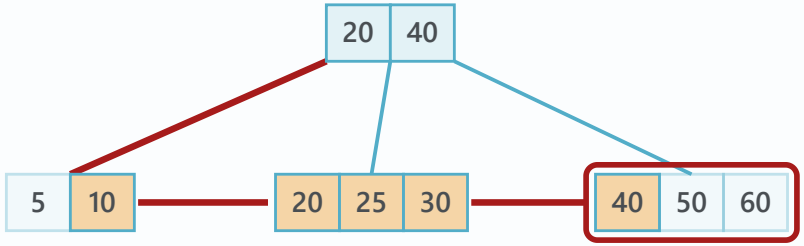


sorgu: 10 ≤ anahtar ≤ 45

sonuç: 10 20 25 30

İkinci yaprak: 20, 25, 30 hepsi aralıkta → alınır. Yine bağı izle.

B+ Ağacında Aralık Sorgusu: 10 ile 45 arası

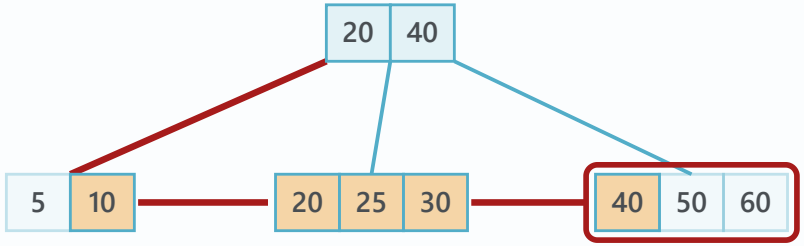


sorgu: 10 ≤ anahtar ≤ 45

sonuç: 10 20 25 30 40

Üçüncü yaprak: **40** alınır; 50 > 45 → **dur**. Sonuç: 10 20 25 30 40.

B+ Ağacında Aralık Sorgusu: 10 ile 45 arası



sorgu: 10 ≤ anahtar ≤ 45

sonuç: 10 20 25 30 40

Maliyet: bir kez aşağı inmek O(log n) + okunan yapraklar. B-ağacında aynı sorgu iç düğümlere tekrar tekrar inip çıkmayı gerektirirdi; B+ ağacında yaprak zinciri **O(log n + k)** sağlar (k: sonuç sayısı).

ÖZET

Özet: Arama Ağaçlarında İşlem Maliyetleri

Yapı	Yükseklik	Arama	Ekleme	Silme	Nasıl dengelenir?
BST	log n … n	O(h)	O(h)	O(h)	dengelenmez; en kötü O(n)
AVL	≤ 1,44 log n	O(log n)	O(log n)	O(log n)	bf ∈ {−1, 0, 1}; LL, RR, LR, RL döndürmeleri
Kırmızı-Siyah	≤ 2 log(n+1)	O(log n)	O(log n)	O(log n)	renk değiştirme + en çok 2–3 döndürme
Splay	sınırsız	O(log n)*	O(log n)*	O(log n)*	erişilen düğüm zig / zig-zig / zig-zag ile köke
B-ağacı / B+	≈ $\log_m n$	O(log n)	O(log n)	O(log n)	taşan düğüm bölünür; tüm yapraklar aynı derinlikte

* Amortize; tek işlem O(n) olabilir. Tüm yapılarda kök ortada gezinme O(n) ve anahtarları sıralı verir; min/max O(h).

BST özelliği

sol < kök < sağ, tüm alt ağaç için.

Silme

yaprak → kaldır; tek çocuk → atla; iki çocuk → ardıl/öncül.

Döndürme

O(1), kök ortada sırayı korur, yükseklikleri değiştirir.

ÖZET

Etkileşimli Simülatörler

İkili Arama Ağacı Simülatörü
Arama, ekleme, üç silme durumu ve gezinmeler.

İkili Arama Ağacı Gezinme Simülatörü
Ekleme, arama, silme; kök önde / ortada / sonda ve düzey sırası gezinmeleri.

AVL Ağacı Simülatörü
Denge faktörleri ve LL / RR / LR / RL döndürmeleri.

Kırmızı-Siyah Ağaç Simülatörü
Renk değiştirme ve döndürmelerle ekleme.

Splay Ağacı Simülatörü
Erişilen düğümün zig, zig-zig, zig-zag ile köke taşınması.

B-Ağacı Simülatörü
Anahtar ekledikçe düğümlerin bölünmesi ve ağacın dengeli büyümesi.

B+ Ağacı Simülatörü
Verinin yapraklarda tutulduğu, yaprakları bağlı çok yollu ağaç.

ÖZET

Sık Yapılan Hatalar

Dönüş değerini atamamak

`sil(kok, 40);` yazmak kök silindiğinde ağacı bozar. Doğrusu: `kok = sil(kok, 40);` (ekle için de aynı).

Yalnız çocuklara bakmak

BST özelliği tüm alt ağaç içindir; doğrulamada izinli **(min, max)** aralığı aşağı taşınmalı (`bstMi`).

Nesneleri `==` ile karşılaştırmak

Referansı karşılaştırır. Anahtarlar `compareTo` ile, eşitlik `equals` ile sınanır.

Yükseklik güncelleme sırası

AVL döndürmesinden sonra önce **alta inen** düğüm, sonra yeni kök güncellenir; tersi yanlış bf verir.

"BST her zaman O(log n)"

Yanlış: sıralı girdiyle h = n − 1 olur. O(log n) ancak dengeli (AVL, kırmızı-siyah) ağaçlarda garantidir.

null çocuğu unutmak

Boş alt ağacın yüksekliği −1'dir, kırmızı-siyahta NIL siyah sayılır; `h(null)` , `kirmizi(null)` gibi yardımcılar bu yüzden var.

ÖZET

Örnek Problemler

1. Boş BST'ye 50, 30, 70, 20, 40, 60, 80 sırayla eklenir. Yükseklik nedir? Kök ortada gezinme çıktısı nedir?
2. Aynı anahtarlar 20, 30, 40, 50, 60, 70, 80 sırasıyla eklenirse yükseklik ve arama maliyeti ne olur?
3. 1. sorudaki ağaçtan kök 50, kök ortada ardıl kullanılarak silinirse ağaç nasıl olur?
4. Boş AVL ağacına 3, 2, 1 eklenir. Hangi durum oluşur, hangi döndürme yapılır?
5. Boş AVL ağacına 10, 30, 20 eklenir. Hangi durum oluşur, sonuç ağaç nedir?
6. Kırmızı-siyah ağaçta kökten NIL'e giden en uzun yol, en kısa yolun en fazla kaç katıdır? Neden?
7. Splay ağacında zig-zig adımında döndürmeler hangi sırayla yapılır?

▶ **Cevapları göster**

ÖZET

Alıştırmalar

1. `taban(kok, x)` : ≤ x olan en büyük anahtarı (yoksa null) döndüren metodu yazın.
2. BST'de **k. en küçük** anahtarı bulan bir metot yazın. Zaman karmaşıklığı nedir?
3. 1, 2, …, 7 boş bir BST'ye hangi sırayla eklenirse yüksekliği 2 olan mükemmel ağaç oluşur?
4. Boş bir AVL ağacına 1, 2, …, 7 sırayla eklenirse son ağaç nasıl olur? Kaç döndürme yapılır?
5. Yüksekliği 5 olan bir AVL ağacı en az kaç düğüm içerir?
6. `aralik(kok, lo, hi)` neden O(n) değil de O(h + s) sürer?

▶ **Cevaplar / 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.
- Sedgewick ve Wayne, *Algorithms*, 4. baskı, 2011.

Son

Bölüm 10: Arama Ağaçları

Sonraki sunum: Bölüm 11: Öncelikli Kuyruklar ve Heap · Çalışma soruları · Simülatörler: İkili Arama Ağacı Simülatörü, İkili Arama Ağacı Gezinme Simülatörü, Kırmızı-Siyah Ağaç Simülatörü, AVL Ağacı Simülatörü, Splay Ağacı Simülatörü, B-Ağacı Simülatörü, B+ Ağacı Simülatörü · Terim sözlüğü