Bölüm 9: Ağaçlar

Veri Yapıları

Ağaç terimleri | İkili ağaçlar | Dolu · tam · mükemmel | Dizi ve bağlı gösterim | Gezinmeler | Özyineleme ve yığın | İfade ağacı

Sercan KÜLCÜ · Giresun Üniversitesi · Bilgisayar Mühendisliği

GİRİŞ

Öğrenme Çıktıları

Temel kavramlar

Kök, ebeveyn, çocuk, yaprak, derinlik ve yükseklik terimlerini açıklamak.

Gösterimler

Ağaçları düğüm bağlantıları ve dizilerle temsil etmek.

Gezinmeler

Kök önde, kök ortada, kök sonda ve düzey sırası gezinmeleri uygulamak; özyinelemeyi çağrı yığınıyla izlemek.

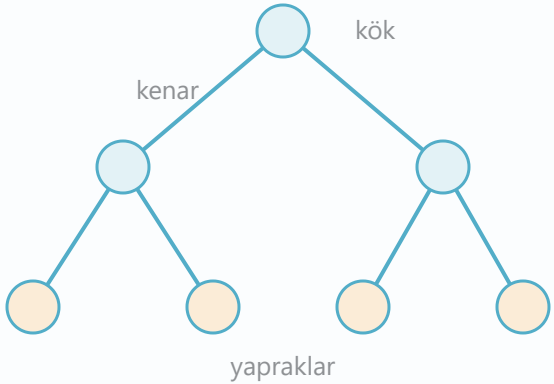
Uygulama ve maliyet

Genel ve ikili ağaçları kullanım alanları ve işlem maliyetleriyle karşılaştırmak.

AĞAÇ KAVRAMI

Ağaç (Tree)

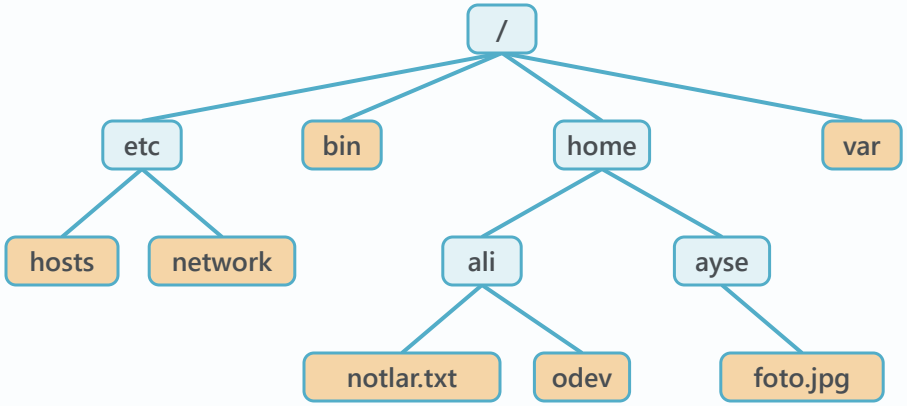
- **Doğrusal olmayan**, hiyerarşik bir veri yapısıdır.
- **Düğümler** (node) ve onları bağlayan **kenarlardan** (edge) oluşur.
- En üstte tek bir **kök** (root) bulunur; ağaç kökten aşağı doğru ters çizilir.
- Kök dışındaki her düğümün **tam olarak bir** ebeveyni vardır; bu yüzden **döngü yoktur**.
- Her düğümün sıfır veya daha fazla çocuğu olabilir.



Kural: n düğümlü bir ağaçta her zaman **n − 1 kenar** vardır (kök hariç her düğüm, ebeveynine tek bir kenarla bağlanır).

AĞAÇ KAVRAMI

Günlük Hayatta Ağaçlar: Hiyerarşi



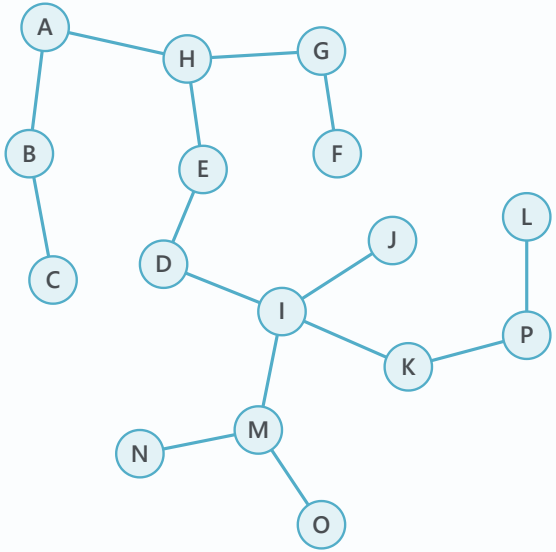
Dosya sistemi
Kök dizin, klasörler (iç düğüm), dosyalar (yaprak).

Kitap
Kitap → kısımlar → bölümler → alt başlıklar.

Coğrafya
Kıta → ülke → eyalet/il → şehir.

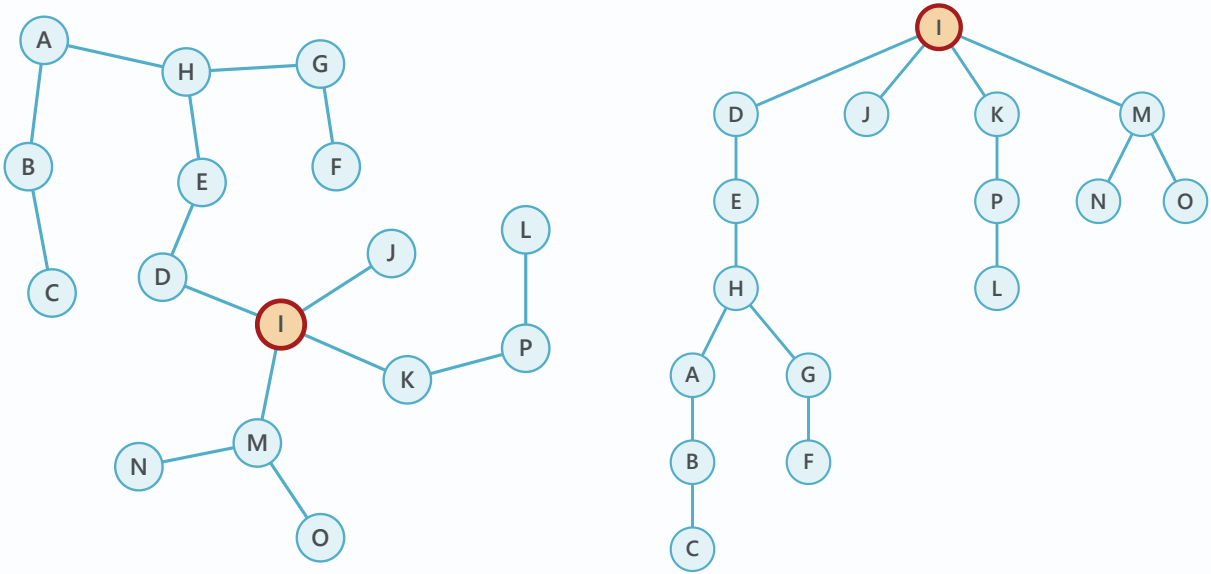
Web sayfası
HTML belgesi iç içe öğelerden oluşan bir ağaçtır (DOM).

Ağaç mı?



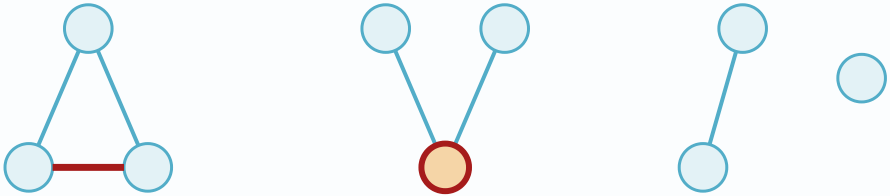
Bu yapı bir ağaç mıdır? 16 düğüm var.

Ağaç mı?



Evet. Bağlantılı, döngüsüz ve 16 düğüme 15 kenar. Herhangi bir düğüm (ör. I) kök seçilip ağaç olarak çizilebilir.

Ağaç mı?



döngü var | iki ebeveyn | bağlantısız

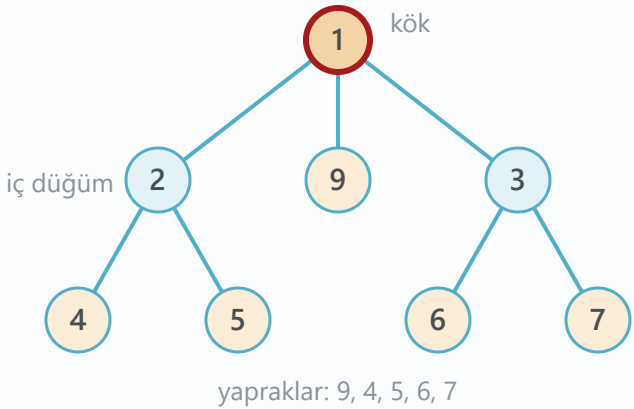
Ağaç **olmayanlar**: döngü içeren, bir düğümü iki ebeveynli olan veya parçaları bağlantısız yapılar.

Temel Unsurlar: Kök, İç Düğüm, Yaprak

Kök düğüm (root)
Ağacın en üst düğümü; ebeveyni yoktur.
`dosya sistemi: kök dizin /`

İç düğüm (internal node)
En az bir çocuğu olan düğüm.
`dosya sistemi: klasörler`

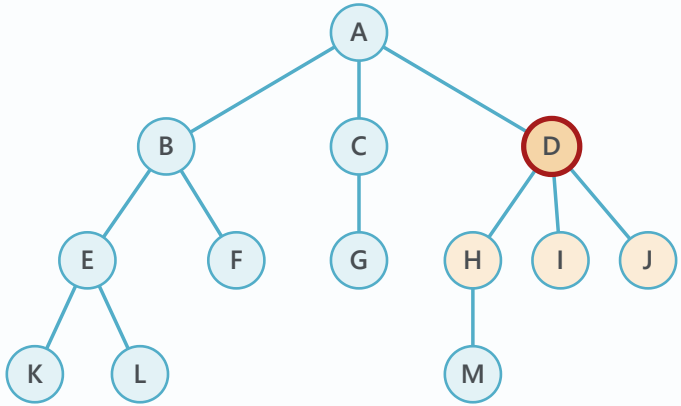
Yaprak (leaf)
Hiç çocuğu olmayan düğüm; dış düğüm de denir.
`dosya sistemi: dosyalar`



Kök de (çocuğu varsa) bir iç düğümdür. Tek düğümlü ağaçta kök aynı zamanda yapraktır.

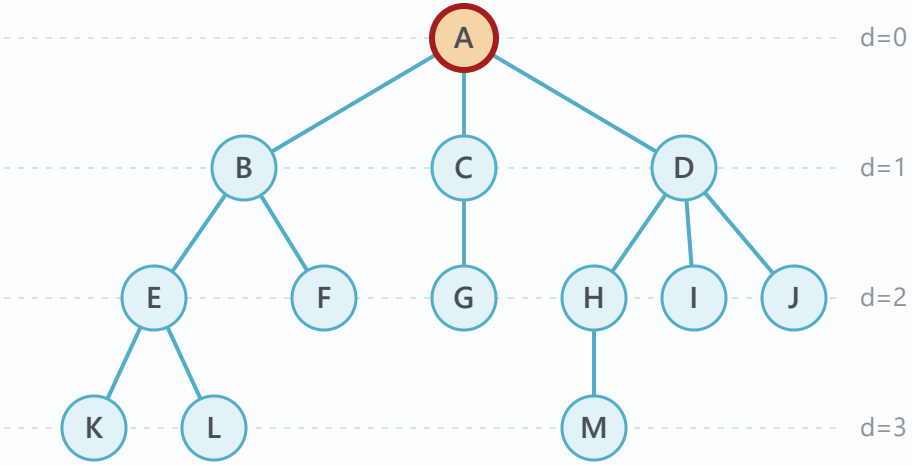
Ağaç İlişkileri

Terim	Anlamı (sağdaki ağaçta)
Ebeveyn (parent)	Hemen üstteki düğüm: D, H'nin ebeveyni
Çocuk (child)	Hemen alttaki düğüm: H, I, J → D'nin çocukları
Kardeş (sibling)	Ebeveyni aynı olanlar: H, I, J
Ata (ancestor)	Köke giden yoldaki düğümler: M'nin ataları H, D, A
Soy (descendant)	Alt ağaçtaki düğümler: D'nin soyu H, I, J, M



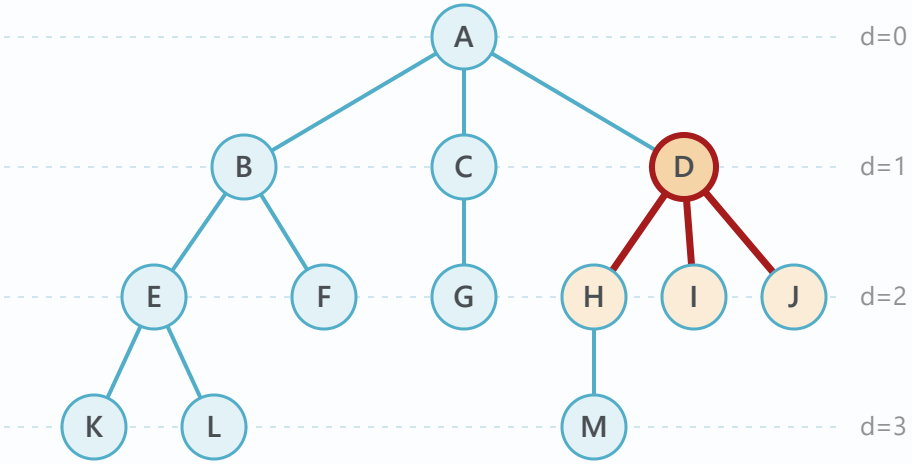
Dikkat: Aynı düzeyde olmak kardeş olmak demek değildir. F ve G aynı derinlikte ama ebeveynleri farklı (B ve C) → kardeş değiller.

Terimler Ağaç Üzerinde



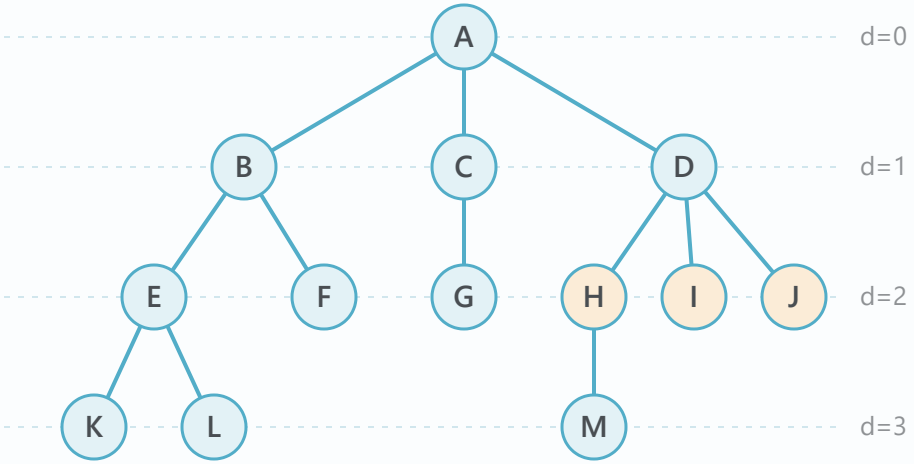
Kök: ebeveyni olmayan tek düğüm → A.

Terimler Ağaç Üzerinde



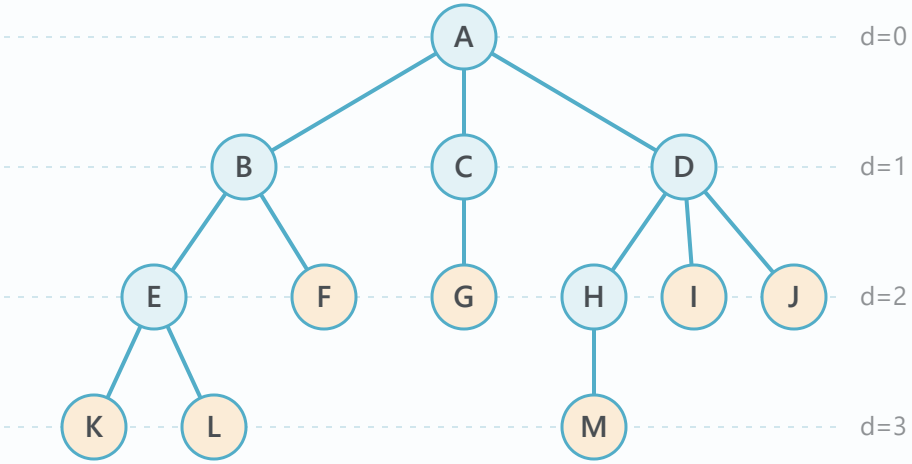
Ebeveyn / çocuk: D, H, I ve J'nin ebeveynidir; H, I, J de D'nin çocuklarıdır.

Terimler Ağaç Üzerinde



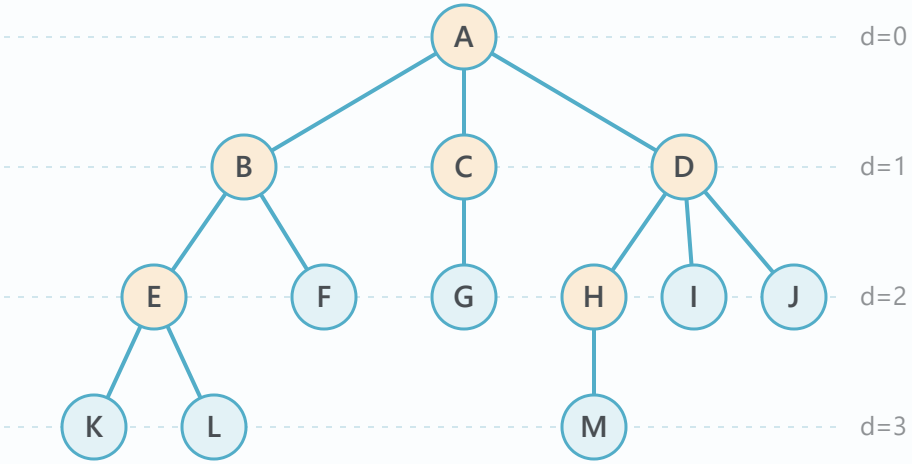
Kardeşler: H, I, J (ebeveynleri D). F ile G kardeş değildir.

Terimler Ağaç Üzerinde



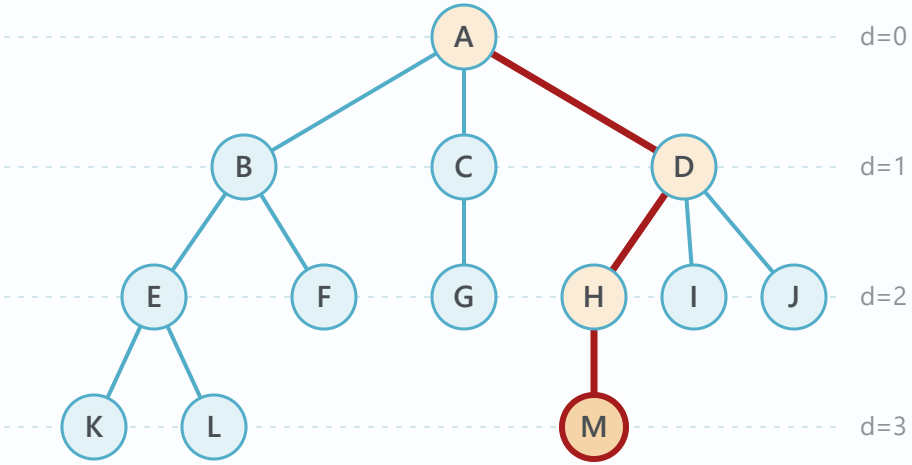
Yapraklar: K, L, F, G, M, I, J (çocuksuz düğümler).

Terimler Ağaç Üzerinde



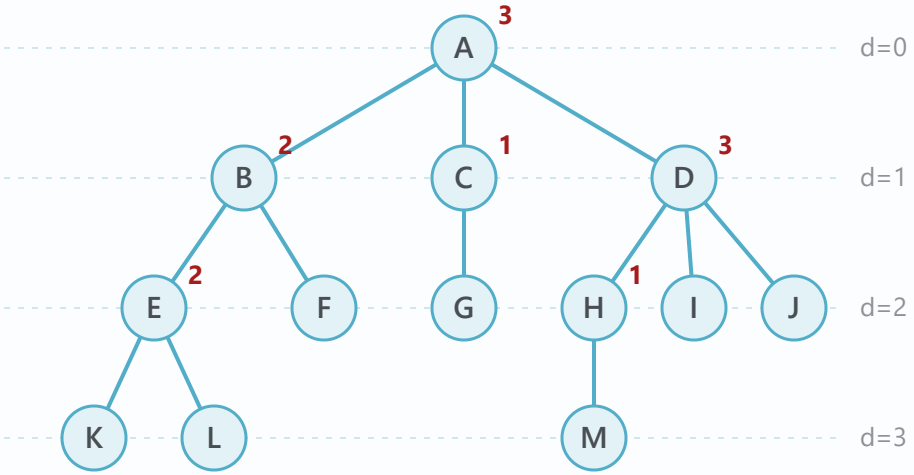
İç düğümler: A, B, C, D, E, H (en az bir çocuğu olanlar).

Terimler Ağaç Üzerinde



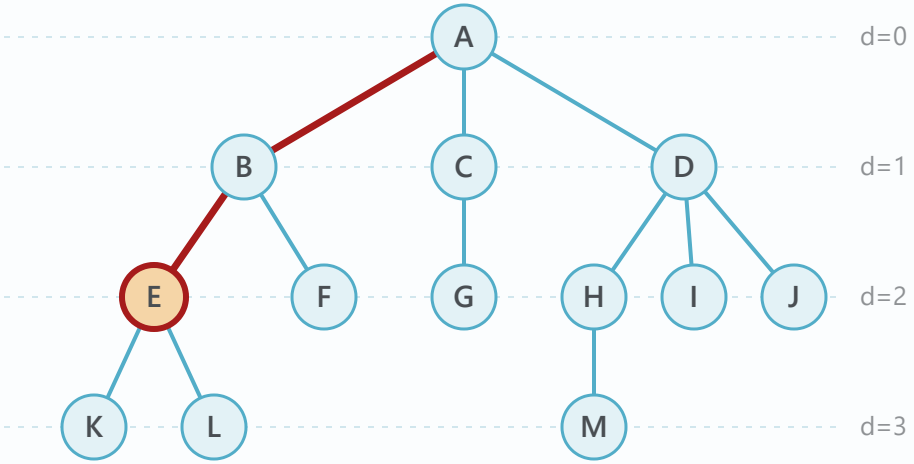
Atalar: M'nin ataları H, D, A. Tersine M; H, D ve A'nın **soyundandır**.

Terimler Ağaç Üzerinde



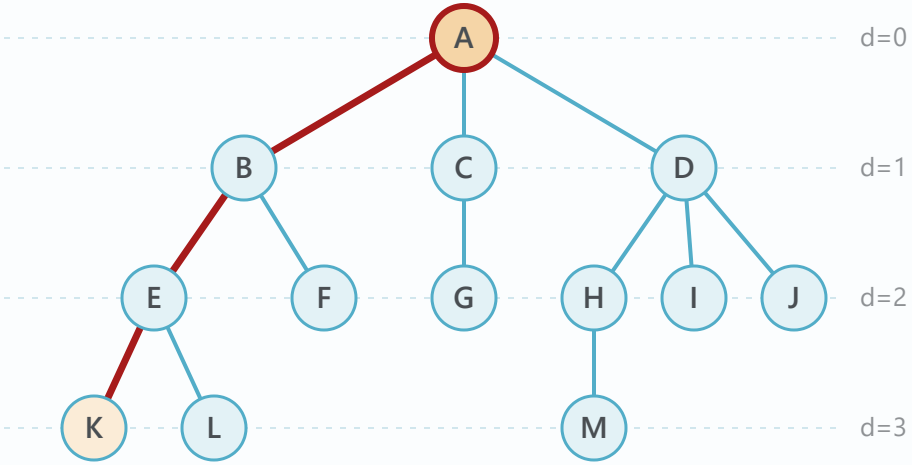
Derece (degree) = çocuk sayısı: A 3, B 2, C 1, D 3 … Ağacın derecesi = en büyük derece = 3.

Terimler Ağaç Üzerinde



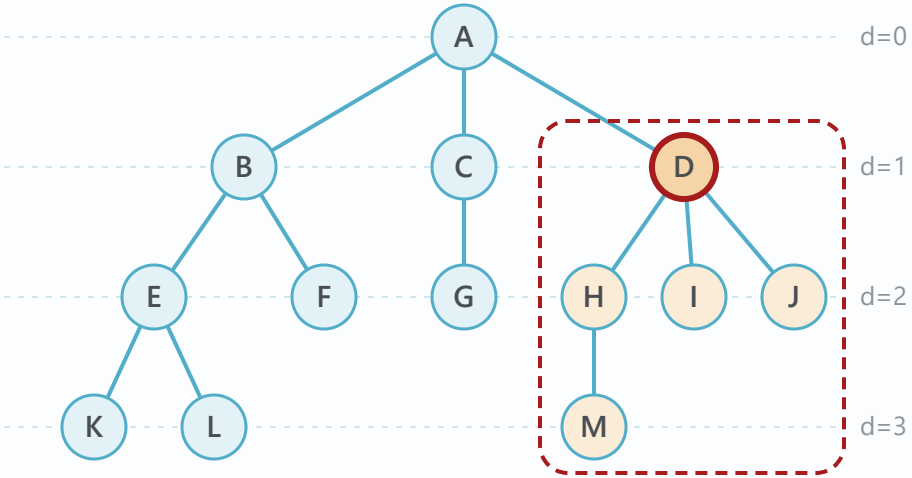
Derinlik (depth): kökten düğüme kenar sayısı. E'nin derinliği 2 (A→B→E).

Terimler Ağaç Üzerinde



Yükseklik (height): kökten en uzak yaprağa kenar sayısı = 3 (A→B→E→K).

Terimler Ağaç Üzerinde



Alt ağaç (subtree): D ve tüm soyu (H, I, J, M) kendi başına bir ağaçtır.

AĞAÇ KAVRAMI

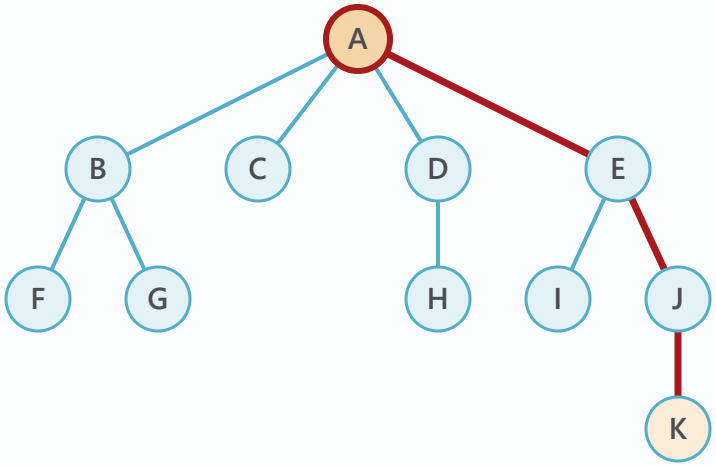
Derece, Derinlik, Yükseklik

Ölçü	Tanım	Önceki ağaçta
Derece (degree)	Bir düğümün çocuk sayısı (alt ağaç sayısı)	D → 3
Derinlik (depth)	Kökten o düğüme kadar olan yoldaki kenar sayısı; kökün derinliği 0	E → 2
Düzey (level)	Aynı derinlikteki düğümlerin kümesi	düzey 2: E … J
Düğüm yüksekliği	Düğümden altındaki en uzak yaprağa kenar sayısı; yaprağın yüksekliği 0	D → 2
Ağacın yüksekliği	Kökün yüksekliği = en büyük derinlik	3

Bu derste yükseklik ve derinlik **kenar sayısı** ile ölçülür. Bazı kaynaklar düğüm sayısını kullanır (düzeyleri 1'den başlatır); o zaman tüm değerler 1 fazla çıkar. Kaynaklar karşılaştırılırken hangi tanımın kullanıldığına bakılmalıdır.

AĞAÇ KAVRAMI

Derinlik ve Yükseklik



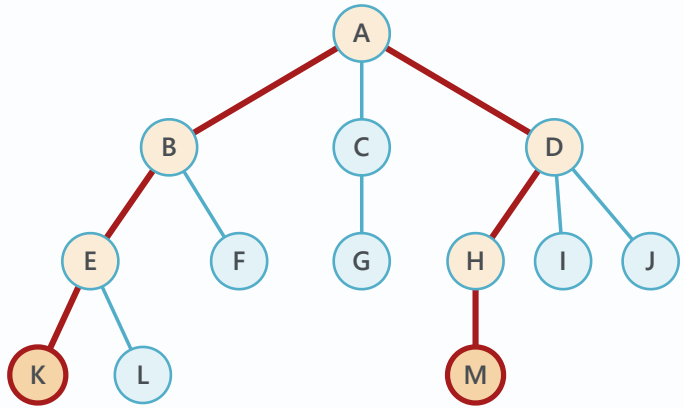
Düğüm	Yükseklik	Derinlik
A	3	0
B	1	1
C	0	1
D	1	1
E	2	1
F	0	2
G	0	2
H	0	2
I	0	2
J	1	2
K	0	3

Ağacın yüksekliği = kökün yüksekliği = **3** (en uzun yol A→E→J→K). En derin düğüm K'nin derinliği de 3'tür.

AĞAÇ KAVRAMI

Yol ve Alt Ağaç

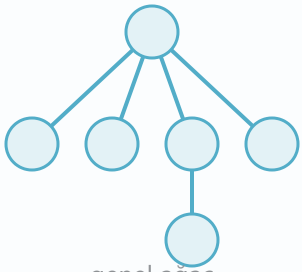
- **Yol** (path): ardışık düğümleri kenarlarla bağlayan dizi.
  $p = (v_0, e_1, v_1, \ldots, e_k, v_k)$
- $start(p) = v_0$, $end(p) = v_k$, $length(p) = k$ (kenar sayısı).
- Ağaçta iki düğüm arasında **tam olarak bir basit yol** vardır; ikinci bir yol olsaydı döngü oluşurdu.
- **Alt ağaç** (subtree): bir düğüm ve onun tüm soyu; kendisi de bir ağaçtır. Ör. bir klasör ve altındaki her şey.

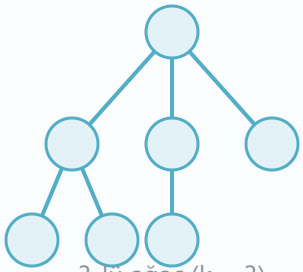


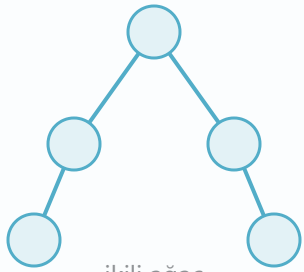
Vurgulu yol: K → E → B → A → D → H → M, uzunluğu **6**. Bu iki düğüm arasında başka yol yoktur.

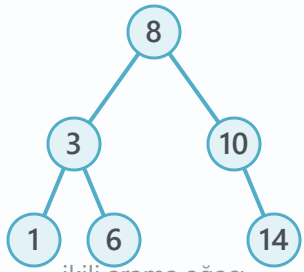
Temel Ağaç Türleri

Tür	Özellik
Genel ağaç	Bir düğümün çocuk sayısına sınır yoktur.
k-lı ağaç (k-ary)	Her düğümün en fazla k çocuğu vardır.
İkili ağaç (binary)	Her düğümün en fazla iki çocuğu vardır: sol ve sağ.
İkili arama ağacı (BST)	İkili ağaç + her düğümde: sol alt ağaçtaki anahtarlar küçük, sağdakiler büyük.


genel ağaç


3-lü ağaç (k = 3)


ikili ağaç

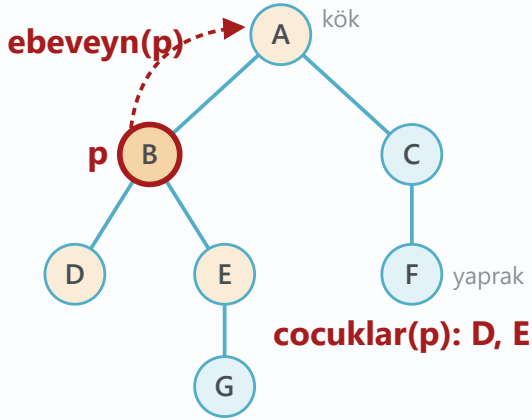


ikili arama ağacı

Ağaç Soyut Veri Tipi (Tree ADT)

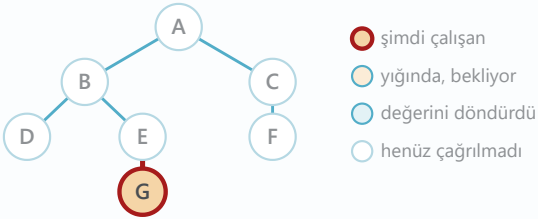
```java
public interface Agac<E> extends Iterable<E> {
  Konum<E> kok();
  Konum<E> ebeveyn(Konum<E> p);
  Iterable<Konum<E>> cocuklar(Konum<E> p);
  int cocukSayisi(Konum<E> p);
  boolean icMi(Konum<E> p);      // isInternal
  boolean yaprakMi(Konum<E> p);  // isExternal
  boolean kokMu(Konum<E> p);     // isRoot
  int size();
  boolean isEmpty();
  Iterable<Konum<E>> konumlar(); // positions
}
// Konum<E> (Position): E getElement();
```

İkili ağaç ADT'si bunlara `sol(p)`, `sag(p)`, `kardes(p)` ekler (Goodrich: `left`, `right`, `sibling`).



Konum (position), düğümün kendisini gizleyen bir "yer" soyutlamasıdır: kullanıcı yalnızca öğeye ve ADT metotlarına erişir.

Kod İzleme: ADT ile `derinlik(G)` ve `yukseklik(B)`



`derinlik(G)` : G kök değil → ebeveyni **E** için `derinlik(E)` çağrılır (yukarı çıkılır).

Adım 1 / 15

```
1  int derinlik(Konum<E> p) {
▶      if (kokMu(p)) return 0;
▶      return 1 + derinlik(ebeveyn(p));
4  }
5  int yukseklik(Konum<E> p) {
6      int h = 0;
7      for (Konum<E> c : cocuklar(p))
8          h = Math.max(h, 1 + yukseklik(c));
9      return h;
10 }
```

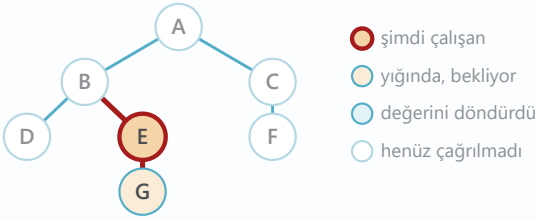
ÇAĞRI YIĞINI

- **`derinlik(G)`**
- `main()`

DEĞİŞKENLER

`p = G` `ebeveyn(p) = E`

Kod İzleme: ADT ile `derinlik(G)` ve `yukseklik(B)`



`derinlik(E)` : E kök değil → ebeveyni **B** için `derinlik(B)` çağrılır (yukarı çıkılır).

Adım 2 / 15

```
1  int derinlik(Konum<E> p) {
▶      if (kokMu(p)) return 0;
▶      return 1 + derinlik(ebeveyn(p));
4  }
5  int yukseklik(Konum<E> p) {
6      int h = 0;
7      for (Konum<E> c : cocuklar(p))
8          h = Math.max(h, 1 + yukseklik(c));
9      return h;
10 }
```

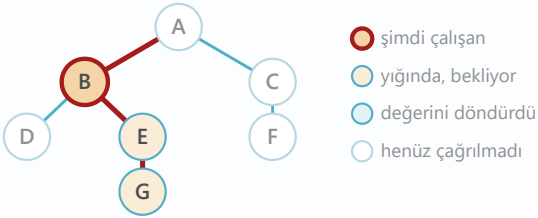
ÇAĞRI YIĞINI

- **`derinlik(E)`**
- `derinlik(G)`
- `main()`

DEĞİŞKENLER

`p = E` `ebeveyn(p) = B`

Kod İzleme: ADT ile `derinlik(G)` ve `yukseklik(B)`



`derinlik(B)` : B kök değil → ebeveyni **A** için `derinlik(A)` çağrılır (yukarı çıkılır).

Adım 3 / 15

```
1  int derinlik(Konum<E> p) {
▶      if (kokMu(p)) return 0;
▶      return 1 + derinlik(ebeveyn(p));
4  }
5  int yukseklik(Konum<E> p) {
6      int h = 0;
7      for (Konum<E> c : cocuklar(p))
8          h = Math.max(h, 1 + yukseklik(c));
9      return h;
10 }
```

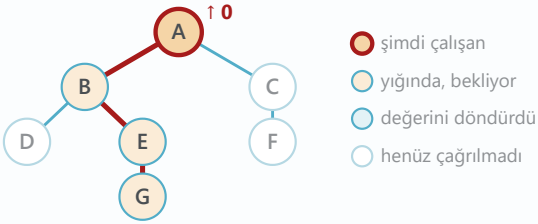
ÇAĞRI YIĞINI

- **`derinlik(B)`**
- `derinlik(E)`
- `derinlik(G)`
- `main()`

DEĞİŞKENLER

`p = B` `ebeveyn(p) = A`

Kod İzleme: ADT ile `derinlik(G)` ve `yukseklik(B)`



`derinlik(A)` : A kök → **return 0**. Taban durum; artık çerçeveler geri dönmeye başlar.

Adım 4 / 15

```
1  int derinlik(Konum<E> p) {
▶      if (kokMu(p)) return 0;
3      return 1 + derinlik(ebeveyn(p));
4  }
5  int yukseklik(Konum<E> p) {
6      int h = 0;
7      for (Konum<E> c : cocuklar(p))
8          h = Math.max(h, 1 + yukseklik(c));
9      return h;
10 }
```

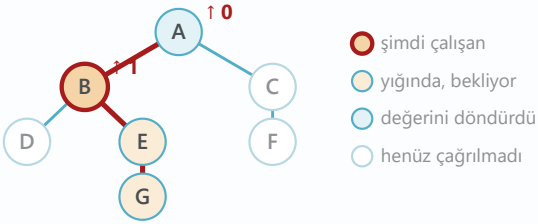
ÇAĞRI YIĞINI

- **`derinlik(A)`**
- `derinlik(B)`
- `derinlik(E)`
- `derinlik(G)`
- `main()`

DEĞİŞKENLER

`p = A` `dönüş = 0`

Kod İzleme: ADT ile `derinlik(G)` ve `yukseklik(B)`



`derinlik(B)` = 1 + 0 = **1** döner.

Adım 5 / 15

```
1  int derinlik(Konum<E> p) {
2      if (kokMu(p)) return 0;
▶      return 1 + derinlik(ebeveyn(p));
4  }
5  int yukseklik(Konum<E> p) {
6      int h = 0;
7      for (Konum<E> c : cocuklar(p))
8          h = Math.max(h, 1 + yukseklik(c));
9      return h;
10 }
```

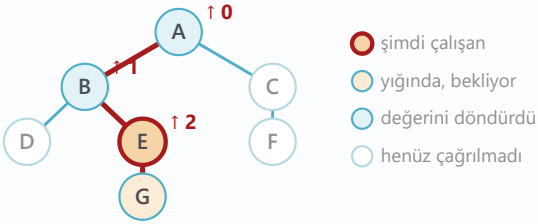
ÇAĞRI YIĞINI

- **`derinlik(B)`**
- `derinlik(E)`
- `derinlik(G)`
- `main()`

DEĞİŞKENLER

`p = B` `dönüş = 1`

Kod İzleme: ADT ile `derinlik(G)` ve `yukseklik(B)`



`derinlik(E)` = 1 + 1 = **2** döner.

Adım 6 / 15

```
1  int derinlik(Konum<E> p) {
2      if (kokMu(p)) return 0;
▶      return 1 + derinlik(ebeveyn(p));
4  }
5  int yukseklik(Konum<E> p) {
6      int h = 0;
7      for (Konum<E> c : cocuklar(p))
8          h = Math.max(h, 1 + yukseklik(c));
9      return h;
10 }
```

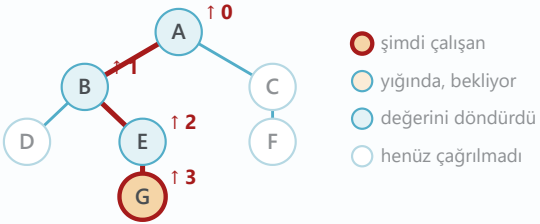
ÇAĞRI YIĞINI

- **`derinlik(E)`**
- `derinlik(G)`
- `main()`

DEĞİŞKENLER

`p = E` `dönüş = 2`

Kod İzleme: ADT ile `derinlik(G)` ve `yukseklik(B)`



`derinlik(G)` = 1 + 2 = **3** döner. **G'nin derinliği 3**: kökten G'ye 3 kenar (A–B–E–G). Maliyet O(d + 1).

Adım 7 / 15

```
1  int derinlik(Konum<E> p) {
2      if (kokMu(p)) return 0;
▶      return 1 + derinlik(ebeveyn(p));
4  }
5  int yukseklik(Konum<E> p) {
6      int h = 0;
7      for (Konum<E> c : cocuklar(p))
8          h = Math.max(h, 1 + yukseklik(c));
9      return h;
10 }
```

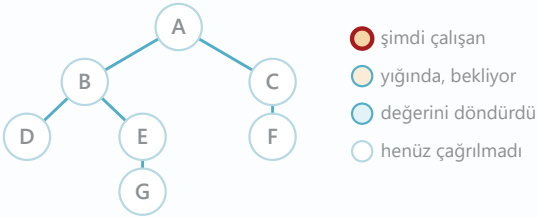
ÇAĞRI YIĞINI

- **`derinlik(G)`**
- `main()`

DEĞİŞKENLER

`p = G` `dönüş = 3`

Kod İzleme: ADT ile `derinlik(G)` ve `yukseklik(B)`



Şimdi `yukseklik(B)` : aşağı doğru, **çocuklar üzerinden** özyineleme (derinlik yukarı, yükseklik aşağı bakar).

Adım 8 / 15

```
1  int derinlik(Konum<E> p) {
2      if (kokMu(p)) return 0;
3      return 1 + derinlik(ebeveyn(p));
4  }
▶ int yukseklik(Konum<E> p) {
6      int h = 0;
7      for (Konum<E> c : cocuklar(p))
8          h = Math.max(h, 1 + yukseklik(c));
9      return h;
10 }
```

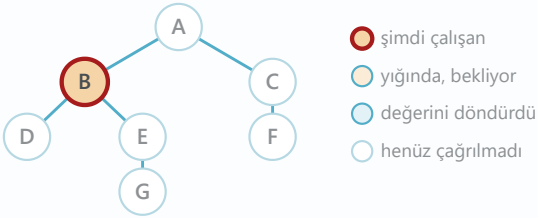
ÇAĞRI YIĞINI

`main()`

DEĞİŞKENLER

p = **B**

Kod İzleme: ADT ile `derinlik(G)` ve `yukseklik(B)`



`yukseklik(B)` : h = 0. Çocuklar: D, E. İlk çocuk **D** için özyineleme.

Adım 9 / 15

```
1  int derinlik(Konum<E> p) {
2      if (kokMu(p)) return 0;
3      return 1 + derinlik(ebeveyn(p));
4  }
5  int yukseklik(Konum<E> p) {
▶      int h = 0;
▶      for (Konum<E> c : cocuklar(p))
8          h = Math.max(h, 1 + yukseklik(c));
9      return h;
10 }
```

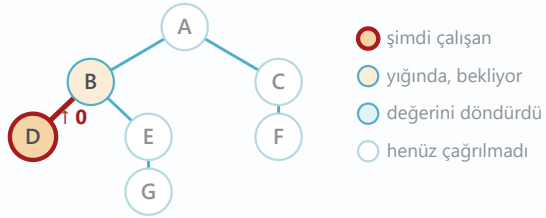
ÇAĞRI YIĞINI

- **yukseklik(B)**
- main()

DEĞİŞKENLER

`p = B` `h = 0` `c = D`

Kod İzleme: ADT ile `derinlik(G)` ve `yukseklik(B)`



`yukseklik(D)` : h = 0; D'nin çocuğu yok, döngü hiç dönmez → **return 0** (yaprak).

Adım 10 / 15

```
1  int derinlik(Konum<E> p) {
2      if (kokMu(p)) return 0;
3      return 1 + derinlik(ebeveyn(p));
4  }
5  int yukseklik(Konum<E> p) {
▶      int h = 0;
▶      for (Konum<E> c : cocuklar(p))
8          h = Math.max(h, 1 + yukseklik(c));
▶      return h;
10 }
```

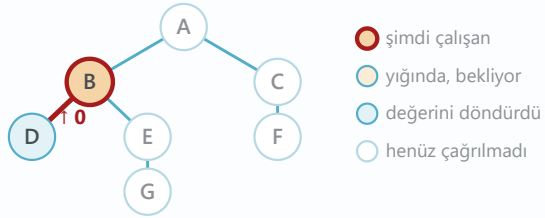
ÇAĞRI YIĞINI

- **`yukseklik(D)`**
- `yukseklik(B)`
- `main()`

DEĞİŞKENLER

`p = D` `h = 0`

Kod İzleme: ADT ile `derinlik(G)` ve `yukseklik(B)`



D'den 0 geldi: h = max(0, 1 + 0) = **1**. Sıradaki çocuk: **E**.

Adım 11 / 15

```
1  int derinlik(Konum<E> p) {
2      if (kokMu(p)) return 0;
3      return 1 + derinlik(ebeveyn(p));
4  }
5  int yukseklik(Konum<E> p) {
6      int h = 0;
▶      for (Konum<E> c : cocuklar(p))
▶          h = Math.max(h, 1 + yukseklik(c));
9      return h;
10 }
```

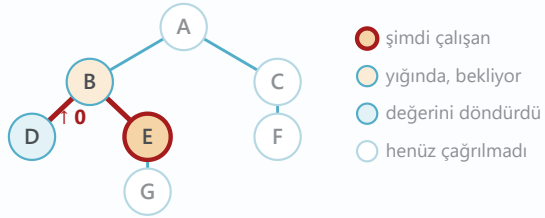
ÇAĞRI YIĞINI

- **yukseklik(B)**
- main()

DEĞİŞKENLER

p = **B** | h = **1** | c = **E**

Kod İzleme: ADT ile `derinlik(G)` ve `yukseklik(B)`



`yukseklik(E)` : h = 0. Çocuklar: G. İlk çocuk **G** için özyineleme.

Adım 12 / 15

```
1  int derinlik(Konum<E> p) {
2      if (kokMu(p)) return 0;
3      return 1 + derinlik(ebeveyn(p));
4  }
5  int yukseklik(Konum<E> p) {
▶      int h = 0;
▶      for (Konum<E> c : cocuklar(p))
8          h = Math.max(h, 1 + yukseklik(c));
9      return h;
10 }
```

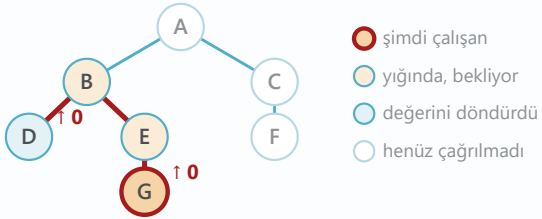
ÇAĞRI YIĞINI

- **`yukseklik(E)`**
- `yukseklik(B)`
- `main()`

DEĞİŞKENLER

`p = E` `h = 0` `c = G`

Kod İzleme: ADT ile `derinlik(G)` ve `yukseklik(B)`



`yukseklik(G)` : h = 0; G'nin çocuğu yok, döngü hiç dönmez → **return 0** (yaprak).

Adım 13 / 15

```
1  int derinlik(Konum<E> p) {
2      if (kokMu(p)) return 0;
3      return 1 + derinlik(ebeveyn(p));
4  }
5  int yukseklik(Konum<E> p) {
▶      int h = 0;
▶      for (Konum<E> c : cocuklar(p))
8          h = Math.max(h, 1 + yukseklik(c));
▶      return h;
10 }
```

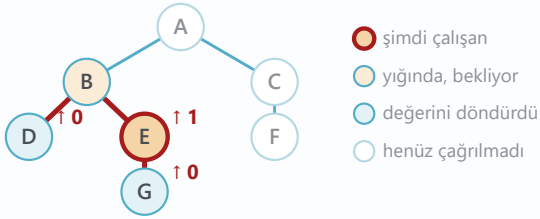
ÇAĞRI YIĞINI

- **`yukseklik(G)`**
- `yukseklik(E)`
- `yukseklik(B)`
- `main()`

DEĞİŞKENLER

`p = G` `h = 0`

Kod İzleme: ADT ile `derinlik(G)` ve `yukseklik(B)`



G'den 0 geldi: h = max(0, 1 + 0) = **1**. Çocuk kalmadı → **return 1**.

Adım 14 / 15

```
1  int derinlik(Konum<E> p) {
2      if (kokMu(p)) return 0;
3      return 1 + derinlik(ebeveyn(p));
4  }
5  int yukseklik(Konum<E> p) {
6      int h = 0;
7      for (Konum<E> c : cocuklar(p))
▶          h = Math.max(h, 1 + yukseklik(c));
▶      return h;
10 }
```

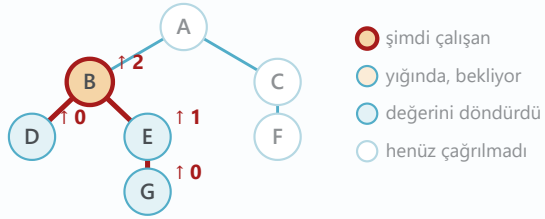
ÇAĞRI YIĞINI

- **`yukseklik(E)`**
- `yukseklik(B)`
- `main()`

DEĞİŞKENLER

`p = E` `h = 1` `dönüş = 1`

Kod İzleme: ADT ile `derinlik(G)` ve `yukseklik(B)`



E'den 1 geldi: h = max(1, 1 + 1) = **2**. Çocuk kalmadı → **return 2**. Yükseklik(B) = 2 (B–E–G); her düğüm bir kez ziyaret edilir.

Adım 15 / 15

```
1  int derinlik(Konum<E> p) {
2      if (kokMu(p)) return 0;
3      return 1 + derinlik(ebeveyn(p));
4  }
5  int yukseklik(Konum<E> p) {
6      int h = 0;
7      for (Konum<E> c : cocuklar(p))
▶          h = Math.max(h, 1 + yukseklik(c));
▶      return h;
10 }
```

ÇAĞRI YIĞINI

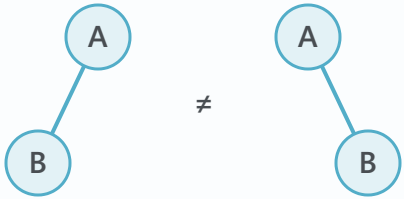
- **`yukseklik(B)`**
- `main()`

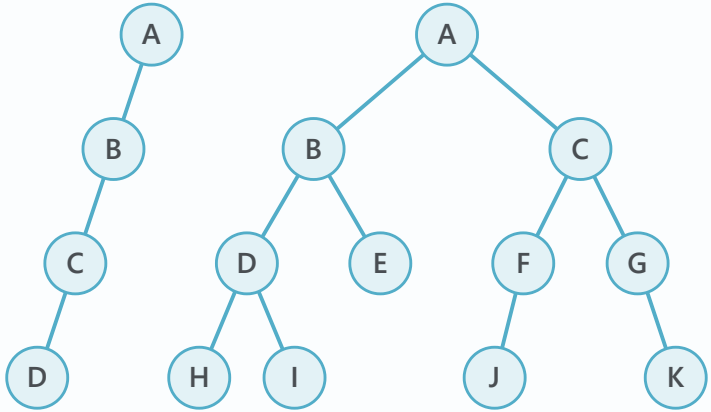
DEĞİŞKENLER

`p = B` `h = 2` `dönüş = 2`

İkili Ağaç (Binary Tree)

- Her düğümün **en fazla iki** çocuğu vardır: **sol çocuk** ve **sağ çocuk**.
- Çocukların yeri önemlidir: yalnızca sol çocuğu olan ağaç, yalnızca sağ çocuğu olandan farklıdır.
- **Özyinelemeli tanım:** İkili ağaç ya boştur ya da bir kök ile iki ikili ağaçtan (sol ve sağ alt ağaç) oluşur.





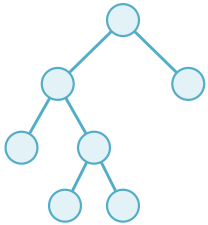
Soldaki zincir de geçerli bir ikili ağaçtır (çarpık / skewed ağaç): düğümler dengeli dağılmak zorunda değildir.

İKİLİ AĞAÇLAR

Dolu, Tam ve Mükemmel

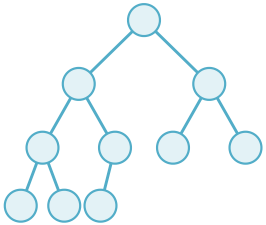
Dolu ikili ağaç (full)

Her düğümün ya **0** ya da **2** çocuğu vardır; tek çocuklu düğüm yoktur.



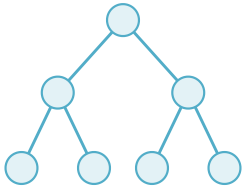
Tam ikili ağaç (complete)

Son düzey hariç tüm düzeyler doludur; son düzey **soldan sağa** boşluksuz dolar.



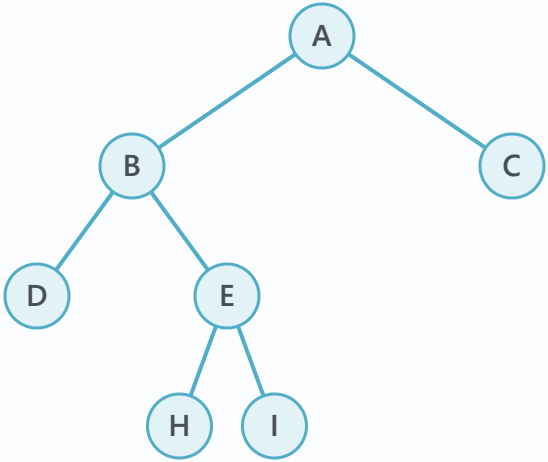
Mükemmel ikili ağaç (perfect)

Tüm iç düğümlerin 2 çocuğu var ve **tüm yapraklar aynı derinlikte**.



Mükemmel ağaç hem dolu hem tamdır; yüksekliği h ise $2^{h+1} - 1$ düğümü vardır. Türkçe kaynaklarda "tam" ve "dolu" bazen ters kullanılır; İngilizce karşılıklar (full / complete / perfect) esas alınmalıdır.

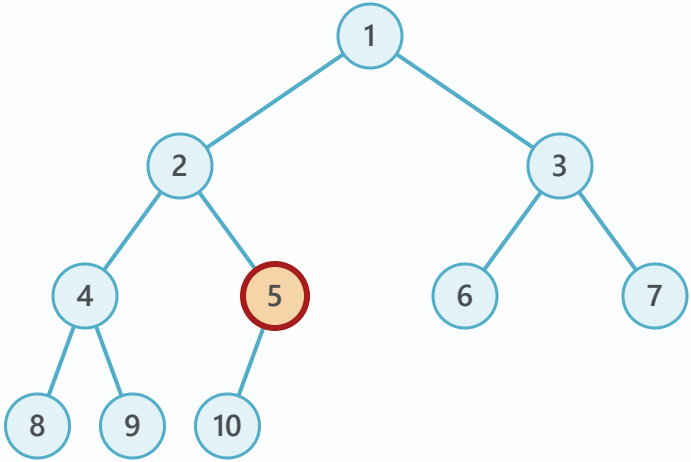
Hangisi? Dolu mu, Tam mı, Mükemmel mi?



Dolu (full)	**evet**
Tam (complete)	**hayır**
Mükemmel (perfect)	**hayır**

Her düğümün 0 ya da 2 çocuğu var → **dolu**. Ama C'nin altı boşken en alt düzey E'nin altında → tam değil.

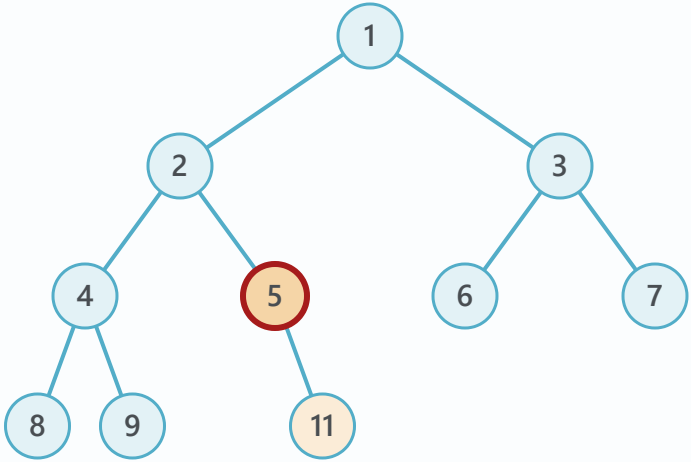
Hangisi? Dolu mu, Tam mı, Mükemmel mi?



Dolu (full)	**hayır**
Tam (complete)	**evet**
Mükemmel (perfect)	**hayır**

Son düzey (8, 9, 10) soldan sağa boşluksuz → **tam**. 5'in tek çocuğu (10) var → dolu değil.

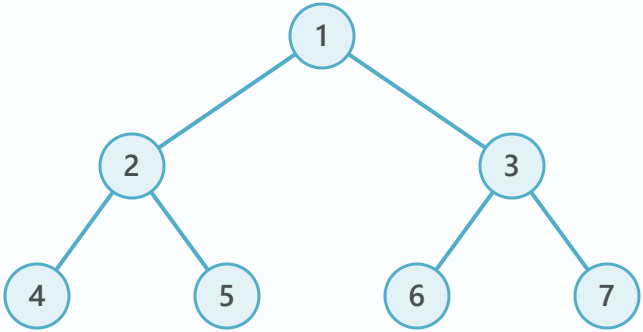
Hangisi? Dolu mu, Tam mı, Mükemmel mi?



Dolu (full)	**hayır**
Tam (complete)	**hayır**
Mükemmel (perfect)	**hayır**

5'in tek çocuğu var → dolu değil. 10. konum boşken 11 dolu → tam değil. **Ne tam ne dolu.**

Hangisi? Dolu mu, Tam mı, Mükemmel mi?



Dolu (full)	**evet**
Tam (complete)	**evet**
Mükemmel (perfect)	**evet**

Tüm iç düğümlerin 2 çocuğu var, tüm yapraklar aynı derinlikte → **mükemmel** (dolayısıyla dolu ve tam).

İkili Ağaç Özellikleri

Özellik	Formül	Örnek
d derinliğindeki en fazla düğüm	2^d	d = 3 → en fazla 8 düğüm
Yüksekliği h olan ağaçta en fazla düğüm	$2^{h+1} - 1$	h = 3 → 15 (mükemmel ağaç)
Yüksekliği h olan ağaçta en az düğüm	$h + 1$	çarpık (zincir) ağaç
n düğümlü ağacın yüksekliği	$\lfloor \log_2 n \rfloor \le h \le n - 1$	n = 15 → $3 \le h \le 14$
n düğümlü ağaçtaki kenar sayısı	$n - 1$	her düğümün tek ebeveyni
Dolu ikili ağaçta yaprak sayısı	iç düğüm + 1	7 düğüm → 3 iç, 4 yaprak

Yükseklik, ağaç işlemlerinin maliyetini belirler: dengeli bir ağaçta $h \approx \log_2 n$ (1 milyon düğüm → ≈ 20 düzey), çarpık ağaçta $h = n - 1$.

Dolu İkili Ağaçta Neden Yaprak = İç Düğüm + 1?



iç düğüm i = 0

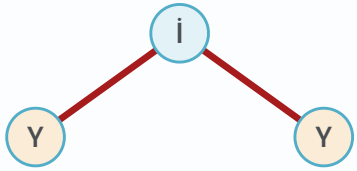
yaprak y = 1

y − i = 1

Tek düğümlü ağaç: kök bir **yaprak**tır (Y). i = 0, y = 1 → y = i + 1.

İ: iç düğüm, Y: yaprak; turuncu: son adımda eklenen yapraklar. Bu özellik ifade ağaçlarında (her işleç 2 işlenen) ve Huffman ağaçlarında kullanılır.

Dolu İkili Ağaçta Neden Yaprak = İç Düğüm + 1?



iç düğüm i = 1

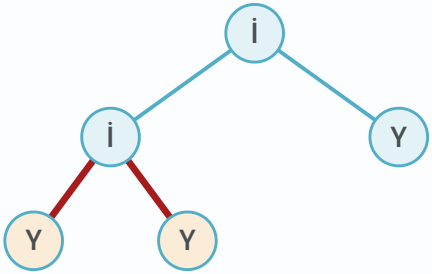
yaprak y = 2

y − i = 1

Bir yaprağa iki çocuk eklendi (dolu kalması için 0 ya da 2 çocuk): o yaprak **iç düğüm** olur (i + 1), iki yeni yaprak gelir (y − 1 + 2 = y + 1). Fark değişmez.

İ: iç düğüm, Y: yaprak; turuncu: son adımda eklenen yapraklar. Bu özellik ifade ağaçlarında (her işleç 2 işlenen) ve Huffman ağaçlarında kullanılır.

Dolu İkili Ağaçta Neden Yaprak = İç Düğüm + 1?



iç düğüm i = 2

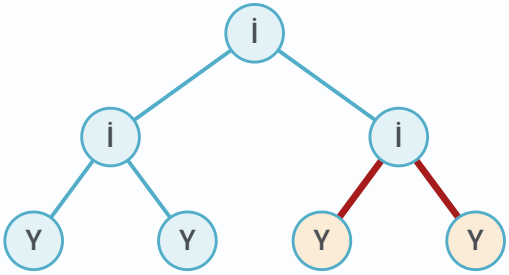
yaprak y = 3

y − i = 1

Bir yaprağa iki çocuk eklendi (dolu kalması için 0 ya da 2 çocuk): o yaprak **iç düğüm** olur (i + 1), iki yeni yaprak gelir (y − 1 + 2 = y + 1). Fark değişmez.

İ: iç düğüm, Y: yaprak; turuncu: son adımda eklenen yapraklar. Bu özellik ifade ağaçlarında (her işleç 2 işlenen) ve Huffman ağaçlarında kullanılır.

Dolu İkili Ağaçta Neden Yaprak = İç Düğüm + 1?



iç düğüm i = 3

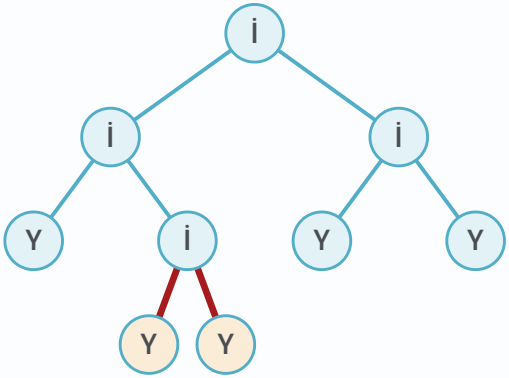
yaprak y = 4

y − i = 1

Bir yaprağa iki çocuk eklendi (dolu kalması için 0 ya da 2 çocuk): o yaprak **iç düğüm** olur (i + 1), iki yeni yaprak gelir (y − 1 + 2 = y + 1). Fark değişmez.

İ: iç düğüm, Y: yaprak; turuncu: son adımda eklenen yapraklar. Bu özellik ifade ağaçlarında (her işleç 2 işlenen) ve Huffman ağaçlarında kullanılır.

Dolu İkili Ağaçta Neden Yaprak = İç Düğüm + 1?



iç düğüm i = 4

yaprak y = 5

y − i = 1

Her genişletme i'yi de y'yi de 1 artırır; başlangıçta y − i = 1 olduğundan **her dolu ikili ağaçta y = i + 1** (tümevarım). Sonuç: n = 2i + 1 → n her zaman tek.

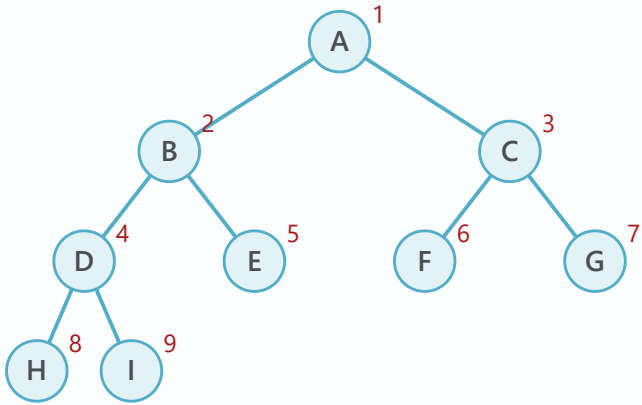
Adım 5 / 5

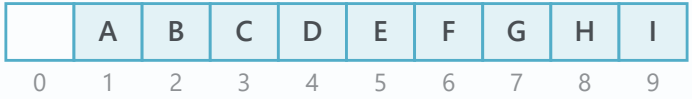
İ: iç düğüm, Y: yaprak; turuncu: son adımda eklenen yapraklar. Bu özellik ifade ağaçlarında (her işleç 2 işlenen) ve Huffman ağaçlarında kullanılır.

Dizi Gösterimi

Düğümler düzey düzey, soldan sağa diziye yerleştirilir. Kök 1 tabanlı dizide indis 1'de, 0 tabanlıda 0'dadır. İndis **i**'deki düğüm için:

	1 tabanlı	0 tabanlı
Sol çocuk	2i	2i + 1
Sağ çocuk	2i + 1	2i + 2
Ebeveyn	⌊i / 2⌋	⌊(i − 1) / 2⌋

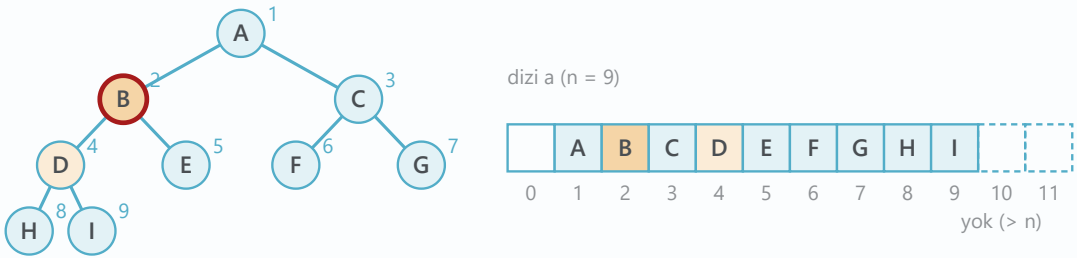




Örnek: E (i = 5) → çocuklar 10 ve 11 (boş), ebeveyn ⌊5/2⌋ = 2 (B).

Referans tutulmaz; ebeveyn/çocuk erişimi O(1) aritmetikle yapılır. Tam ikili ağaçlar için idealdir (ör. heap).

Kod İzleme: Dizi Gösteriminde İndis Hesabı



`sol(2) = 2·2 = 4` → a[4] = **D**, B'nin sol çocuğu (dizi 1 tabanlı, a[0] boş). Referans yok; konum aritmetikle bulunur.

Adım 1 / 16

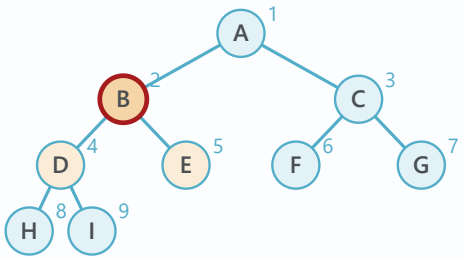
```
▶ int sol(int i)     { return 2 * i; }
2 int sag(int i)     { return 2 * i + 1; }
3 int ebeveyn(int i) { return i / 2; }
4 void kokeYol(char[] a, int i) {  // i'den köke
5   while (i >= 1) {
6     System.out.print(a[i] + " ");
7     i = ebeveyn(i);
8   }
9 }
```

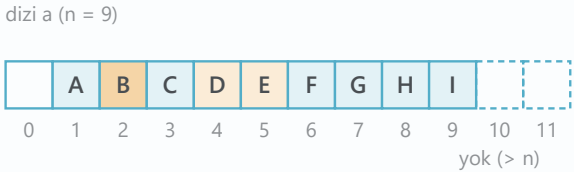
DEĞİŞKENLER

`i = 2 (B)` `sol = 4`

ÇIKTI

Kod İzleme: Dizi Gösteriminde İndis Hesabı





`sag(2) = 2·2 + 1 = 5` → a[5] = **E**. Sağ çocuk her zaman sol çocuğun hemen yanındadır.

Adım 2 / 16

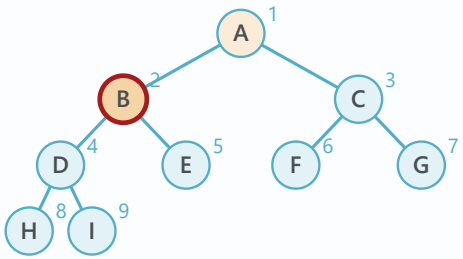
```
1 int sol(int i)     { return 2 * i; }
▶ int sag(int i)     { return 2 * i + 1; }
3 int ebeveyn(int i) { return i / 2; }
4 void kokeYol(char[] a, int i) {  // i'den köke
5   while (i >= 1) {
6     System.out.print(a[i] + " ");
7     i = ebeveyn(i);
8   }
9 }
```

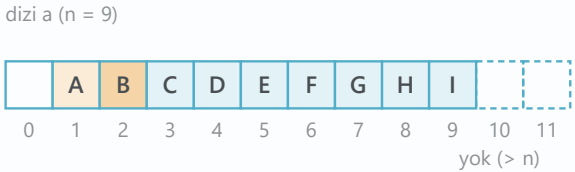
DEĞİŞKENLER

`i = 2 (B)` `sol = 4` `sag = 5`

ÇIKTI

Kod İzleme: Dizi Gösteriminde İndis Hesabı





`ebeveyn(2) = 2 / 2 = 1` → a[1] = **A** (kök).

Adım 3 / 16

```
1 int sol(int i)     { return 2 * i; }
2 int sag(int i)     { return 2 * i + 1; }
▶ int ebeveyn(int i) { return i / 2; }
4 void kokeYol(char[] a, int i) {  // i'den köke
5   while (i >= 1) {
6     System.out.print(a[i] + " ");
7     i = ebeveyn(i);
8   }
9 }
```

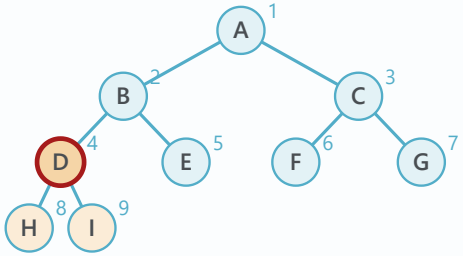
DEĞİŞKENLER

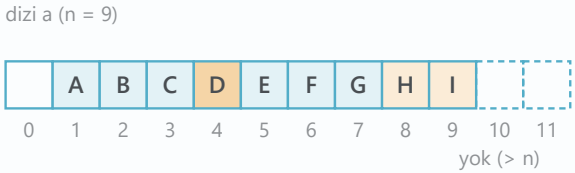
`i = 2 (B)` `sol = 4` `sag = 5`

`ebeveyn = 1`

ÇIKTI

Kod İzleme: Dizi Gösteriminde İndis Hesabı





D (i = 4): çocuklar **8** → **H** ve **9** → **I**.

Adım 4 / 16

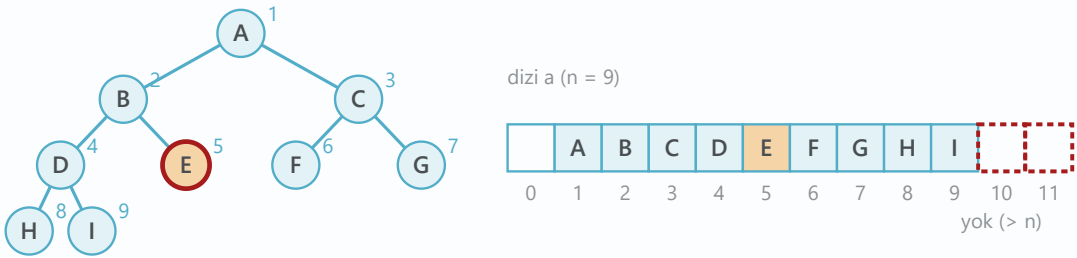
```
▶ int sol(int i)     { return 2 * i; }
▶ int sag(int i)     { return 2 * i + 1; }
3 int ebeveyn(int i) { return i / 2; }
4 void kokeYol(char[] a, int i) {  // i'den köke
5   while (i >= 1) {
6     System.out.print(a[i] + " ");
7     i = ebeveyn(i);
8   }
9 }
```

DEĞİŞKENLER

`i = 4 (D)` `sol = 8` `sag = 9`

ÇIKTI

Kod İzleme: Dizi Gösteriminde İndis Hesabı



E (i = 5): 10 ve 11 > n = 9 → **dizinin dışında**, yani E'nin çocuğu yok (yaprak). Kontrol: `2i ≤ n`.

Adım 5 / 16

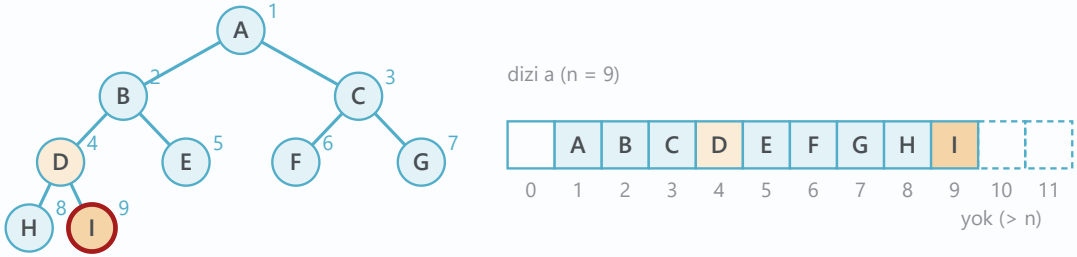
```
▶ int sol(int i)     { return 2 * i; }
▶ int sag(int i)     { return 2 * i + 1; }
3 int ebeveyn(int i) { return i / 2; }
4 void kokeYol(char[] a, int i) {  // i'den köke
5   while (i >= 1) {
6     System.out.print(a[i] + " ");
7     i = ebeveyn(i);
8   }
9 }
```

DEĞİŞKENLER

`i = 5 (E)` `sol = 10` `sag = 11`

ÇIKTI

Kod İzleme: Dizi Gösteriminde İndis Hesabı



`ebeveyn(9) = 9 / 2 = 4` : tam sayı bölmesi aşağı yuvarlar (⌊4,5⌋ = 4) → **D**. 8 ve 9 kardeş olduğundan ikisi de 4'ü verir.

Adım 6 / 16

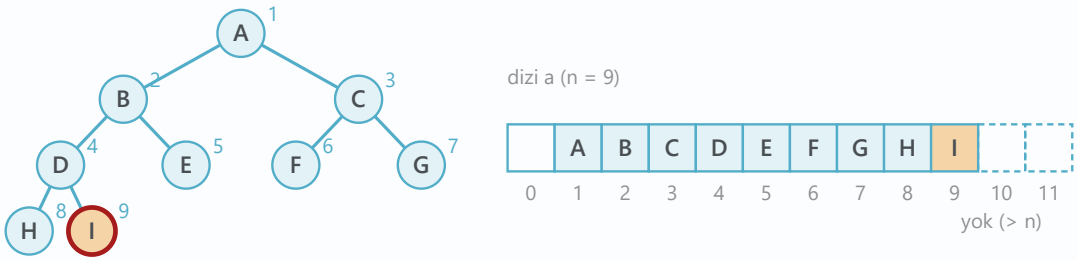
```
1 int sol(int i)     { return 2 * i; }
2 int sag(int i)     { return 2 * i + 1; }
▶ int ebeveyn(int i) { return i / 2; }
4 void kokeYol(char[] a, int i) {  // i'den köke
5   while (i >= 1) {
6     System.out.print(a[i] + " ");
7     i = ebeveyn(i);
8   }
9 }
```

DEĞİŞKENLER

`i = 9 (I)` `ebeveyn = 4`

ÇIKTI

Kod İzleme: Dizi Gösteriminde İndis Hesabı



`kokeYol(a, 9)` : I'dan köke kadar ebeveyn zinciri izlenecek.

Adım 7 / 16

```
1 int sol(int i)     { return 2 * i; }
2 int sag(int i)     { return 2 * i + 1; }
3 int ebeveyn(int i) { return i / 2; }
▶ void kokeYol(char[] a, int i) {  // i'den köke
5   while (i >= 1) {
6     System.out.print(a[i] + " ");
7     i = ebeveyn(i);
8   }
9 }
```

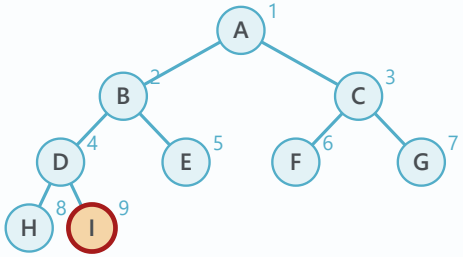
DEĞİŞKENLER

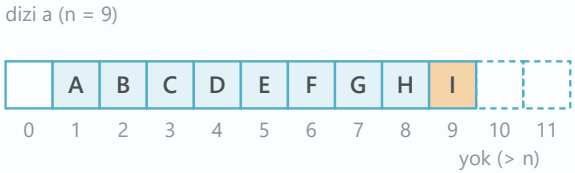
`a = [_, A, B, C, D, E, F, G, H, I]`

`i = 9`

ÇIKTI

Kod İzleme: Dizi Gösteriminde İndis Hesabı





9 ≥ 1 → döngü sürer; **a[9] = I** yazdırılır.

Adım 8 / 16

```
1 int sol(int i)     { return 2 * i; }
2 int sag(int i)     { return 2 * i + 1; }
3 int ebeveyn(int i) { return i / 2; }
4 void kokeYol(char[] a, int i) {  // i'den köke
▶   while (i >= 1) {
▶     System.out.print(a[i] + " ");
7     i = ebeveyn(i);
8   }
9 }
```

DEĞİŞKENLER

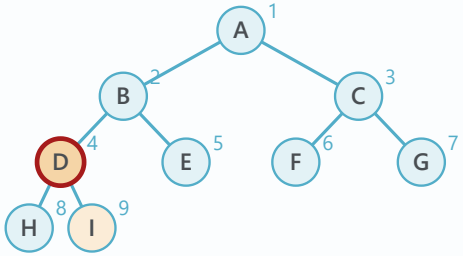
`a = [_, A, B, C, D, E, F, G, H, I]`

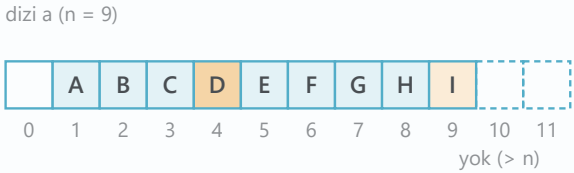
`i = 9`

ÇIKTI

```
I
```

Kod İzleme: Dizi Gösteriminde İndis Hesabı





`i = ebeveyn(9) = 4` → D.

Adım 9 / 16

```
1 int sol(int i)     { return 2 * i; }
2 int sag(int i)     { return 2 * i + 1; }
3 int ebeveyn(int i) { return i / 2; }
4 void kokeYol(char[] a, int i) {  // i'den köke
5   while (i >= 1) {
6     System.out.print(a[i] + " ");
      i = ebeveyn(i);
8   }
9 }
```

DEĞİŞKENLER

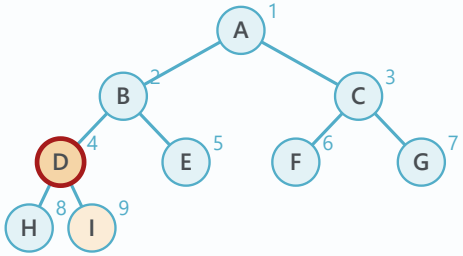
`a = [_, A, B, C, D, E, F, G, H, I]`

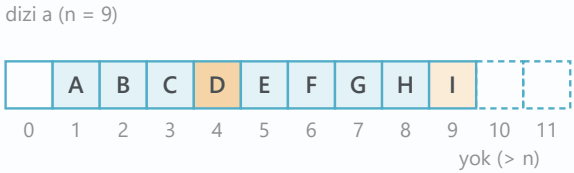
`i = 4`

ÇIKTI

```
I
```

Kod İzleme: Dizi Gösteriminde İndis Hesabı





4 ≥ 1 → döngü sürer; **a[4] = D** yazdırılır.

Adım 10 / 16

```
1 int sol(int i)     { return 2 * i; }
2 int sag(int i)     { return 2 * i + 1; }
3 int ebeveyn(int i) { return i / 2; }
4 void kokeYol(char[] a, int i) {  // i'den köke
▶   while (i >= 1) {
▶     System.out.print(a[i] + " ");
7     i = ebeveyn(i);
8   }
9 }
```

DEĞİŞKENLER

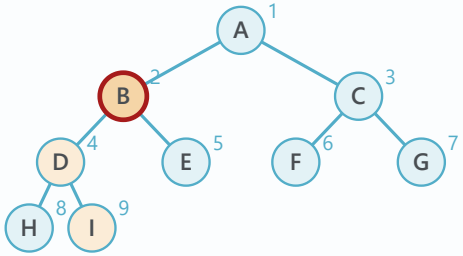
`a = [_, A, B, C, D, E, F, G, H, I]`

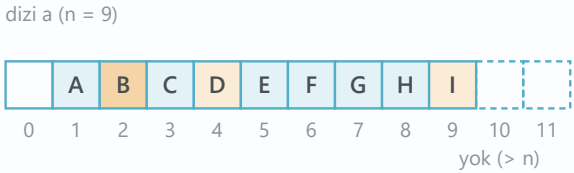
`i = 4`

ÇIKTI

```
I D
```

Kod İzleme: Dizi Gösteriminde İndis Hesabı





`i = ebeveyn(4) = 2` → B.

Adım 11 / 16

```
1  int sol(int i)     { return 2 * i; }
2  int sag(int i)     { return 2 * i + 1; }
3  int ebeveyn(int i) { return i / 2; }
4  void kokeYol(char[] a, int i) {  // i'den köke
5    while (i >= 1) {
6      System.out.print(a[i] + " ");
▶      i = ebeveyn(i);
8    }
9  }
```

DEĞİŞKENLER

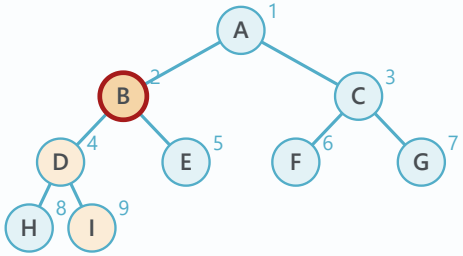
`a = [_, A, B, C, D, E, F, G, H, I]`

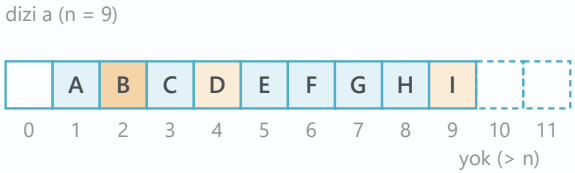
`i = 2`

ÇIKTI

```
I D
```

Kod İzleme: Dizi Gösteriminde İndis Hesabı





2 ≥ 1 → döngü sürer; **a[2] = B** yazdırılır.

Adım 12 / 16

```
1 int sol(int i)     { return 2 * i; }
2 int sag(int i)     { return 2 * i + 1; }
3 int ebeveyn(int i) { return i / 2; }
4 void kokeYol(char[] a, int i) {  // i'den köke
▶   while (i >= 1) {
▶     System.out.print(a[i] + " ");
7     i = ebeveyn(i);
8   }
9 }
```

DEĞİŞKENLER

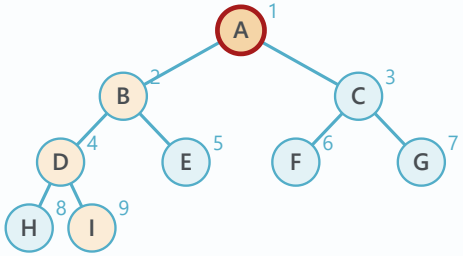
`a = [_, A, B, C, D, E, F, G, H, I]`

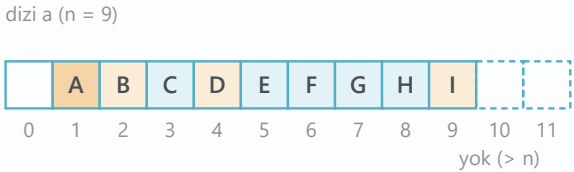
`i = 2`

ÇIKTI

```
I D B
```

Kod İzleme: Dizi Gösteriminde İndis Hesabı





`i = ebeveyn(2) = 1` → A.

Adım 13 / 16

```
1 int sol(int i)     { return 2 * i; }
2 int sag(int i)     { return 2 * i + 1; }
3 int ebeveyn(int i) { return i / 2; }
4 void kokeYol(char[] a, int i) {  // i'den köke
5   while (i >= 1) {
6     System.out.print(a[i] + " ");
      i = ebeveyn(i);
8   }
9 }
```

DEĞİŞKENLER

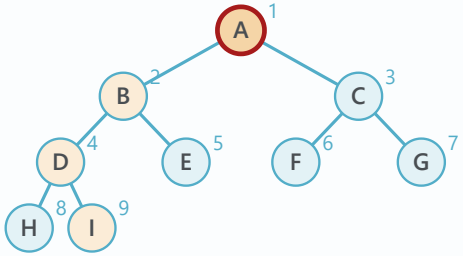
`a = [_, A, B, C, D, E, F, G, H, I]`

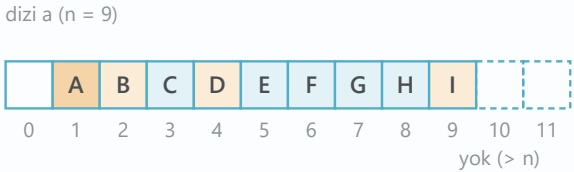
`i = 1`

ÇIKTI

```
I D B
```

Kod İzleme: Dizi Gösteriminde İndis Hesabı





1 ≥ 1 → döngü sürer; **a[1] = A** yazdırılır.

Adım 14 / 16

```
1 int sol(int i)     { return 2 * i; }
2 int sag(int i)     { return 2 * i + 1; }
3 int ebeveyn(int i) { return i / 2; }
4 void kokeYol(char[] a, int i) {  // i'den köke
▶   while (i >= 1) {
▶     System.out.print(a[i] + " ");
7     i = ebeveyn(i);
8   }
9 }
```

DEĞİŞKENLER

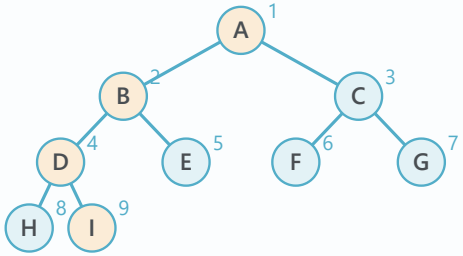
`a = [_, A, B, C, D, E, F, G, H, I]`

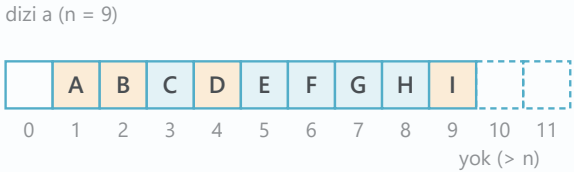
`i = 1`

ÇIKTI

```
I D B A
```

Kod İzleme: Dizi Gösteriminde İndis Hesabı





`i = ebeveyn(1) = 0` (kökün ebeveyni yok).

Adım 15 / 16

```
1 int sol(int i)     { return 2 * i; }
2 int sag(int i)     { return 2 * i + 1; }
3 int ebeveyn(int i) { return i / 2; }
4 void kokeYol(char[] a, int i) {  // i'den köke
5   while (i >= 1) {
6     System.out.print(a[i] + " ");
      i = ebeveyn(i);
8   }
9 }
```

DEĞİŞKENLER

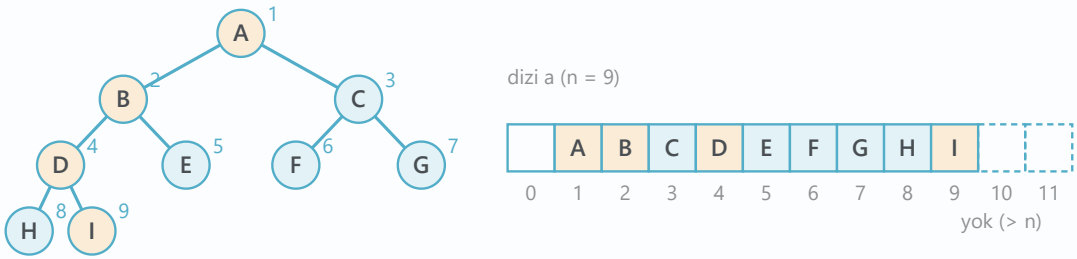
`a = [_, A, B, C, D, E, F, G, H, I]`

`i = 0`

ÇIKTI

```
I D B A
```

Kod İzleme: Dizi Gösteriminde İndis Hesabı



0 ≥ 1 yanlış → döngü biter. **Çıktı: I D B A**. Her adım indisi yarıya indirir → **O(log n)** (derinlik kadar).

Adım 16 / 16

```
1 int sol(int i)     { return 2 * i; }
2 int sag(int i)     { return 2 * i + 1; }
3 int ebeveyn(int i) { return i / 2; }
4 void kokeYol(char[] a, int i) {  // i'den köke
▶   while (i >= 1) {
6     System.out.print(a[i] + " ");
7     i = ebeveyn(i);
8   }
9 }
```

DEĞİŞKENLER

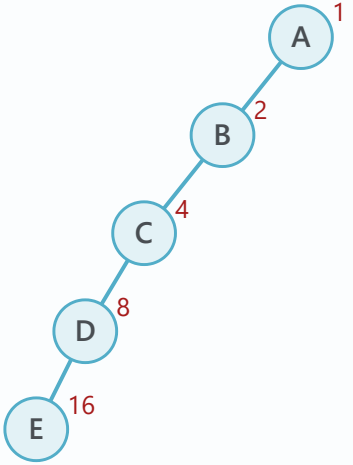
`a = [_, A, B, C, D, E, F, G, H, I]`

`i = 0`

ÇIKTI

```
I D B A
```

Dizide Bellek İsrafı



- Çarpık ağaçta düğümlerin indisleri katlanarak büyür: 1, 2, 4, 8, 16.
- 5 düğüm için **16 hücre** gerekir; 11 hücre boş kalır. Sağa çarpık olsaydı indisler 1, 3, 7, 15, 31 olurdu → 31 hücre.
- Genel olarak n düğümlü bir ağaç $2^n - 1$ hücreye kadar yer isteyebilir.



Dizi gösterimi

+ Referans yok, hızlı indis hesabı
+ Tam ağaçlarda boşluksuz

Ama

– Seyrek/çarpık ağaçta büyük israf
– Boyut sabit; büyütmek için kopyalama

Bağlı Gösterim (Linked Representation)

- Her ağaç düğümü bir nesnedir: **veri** + **sol** ve **sağ** çocuk referansı.
- Çocuk yoksa referans `null` 'dır.
- Boyut dinamiktir; yalnızca var olan düğümler için bellek ayrılır.
- Bir düğüme ulaşmak için kökten başlayıp referansları izlemek gerekir.

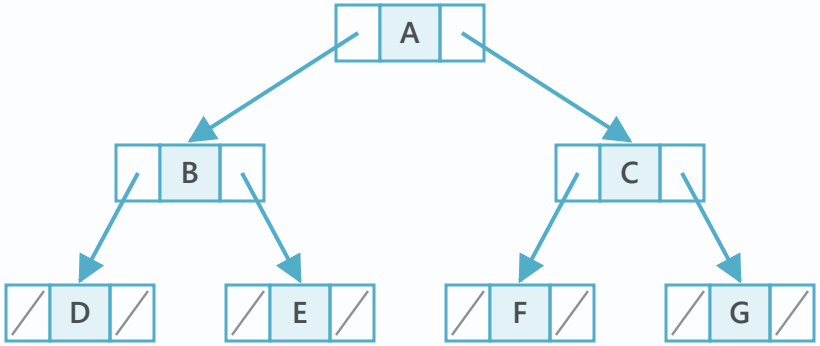



```java
public class AgacDugumu {
    int veri;
    AgacDugumu sol;   // sol çocuk (yoksa null)
    AgacDugumu sag;   // sağ çocuk (yoksa null)

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

```java
// 9 kökünde küçük bir ağaç kuralım
AgacDugumu kok = new AgacDugumu(9);
kok.sol = new AgacDugumu(2);
kok.sag = new AgacDugumu(3);
kok.sol.sol = new AgacDugumu(4);
```

Bağlı Gösterim: Örnek



çapraz çizgi = null referans

7 düğüm

14 referans alanı: 6'sı dolu (kenar sayısı n − 1), 8'i null.

Genel kural

n düğümlü ikili ağaçta her zaman n + 1 null referans vardır.

Ebeveyne çıkış

Gerekirse düğüme bir `ebeveyn` referansı da eklenir.

Dizi mi, Bağlı mı?

Ölçüt	Dizi gösterimi	Bağlı gösterim
Bellek	Tam ağaçta en az; seyrek ağaçta $2^n - 1$'e kadar israf	Düğüm başına 2 ek referans; israf yok
Çocuk / ebeveyn erişimi	O(1), indis aritmetiğiyle	Çocuk O(1); ebeveyn için ek referans gerekir
i. düzeydeki k. düğüm	O(1): indis $2^i + k - 1$ (1 tabanlı, $k \geq 1$)	Kökten gezinme gerekir
Büyüme	Dizi dolunca kopyalanarak büyütülür	Yeni düğüm eklemek O(1) (konum biliniyorsa)
Uygun olduğu yer	Tam ikili ağaçlar (heap)	Genel ve değişken şekilli ağaçlar (BST, ifade ağacı)

Kural: ağacın şekli **tam** kalıyorsa dizi, şekli öngörülemiyorsa bağlı gösterim tercih edilir.

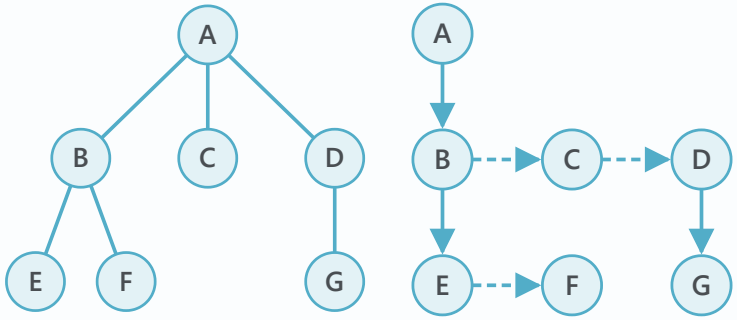
GÖSTERİMLER

Genel Ağaç Gösterimi

Çocuk sayısı sınırsız olduğunda iki yaygın yol vardır:

```
// 1) Çocuk listesi
class GenelDugum {
    int veri;
    List<GenelDugum> cocuklar
        = new ArrayList<>();
}
// 2) İlk çocuk – sonraki kardeş
class KardesDugum {
    int veri;
    KardesDugum ilkCocuk;
    KardesDugum sonrakiKardes;
}
```

İkinci yolda her düğüm yine iki referans tutar: **ilk çocuk** (aşağı) ve **sonraki kardeş** (sağa). Böylece her genel ağaç bir ikili ağaç gibi saklanabilir.



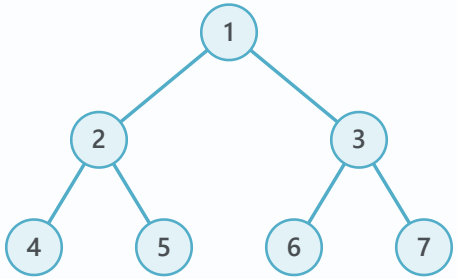
Solda genel ağaç; sağda ilk çocuk (düz ok) – sonraki kardeş (kesikli ok) gösterimi.

Denemek için: Genel Ağaç simülatörü

GEZİNMELER

Ağaç Gezinmeleri

Gezinme: Her düğümü bir kez ziyaret eder. Derinlik öncelikli (DFS) gezinmeler kökün ziyaret zamanına göre; düzey sırası (BFS) ise düzeylere göre ilerler.



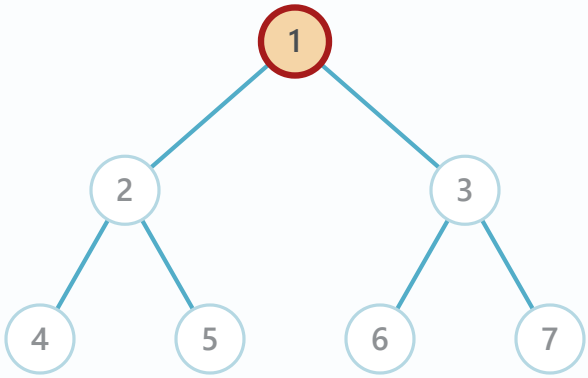
Gezinme	Sıra	Çıktı	Tipik kullanım
Kök önde (preorder)	kök → sol → sağ	1 2 4 5 3 6 7	Ağacı kopyalamak, önek ifade
Kök ortada (inorder)	sol → kök → sağ	4 2 5 1 6 3 7	BST'yi sıralı gezinmek
Kök sonda (postorder)	sol → sağ → kök	4 5 2 6 7 3 1	Silme, boyut/yükseklik hesabı
Düzey sırası (level-order)	düzey düzey, soldan sağa	1 2 3 4 5 6 7	Düzey bazlı işlemler

Kök Önde Gezinme (Preorder)

1. Kökü ziyaret et.
2. Sol alt ağacı kök önde gez.
3. Sağ alt ağacı kök önde gez.

Kullanım: ağacı kopyalamak, düğümleri diziye yazmak, önek (prefix) ifade.

```
ziyaret(kok);
kokBasta(kok.sol);
kokBasta(kok.sag);
```



Çıktı: 1

şimdi ziyaret edildi bekliyor

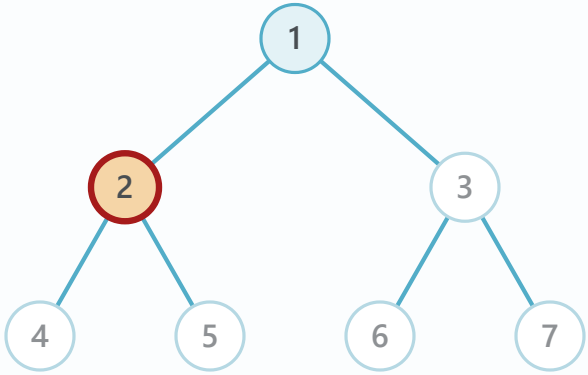
Kök 1 ziyaret edilir; sonra sol alt ağaca geçilir.

Kök Önde Gezinme (Preorder)

1. Kökü ziyaret et.
2. Sol alt ağacı kök önde gez.
3. Sağ alt ağacı kök önde gez.

Kullanım: ağacı kopyalamak, düğümleri diziye yazmak, önek (prefix) ifade.

```
ziyaret(kok);
kokBasta(kok.sol);
kokBasta(kok.sag);
```



Çıktı: 1 2

şimdi ziyaret edildi bekliyor

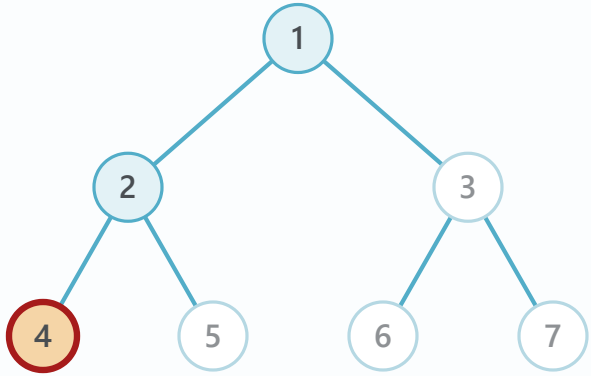
2 ziyaret edilir; sonra onun sol alt ağacı.

Kök Önde Gezinme (Preorder)

1. Kökü ziyaret et.
2. Sol alt ağacı kök önde gez.
3. Sağ alt ağacı kök önde gez.

Kullanım: ağacı kopyalamak, düğümleri diziye yazmak, önek (prefix) ifade.

```
ziyaret(kok);
kokBasta(kok.sol);
kokBasta(kok.sag);
```



Çıktı: 1 2 4

şimdi ziyaret edildi bekliyor

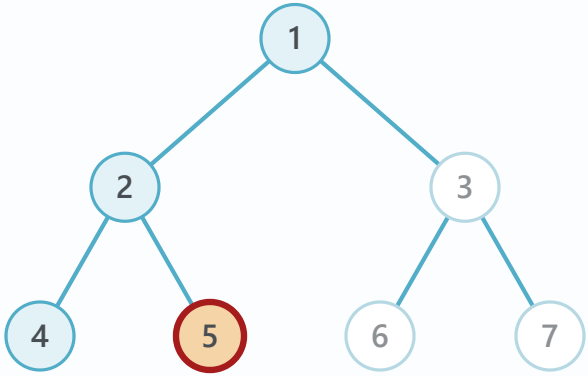
4 yaprak: ziyaret edilir, çocuğu yok.

Kök Önde Gezinme (Preorder)

1. Kökü ziyaret et.
2. Sol alt ağacı kök önde gez.
3. Sağ alt ağacı kök önde gez.

Kullanım: ağacı kopyalamak, düğümleri diziye yazmak, önek (prefix) ifade.

```
ziyaret(kok);
kokBasta(kok.sol);
kokBasta(kok.sag);
```



Çıktı: **1 2 4 5**

şimdi ziyaret edildi bekliyor

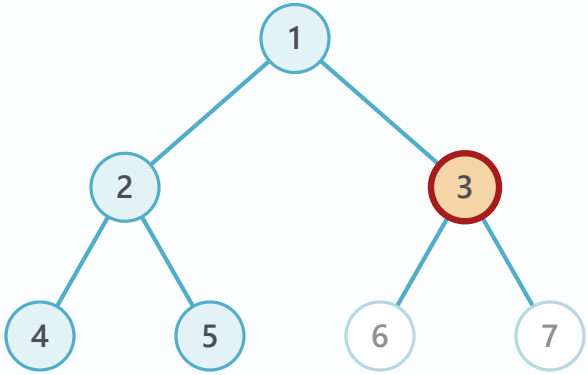
2'nin sağ alt ağacı: 5.

Kök Önde Gezinme (Preorder)

1. Kökü ziyaret et.
2. Sol alt ağacı kök önde gez.
3. Sağ alt ağacı kök önde gez.

Kullanım: ağacı kopyalamak, düğümleri diziye yazmak, önek (prefix) ifade.

```
ziyaret(kok);
kokBasta(kok.sol);
kokBasta(kok.sag);
```



Çıktı: **1 2 4 5 3**

şimdi ziyaret edildi bekliyor

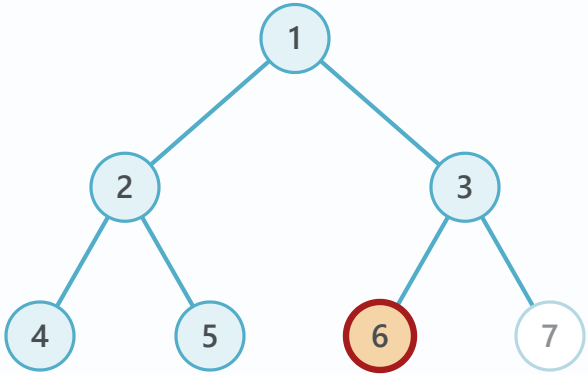
Sol alt ağaç bitti; 1'in sağ alt ağacı: 3.

Kök Önde Gezinme (Preorder)

1. Kökü ziyaret et.
2. Sol alt ağacı kök önde gez.
3. Sağ alt ağacı kök önde gez.

Kullanım: ağacı kopyalamak, düğümleri diziye yazmak, önek (prefix) ifade.

```
ziyaret(kok);
kokBasta(kok.sol);
kokBasta(kok.sag);
```



Çıktı: **1 2 4 5 3 6**

şimdi ziyaret edildi bekliyor

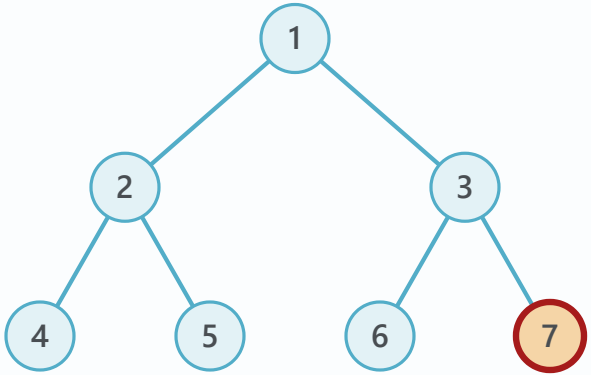
3'ün sol çocuğu 6.

Kök Önde Gezinme (Preorder)

1. Kökü ziyaret et.
2. Sol alt ağacı kök önde gez.
3. Sağ alt ağacı kök önde gez.

Kullanım: ağacı kopyalamak, düğümleri diziye yazmak, önek (prefix) ifade.

```
ziyaret(kok);
kokBasta(kok.sol);
kokBasta(kok.sag);
```



Çıktı: **1 2 4 5 3 6 7**

şimdi ziyaret edildi bekliyor

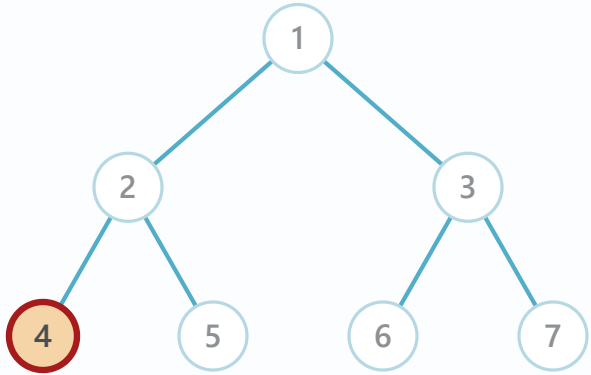
3'ün sağ çocuğu 7. Çıktı: 1 2 4 5 3 6 7

Kök Ortada Gezinme (Inorder)

1. Sol alt ağacı kök ortada gez.
2. Kökü ziyaret et.
3. Sağ alt ağacı kök ortada gez.

Kullanım: ikili arama ağacında anahtarları küçükten büyüğe sıralı vermek, ara ek (infix) ifade.

```
kokOrtada(kok.sol);
ziyaret(kok);
kokOrtada(kok.sag);
```



Çıktı: 4

şimdi ziyaret edildi bekliyor

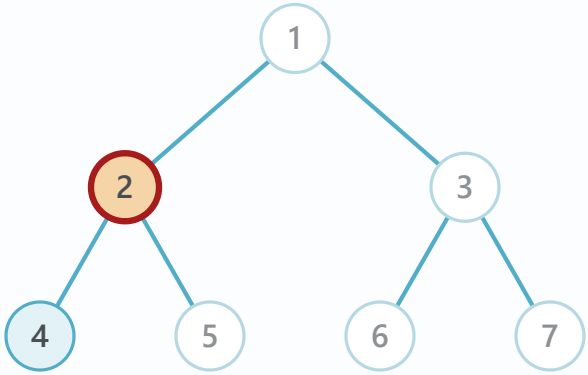
Kökten hep sola inilir; ilk ziyaret en soldaki düğüm: 4.

Kök Ortada Gezinme (Inorder)

1. Sol alt ağacı kök ortada gez.
2. Kökü ziyaret et.
3. Sağ alt ağacı kök ortada gez.

Kullanım: ikili arama ağacında anahtarları küçükten büyüğe sıralı vermek, ara ek (infix) ifade.

```
kokOrtada(kok.sol);
ziyaret(kok);
kokOrtada(kok.sag);
```



Çıktı: 4 2

şimdi ziyaret edildi bekliyor

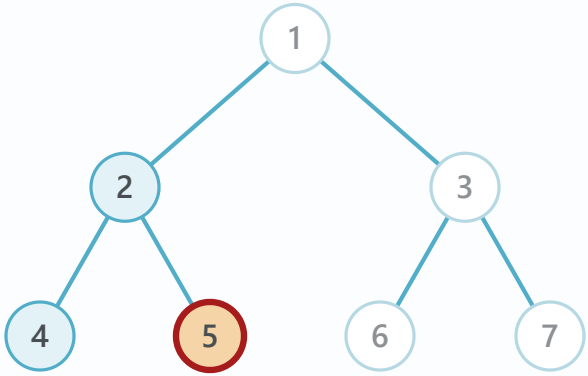
4'ün alt ağacı bitti → ebeveyni 2.

Kök Ortada Gezinme (Inorder)

1. Sol alt ağacı kök ortada gez.
2. Kökü ziyaret et.
3. Sağ alt ağacı kök ortada gez.

Kullanım: ikili arama ağacında anahtarları küçükten büyüğe sıralı vermek, ara ek (infix) ifade.

```
kokOrtada(kok.sol);
ziyaret(kok);
kokOrtada(kok.sag);
```



Çıktı: 4 2 5

şimdi ziyaret edildi bekliyor

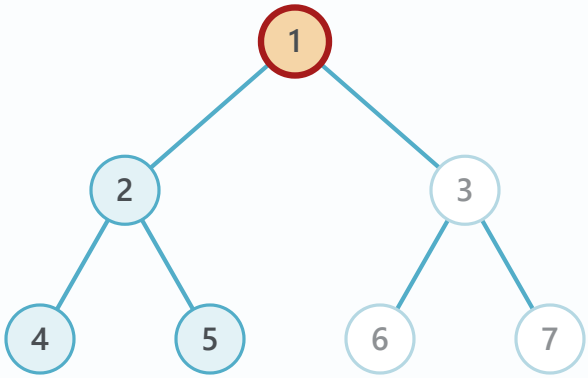
2'nin sağ alt ağacı: 5.

Kök Ortada Gezinme (Inorder)

1. Sol alt ağacı kök ortada gez.
2. Kökü ziyaret et.
3. Sağ alt ağacı kök ortada gez.

Kullanım: ikili arama ağacında anahtarları küçükten büyüğe sıralı vermek, ara ek (infix) ifade.

```
kokOrtada(kok.sol);
ziyaret(kok);
kokOrtada(kok.sag);
```



Çıktı: **4 2 5 1**

şimdi ziyaret edildi bekliyor

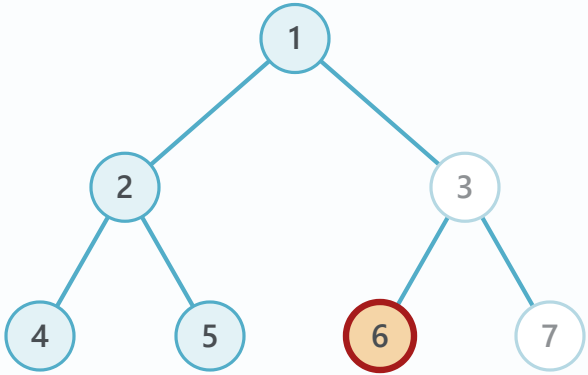
Sol alt ağaç tamamlandı → kök 1.

Kök Ortada Gezinme (Inorder)

1. Sol alt ağacı kök ortada gez.
2. Kökü ziyaret et.
3. Sağ alt ağacı kök ortada gez.

Kullanım: ikili arama ağacında anahtarları küçükten büyüğe sıralı vermek, ara ek (infix) ifade.

```
kokOrtada(kok.sol);
ziyaret(kok);
kokOrtada(kok.sag);
```



Çıktı: 4 2 5 1 6

şimdi ziyaret edildi bekliyor

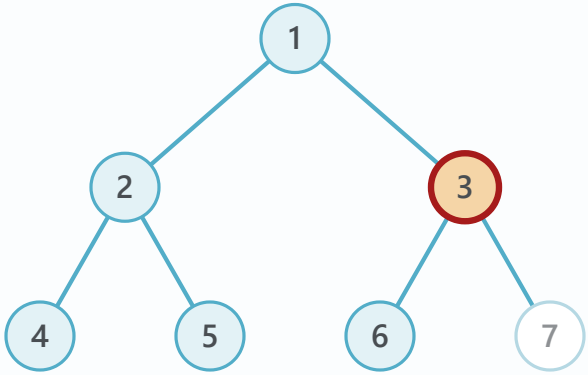
Sağ alt ağaçta yine en sola inilir: 6.

Kök Ortada Gezinme (Inorder)

1. Sol alt ağacı kök ortada gez.
2. Kökü ziyaret et.
3. Sağ alt ağacı kök ortada gez.

Kullanım: ikili arama ağacında anahtarları küçükten büyüğe sıralı vermek, ara ek (infix) ifade.

```
kokOrtada(kok.sol);
ziyaret(kok);
kokOrtada(kok.sag);
```



Çıktı: **4 2 5 1 6 3**

şimdi ziyaret edildi bekliyor

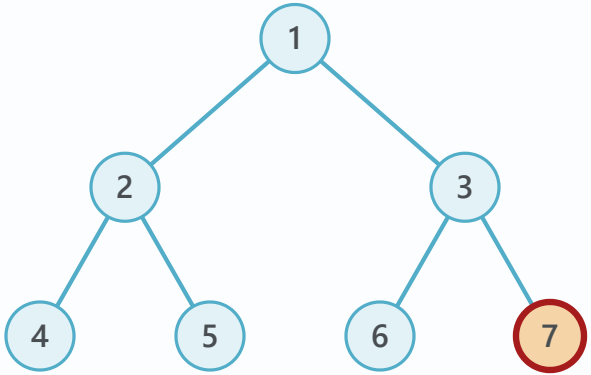
6'dan sonra ebeveyni 3.

Kök Ortada Gezinme (Inorder)

1. Sol alt ağacı kök ortada gez.
2. Kökü ziyaret et.
3. Sağ alt ağacı kök ortada gez.

Kullanım: ikili arama ağacında anahtarları küçükten büyüğe sıralı vermek, ara ek (infix) ifade.

```
kokOrtada(kok.sol);
ziyaret(kok);
kokOrtada(kok.sag);
```



Çıktı: **4 2 5 1 6 3 7**

şimdi ziyaret edildi bekliyor

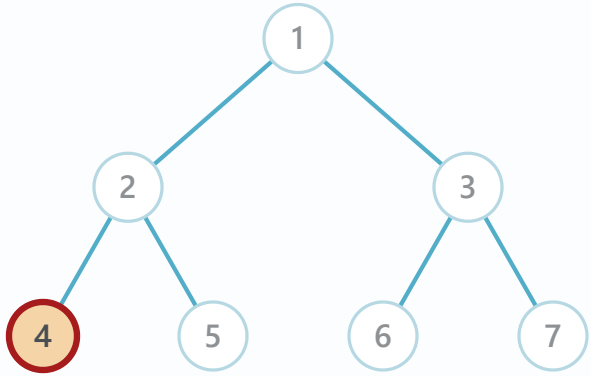
Son olarak 7. Çıktı: 4 2 5 1 6 3 7

Kök Sonda Gezinme (Postorder)

1. Sol alt ağacı kök sonda gez.
2. Sağ alt ağacı kök sonda gez.
3. Kökü ziyaret et.

Kullanım: ağacı silmek (önce çocuklar), boyut/yükseklik hesabı, sonek (postfix) ifade.

```
kokSonda(kok.sol);
kokSonda(kok.sag);
ziyaret(kok);
```



Çıktı: 4

şimdi ziyaret edildi bekliyor

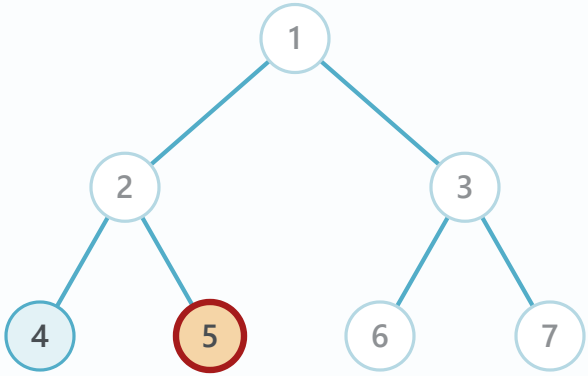
En soldaki yaprak 4 ilk ziyaret edilir.

Kök Sonda Gezinme (Postorder)

1. Sol alt ağacı kök sonda gez.
2. Sağ alt ağacı kök sonda gez.
3. Kökü ziyaret et.

Kullanım: ağacı silmek (önce çocuklar), boyut/yükseklik hesabı, sonek (postfix) ifade.

```
kokSonda(kok.sol);
kokSonda(kok.sag);
ziyaret(kok);
```



Çıktı: 4 5

şimdi ziyaret edildi bekliyor

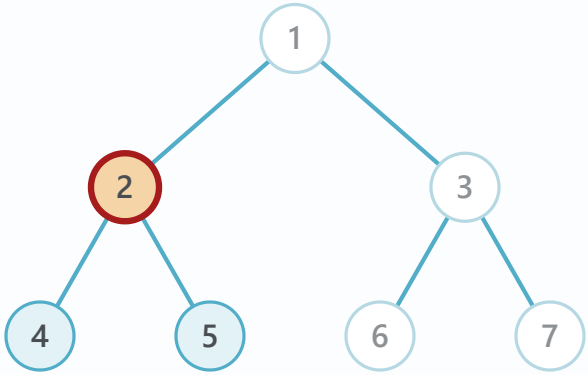
Sonra 2'nin sağ çocuğu 5.

Kök Sonda Gezinme (Postorder)

1. Sol alt ağacı kök sonda gez.
2. Sağ alt ağacı kök sonda gez.
3. Kökü ziyaret et.

Kullanım: ağacı silmek (önce çocuklar), boyut/yükseklik hesabı, sonek (postfix) ifade.

```
kokSonda(kok.sol);
kokSonda(kok.sag);
ziyaret(kok);
```



Çıktı: 4 5 2

şimdi ziyaret edildi bekliyor

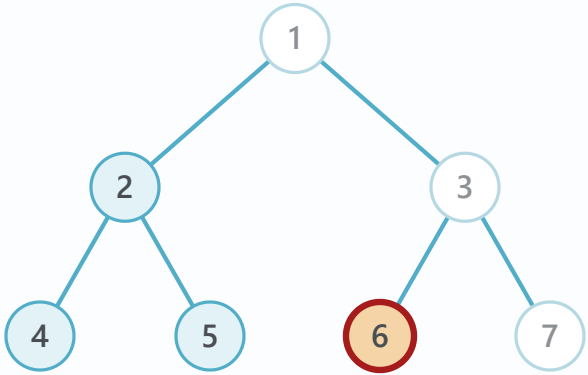
2'nin iki alt ağacı da bitti → 2.

Kök Sonda Gezinme (Postorder)

1. Sol alt ağacı kök sonda gez.
2. Sağ alt ağacı kök sonda gez.
3. Kökü ziyaret et.

Kullanım: ağacı silmek (önce çocuklar), boyut/yükseklik hesabı, sonek (postfix) ifade.

```
kokSonda(kok.sol);
kokSonda(kok.sag);
ziyaret(kok);
```



Çıktı: 4 5 2 6

şimdi ziyaret edildi bekliyor

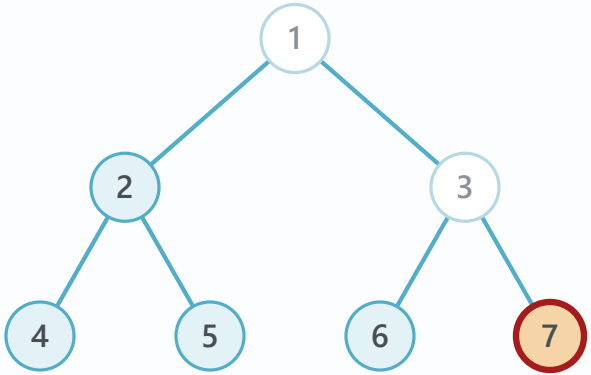
1'in sağ alt ağacı: önce 6.

Kök Sonda Gezinme (Postorder)

1. Sol alt ağacı kök sonda gez.
2. Sağ alt ağacı kök sonda gez.
3. Kökü ziyaret et.

Kullanım: ağacı silmek (önce çocuklar), boyut/yükseklik hesabı, sonek (postfix) ifade.

```
kokSonda(kok.sol);
kokSonda(kok.sag);
ziyaret(kok);
```



Çıktı: 4 5 2 6 7

şimdi ziyaret edildi bekliyor

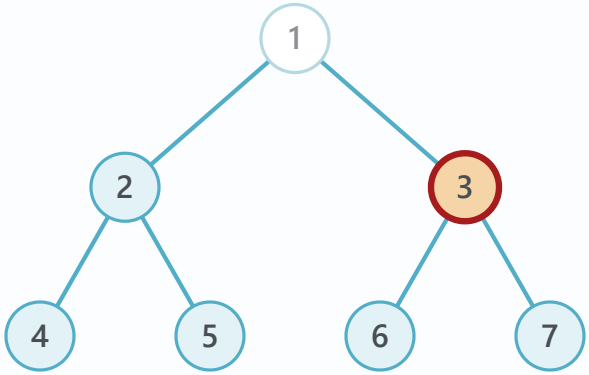
Sonra 7.

Kök Sonda Gezinme (Postorder)

1. Sol alt ağacı kök sonda gez.
2. Sağ alt ağacı kök sonda gez.
3. Kökü ziyaret et.

Kullanım: ağacı silmek (önce çocuklar), boyut/yükseklik hesabı, sonek (postfix) ifade.

```
kokSonda(kok.sol);
kokSonda(kok.sag);
ziyaret(kok);
```



Çıktı: **4 5 2 6 7 3**

şimdi ziyaret edildi bekliyor

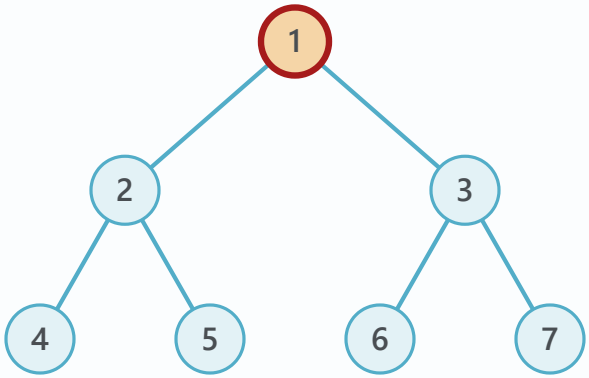
3'ün çocukları bitti → 3.

Kök Sonda Gezinme (Postorder)

1. Sol alt ağacı kök sonda gez.
2. Sağ alt ağacı kök sonda gez.
3. Kökü ziyaret et.

Kullanım: ağacı silmek (önce çocuklar), boyut/yükseklik hesabı, sonek (postfix) ifade.

```
kokSonda(kok.sol);
kokSonda(kok.sag);
ziyaret(kok);
```

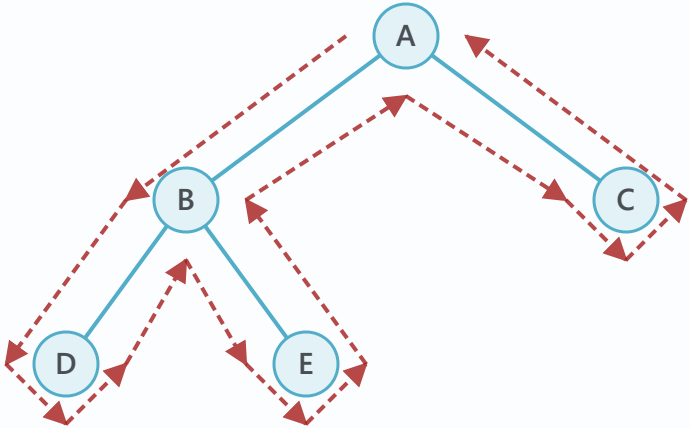


Çıktı: 4 5 2 6 7 3 1

şimdi ziyaret edildi bekliyor

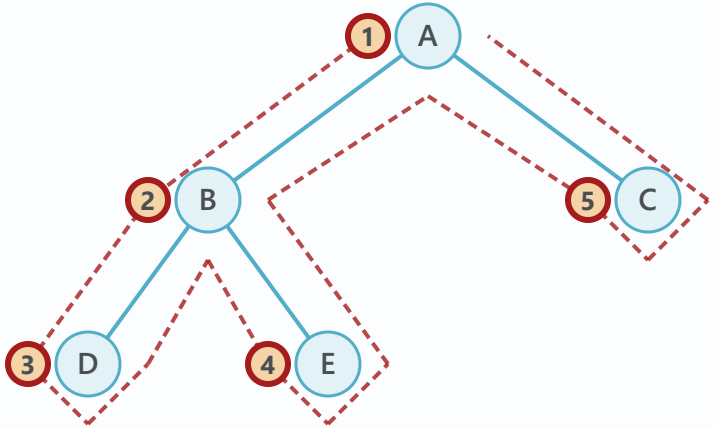
Kök en son ziyaret edilir: 1. Çıktı: 4 5 2 6 7 3 1

Üç Gezinme, Tek Tur: Euler Turu



Ağacın etrafında, kökün solundan başlayıp kenarlara sol elimizle dokunarak yürüyelim (**Euler turu**). Her düğüme **üç kez** uğranır: solundan, altından, sağından.

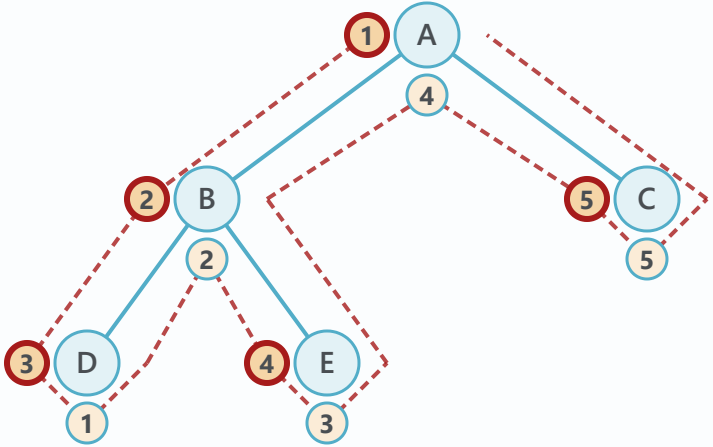
Üç Gezinme, Tek Tur: Euler Turu



Sol işaret → Kök önde (preorder):
A B D E C

Bir düğümü **soldan geçerken** yazarsak (ilk karşılaşma) kök önde sırası çıkar: **A B D E C**.

Üç Gezinme, Tek Tur: Euler Turu



Sol işaret → Kök önde (preorder):

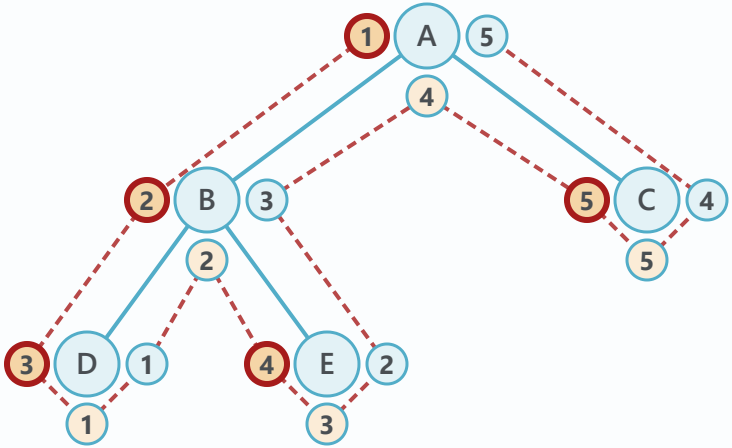
A B D E C

Alt işaret → Kök ortada (inorder):

D B E A C

Altından geçerken (sol alt ağaç bitmiş, sağ başlamamış) yazarsak kök ortada sırası çıkar: **D B E A C**.

Üç Gezinme, Tek Tur: Euler Turu



Sol işaret → Kök önde (preorder):
A B D E C

Alt işaret → Kök ortada (inorder):
D B E A C

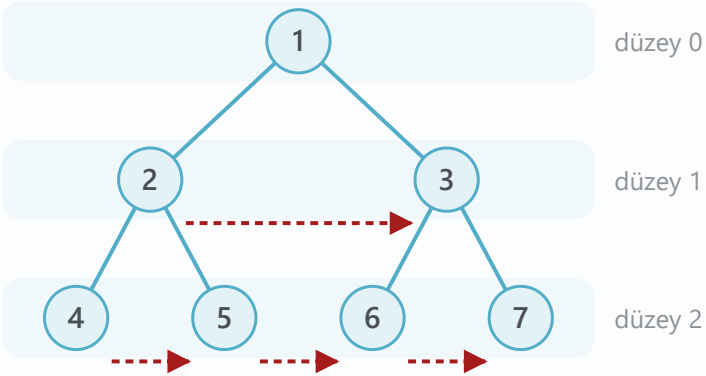
Sağ işaret → Kök sonda (postorder):
D E B C A

Sağından geçerken (iki alt ağaç da bitmiş) yazarsak kök sonda sırası çıkar: **D E B C A**. Üç gezinme aynı turun farklı anlarıdır; tur O(n) sürer.

Düzey Sırası Gezinme (Level-order)

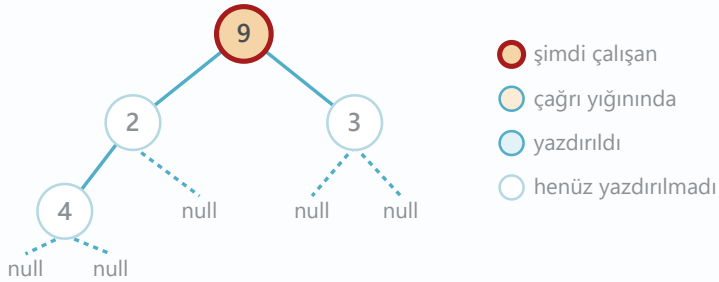
- Kökten başlar, **her düzeyi soldan sağa** tarar, sonra bir alt düzeye iner.
- Genişlik öncelikli aramanın (BFS) ağaçtaki hâlidir.
- Özyineleme yerine bir **kuyruk** (FIFO) kullanır: önce gelen düğümün çocukları önce işlenir.

Kullanım: düzey bazlı işlemler (her düzeyin toplamı, en yakın yaprak), ağacı düzey düzey yazdırmak.



Çıktı: **1 2 3 4 5 6 7**

Özyinelemeli Kök Önde ve Çağrı Yığını



`kokBasta(9)` çağrıldı (kök). null değil → önce **kök yazdırılır: 9**, sonra alt ağaçlar.

Adım 1 / 13

```
1 public void kokBasta(AgacDugumu kok) {
2   if (kok == null) {
3     return;
4   }
▶   System.out.print(kok.veri + " ");
6   kokBasta(kok.sol);
7   kokBasta(kok.sag);
8 }
```

ÇAĞRI YIĞINI

`kokBasta(9)`

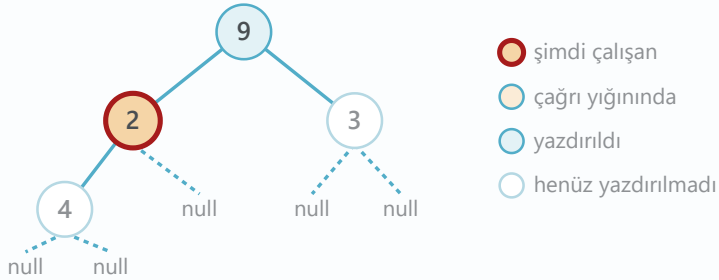
DEĞİŞKENLER

`kok = 9`

ÇIKTI

```
9
```

Özyinelemeli Kök Önde ve Çağrı Yığını



`kokBasta(2)` çağrıldı (9'un sol çocuğu). null değil → önce **kök yazdırılır: 2**, sonra alt ağaçlar.

Adım 2 / 13

```
1 public void kokBasta(AgacDugumu kok) {
2   if (kok == null) {
3     return;
4   }
▶   System.out.print(kok.veri + " ");
6   kokBasta(kok.sol);
7   kokBasta(kok.sag);
8 }
```

ÇAĞRI YIĞINI

- `kokBasta(2)`
- `kokBasta(9) · s.6`

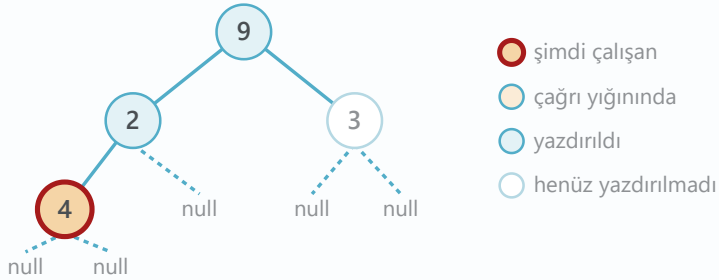
DEĞİŞKENLER

`kok = 2`

ÇIKTI

```
9 2
```

Özyinelemeli Kök Önde ve Çağrı Yığını



`kokBasta(4)` çağrıldı (2'nin sol çocuğu). null değil → önce **kök yazdırılır: 4**, sonra alt ağaçlar.

Adım 3 / 13

```
1 public void kokBasta(AgacDugumu kok) {
2   if (kok == null) {
3     return;
4   }
▶   System.out.print(kok.veri + " ");
6   kokBasta(kok.sol);
7   kokBasta(kok.sag);
8 }
```

ÇAĞRI YIĞINI

- `kokBasta(4)`
- `kokBasta(2) · s.6`
- `kokBasta(9) · s.6`

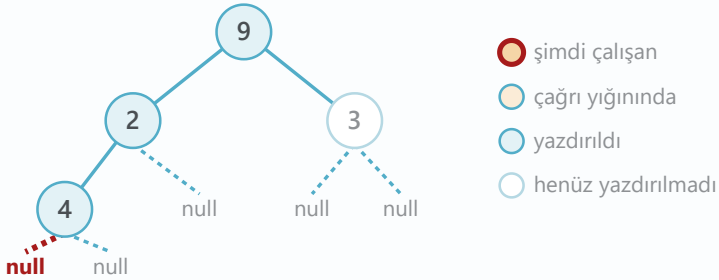
DEĞİŞKENLER

`kok = 4`

ÇIKTI

```
9 2 4
```

Özyinelemeli Kök Önde ve Çağrı Yığını



`kokBasta(4.sol)` = `kokBasta(null)` : **taban durum**, hemen **return**. Boş alt ağaçta yapılacak iş yok.

Adım 4 / 13

```
1 public void kokBasta(AgacDugumu kok) {
2   if (kok == null) {
▶     return;
4   }
5   System.out.print(kok.veri + " ");
6   kokBasta(kok.sol);
7   kokBasta(kok.sag);
8 }
```

ÇAĞRI YIĞINI

- `kokBasta(null)`
- `kokBasta(4) · s.6`
- `kokBasta(2) · s.6`
- `kokBasta(9) · s.6`

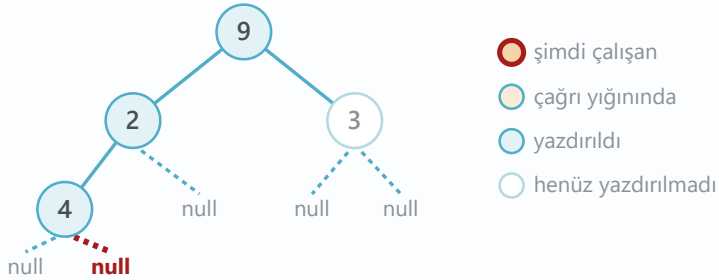
DEĞİŞKENLER

`kok = null`

ÇIKTI

```
9 2 4
```

Özyinelemeli Kök Önde ve Çağrı Yığını



`kokBasta(4.sag)` = `kokBasta(null)` : **taban durum**, hemen **return**. Boş alt ağaçta yapılacak iş yok.

Adım 5 / 13

```
1 public void kokBasta(AgacDugumu kok) {
2   if (kok == null) {
▶     return;
4   }
5   System.out.print(kok.veri + " ");
6   kokBasta(kok.sol);
7   kokBasta(kok.sag);
8 }
```

ÇAĞRI YIĞINI

- `kokBasta(null)`
- `kokBasta(4) · s.7`
- `kokBasta(2) · s.6`
- `kokBasta(9) · s.6`

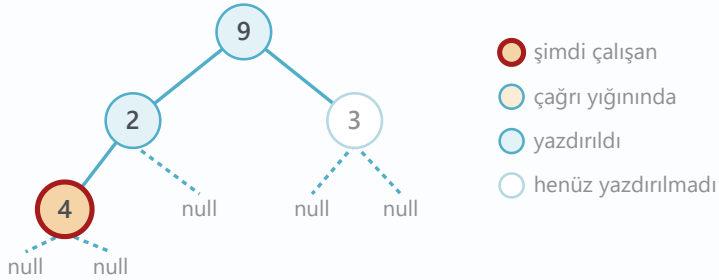
DEĞİŞKENLER

`kok = null`

ÇIKTI

```
9 2 4
```

Özyinelemeli Kök Önde ve Çağrı Yığını



`kokBasta(4)` bitti (satır 8); çerçeve yığından çıkar, `kokBasta(2)`'ye dönülür.

Adım 6 / 13

```
1 public void kokBasta(AgacDugumu kok) {
2   if (kok == null) {
3     return;
4   }
5   System.out.print(kok.veri + " ");
6   kokBasta(kok.sol);
7   kokBasta(kok.sag);
▶ }
```

ÇAĞRI YIĞINI

- `kokBasta(4)`
- `kokBasta(2) · s.6`
- `kokBasta(9) · s.6`

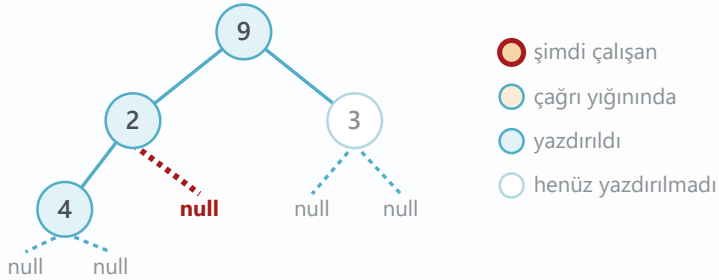
DEĞİŞKENLER

`kok = 4`

ÇIKTI

```
9 2 4
```

Özyinelemeli Kök Önde ve Çağrı Yığını



`kokBasta(2.sag)` = `kokBasta(null)` : **taban durum**, hemen **return**. Boş alt ağaçta yapılacak iş yok.

Adım 7 / 13

```
1 public void kokBasta(AgacDugumu kok) {
2   if (kok == null) {
▶     return;
4   }
5   System.out.print(kok.veri + " ");
6   kokBasta(kok.sol);
7   kokBasta(kok.sag);
8 }
```

ÇAĞRI YIĞINI

- `kokBasta(null)`
- `kokBasta(2) · s.7`
- `kokBasta(9) · s.6`

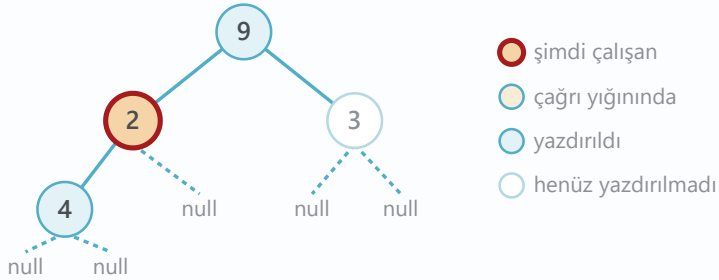
DEĞİŞKENLER

`kok = null`

ÇIKTI

```
9 2 4
```

Özyinelemeli Kök Önde ve Çağrı Yığını



`kokBasta(2)` bitti (satır 8); çerçeve yığından çıkar, `kokBasta(9)`'a dönülür.

Adım 8 / 13

```
1 public void kokBasta(AgacDugumu kok) {
2   if (kok == null) {
3     return;
4   }
5   System.out.print(kok.veri + " ");
6   kokBasta(kok.sol);
7   kokBasta(kok.sag);
▶ }
```

ÇAĞRI YIĞINI

- `kokBasta(2)`
- `kokBasta(9) · s.6`

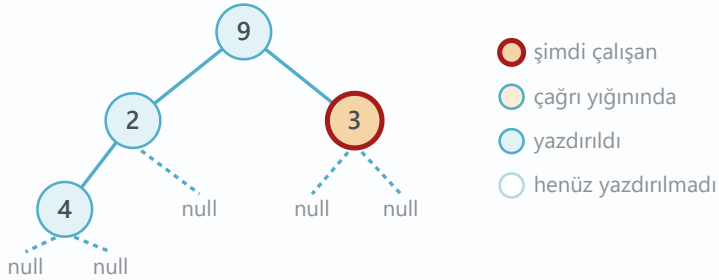
DEĞİŞKENLER

`kok = 2`

ÇIKTI

```
9 2 4
```

Özyinelemeli Kök Önde ve Çağrı Yığını



`kokBasta(3)` çağrıldı (9'un sağ çocuğu). null değil → önce **kök yazdırılır: 3**, sonra alt ağaçlar.

Adım 9 / 13

```
1 public void kokBasta(AgacDugumu kok) {
2   if (kok == null) {
3     return;
4   }
▶   System.out.print(kok.veri + " ");
6   kokBasta(kok.sol);
7   kokBasta(kok.sag);
8 }
```

ÇAĞRI YIĞINI

- `kokBasta(3)`
- `kokBasta(9) · s.7`

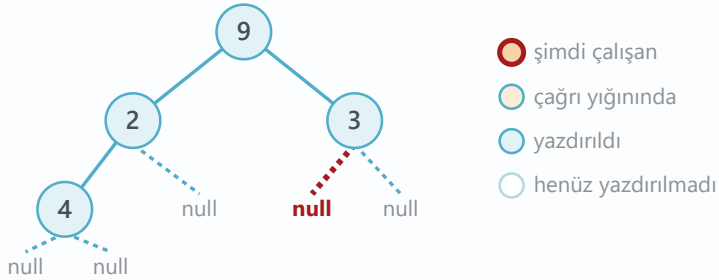
DEĞİŞKENLER

`kok = 3`

ÇIKTI

```
9 2 4 3
```

Özyinelemeli Kök Önde ve Çağrı Yığını



`kokBasta(3.sol)` = `kokBasta(null)` : **taban durum**, hemen **return**. Boş alt ağaçta yapılacak iş yok.

Adım 10 / 13

```
1 public void kokBasta(AgacDugumu kok) {
2   if (kok == null) {
▶     return;
4   }
5   System.out.print(kok.veri + " ");
6   kokBasta(kok.sol);
7   kokBasta(kok.sag);
8 }
```

ÇAĞRI YIĞINI

- `kokBasta(null)`
- `kokBasta(3) · s.6`
- `kokBasta(9) · s.7`

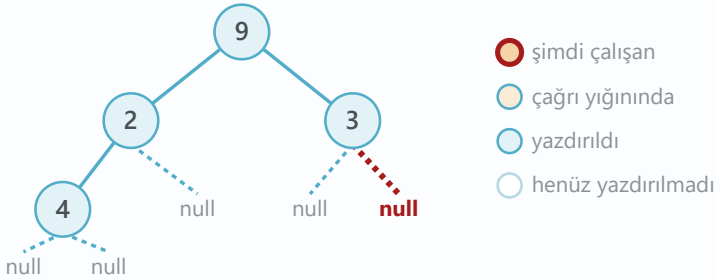
DEĞİŞKENLER

`kok = null`

ÇIKTI

```
9 2 4 3
```

Özyinelemeli Kök Önde ve Çağrı Yığını



`kokBasta(3.sag)` = `kokBasta(null)` : **taban durum**, hemen **return**. Boş alt ağaçta yapılacak iş yok.

Adım 11 / 13

```
1 public void kokBasta(AgacDugumu kok) {
2   if (kok == null) {
▶     return;
4   }
5   System.out.print(kok.veri + " ");
6   kokBasta(kok.sol);
7   kokBasta(kok.sag);
8 }
```

ÇAĞRI YIĞINI

- **`kokBasta(null)`**
- `kokBasta(3)` · s.7
- `kokBasta(9)` · s.7

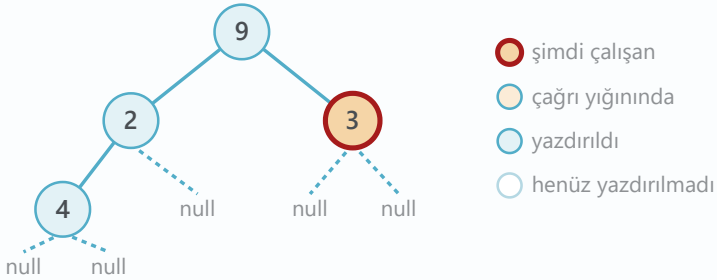
DEĞİŞKENLER

`kok` = **`null`**

ÇIKTI

```
9 2 4 3
```

Özyinelemeli Kök Önde ve Çağrı Yığını



`kokBasta(3)` bitti (satır 8); çerçeve yığından çıkar, `kokBasta(9)`'a dönülür.

Adım 12 / 13

```
1 public void kokBasta(AgacDugumu kok) {
2   if (kok == null) {
3     return;
4   }
5   System.out.print(kok.veri + " ");
6   kokBasta(kok.sol);
7   kokBasta(kok.sag);
▶ }
```

ÇAĞRI YIĞINI

- `kokBasta(3)`
- `kokBasta(9) · s.7`

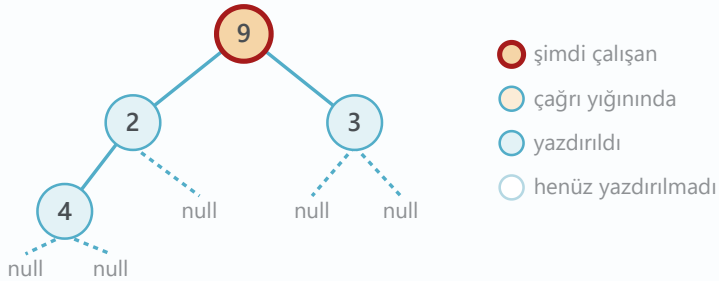
DEĞİŞKENLER

`kok = 3`

ÇIKTI

```
9 2 4 3
```

Özyinelemeli Kök Önde ve Çağrı Yığını



`kokBasta(9)` bitti (satır 8); yığın boşalır. **Çıktı: 9 2 4 3** — her düğüm bir kez, en fazla h + 2 = 4 çerçeve.

Adım 13 / 13

```
1 public void kokBasta(AgacDugumu kok) {
2   if (kok == null) {
3     return;
4   }
5   System.out.print(kok.veri + " ");
6   kokBasta(kok.sol);
7   kokBasta(kok.sag);
▶ }
```

ÇAĞRI YIĞINI

`kokBasta(9)`

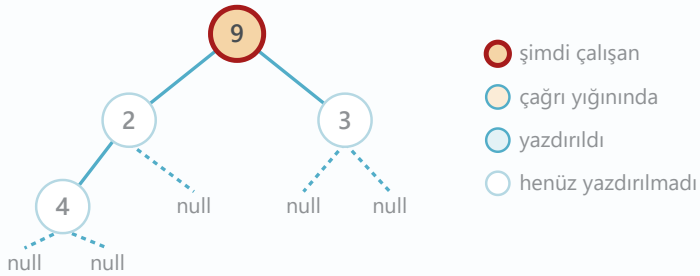
DEĞİŞKENLER

`kok = 9`

ÇIKTI

```
9 2 4 3
```

Özyinelemeli Kök Ortada ve Çağrı Yığını



`kokOrtada(9)` çağrıldı (kök). 9 henüz yazdırılmaz: **önce sol alt ağaç** gezilmeli.

Adım 1 / 17

```
1 public void kokOrtada(AgacDugumu kok)
  {
2   if (kok == null) {
3     return;
4   }
▶   kokOrtada(kok.sol);
6   System.out.print(kok.veri + " ");
7   kokOrtada(kok.sag);
8 }
```

ÇAĞRI YIĞINI

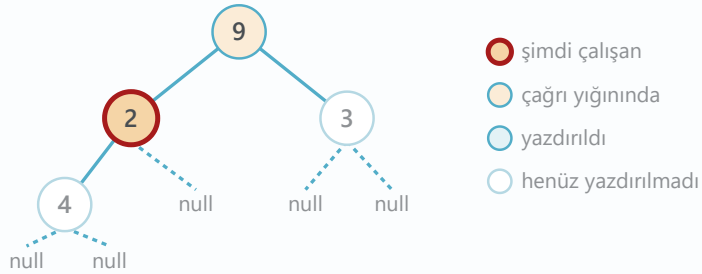
`kokOrtada(9)`

DEĞİŞKENLER

`kok = 9`

ÇIKTI

Özyinelemeli Kök Ortada ve Çağrı Yığını



`kokOrtada(2)` çağrıldı (9'un sol çocuğu). 2 henüz yazdırılmaz: **önce sol alt ağaç** gezilmeli.

Adım 2 / 17

```
1 public void kokOrtada(AgacDugumu kok)
  {
2   if (kok == null) {
3     return;
4   }
▶   kokOrtada(kok.sol);
6   System.out.print(kok.veri + " ");
7   kokOrtada(kok.sag);
8 }
```

ÇAĞRI YIĞINI

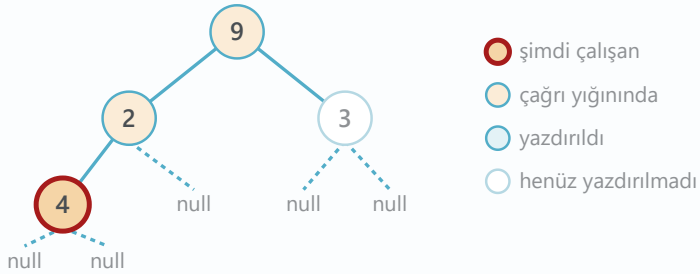
- `kokOrtada(2)`
- `kokOrtada(9) · s.5`

DEĞİŞKENLER

`kok = 2`

ÇIKTI

Özyinelemeli Kök Ortada ve Çağrı Yığını



`kokOrtada(4)` çağrıldı (2'nin sol çocuğu). 4 henüz yazdırılmaz: **önce sol alt ağaç** gezilmeli.

Adım 3 / 17

```
1 public void kokOrtada(AgacDugumu kok)
  {
2   if (kok == null) {
3     return;
4   }
    kokOrtada(kok.sol);
6   System.out.print(kok.veri + " ");
7   kokOrtada(kok.sag);
8 }
```

ÇAĞRI YIĞINI

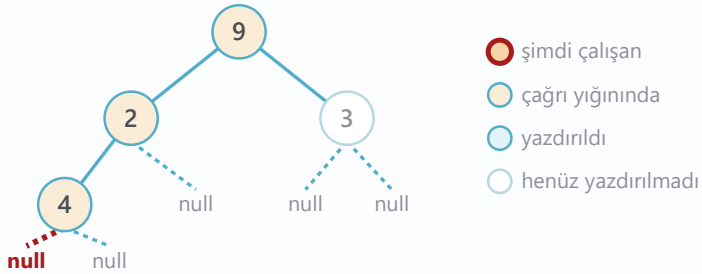
- `kokOrtada(4)`
- `kokOrtada(2) · s.5`
- `kokOrtada(9) · s.5`

DEĞİŞKENLER

`kok = 4`

ÇIKTI

Özyinelemeli Kök Ortada ve Çağrı Yığını



`kokOrtada(4.sol)` = `kokOrtada(null)` : **taban durum**, hemen **return**. Boş alt ağaçta yapılacak iş yok.

Adım 4 / 17

```
1 public void kokOrtada(AgacDugumu kok)
  {
2   if (kok == null) {
      return;
4   }
5   kokOrtada(kok.sol);
6   System.out.print(kok.veri + " ");
7   kokOrtada(kok.sag);
8 }
```

ÇAĞRI YIĞINI

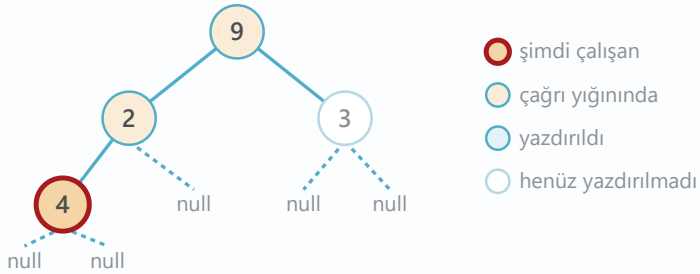
- `kokOrtada(null)`
- `kokOrtada(4) · s.5`
- `kokOrtada(2) · s.5`
- `kokOrtada(9) · s.5`

DEĞİŞKENLER

`kok = null`

ÇIKTI

Özyinelemeli Kök Ortada ve Çağrı Yığını



Sol alt ağaç bitti, `kokOrtada(4)`'e dönüldü → **4 yazdırılır**. Sırada sağ alt ağaç var.

Adım 5 / 17

```
1 public void kokOrtada(AgacDugumu kok)
  {
2   if (kok == null) {
3     return;
4   }
5   kokOrtada(kok.sol);
▶   System.out.print(kok.veri + " ");
7   kokOrtada(kok.sag);
8 }
```

ÇAĞRI YIĞINI

- `kokOrtada(4)`
- `kokOrtada(2) · s.5`
- `kokOrtada(9) · s.5`

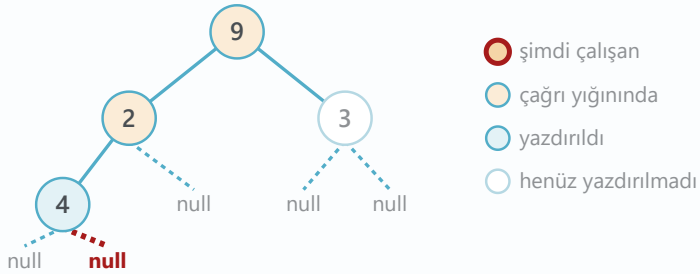
DEĞİŞKENLER

`kok = 4`

ÇIKTI

```
4
```

Özyinelemeli Kök Ortada ve Çağrı Yığını



`kokOrtada(4.sag)` = `kokOrtada(null)` : **taban durum**, hemen **return**. Boş alt ağaçta yapılacak iş yok.

Adım 6 / 17

```
1 public void kokOrtada(AgacDugumu kok)
  {
2   if (kok == null) {
▶       return;
4   }
5   kokOrtada(kok.sol);
6   System.out.print(kok.veri + " ");
7   kokOrtada(kok.sag);
8 }
```

ÇAĞRI YIĞINI

- `kokOrtada(null)`
- `kokOrtada(4) · s.7`
- `kokOrtada(2) · s.5`
- `kokOrtada(9) · s.5`

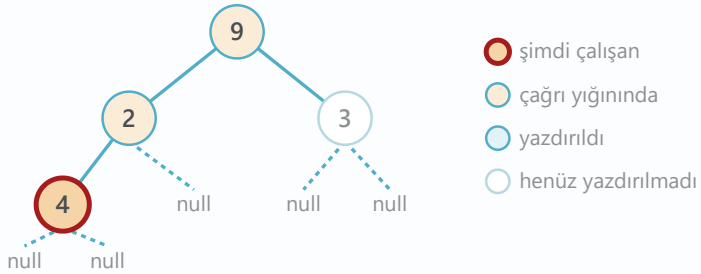
DEĞİŞKENLER

`kok = null`

ÇIKTI

```
4
```

Özyinelemeli Kök Ortada ve Çağrı Yığını



`kokOrtada(4)` bitti (satır 8); çerçeve yığından çıkar, `kokOrtada(2)`'ye dönülür.

Adım 7 / 17

```
1 public void kokOrtada(AgacDugumu kok)
  {
2   if (kok == null) {
3     return;
4   }
5   kokOrtada(kok.sol);
6   System.out.print(kok.veri + " ");
7   kokOrtada(kok.sag);
▶ }
```

ÇAĞRI YIĞINI

- `kokOrtada(4)`
- `kokOrtada(2) · s.5`
- `kokOrtada(9) · s.5`

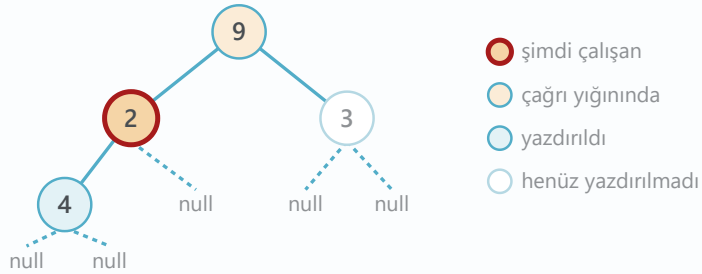
DEĞİŞKENLER

`kok = 4`

ÇIKTI

```
4
```

Özyinelemeli Kök Ortada ve Çağrı Yığını



Sol alt ağaç bitti, `kokOrtada(2)`'ye dönüldü → **2 yazdırılır**. Sırada sağ alt ağaç var.

Adım 8 / 17

```
1 public void kokOrtada(AgacDugumu kok)
  {
2   if (kok == null) {
3     return;
4   }
5   kokOrtada(kok.sol);
▶   System.out.print(kok.veri + " ");
7   kokOrtada(kok.sag);
8 }
```

ÇAĞRI YIĞINI

- `kokOrtada(2)`
- `kokOrtada(9) · s.5`

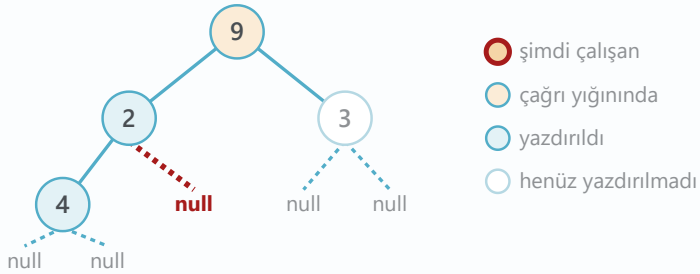
DEĞİŞKENLER

`kok = 2`

ÇIKTI

```
4 2
```

Özyinelemeli Kök Ortada ve Çağrı Yığını



`kokOrtada(2.sag)` = `kokOrtada(null)` : **taban durum**, hemen **return**. Boş alt ağaçta yapılacak iş yok.

Adım 9 / 17

```
1 public void kokOrtada(AgacDugumu kok)
  {
2   if (kok == null) {
▶       return;
4   }
5   kokOrtada(kok.sol);
6   System.out.print(kok.veri + " ");
7   kokOrtada(kok.sag);
8 }
```

ÇAĞRI YIĞINI

- `kokOrtada(null)`
- `kokOrtada(2) · s.7`
- `kokOrtada(9) · s.5`

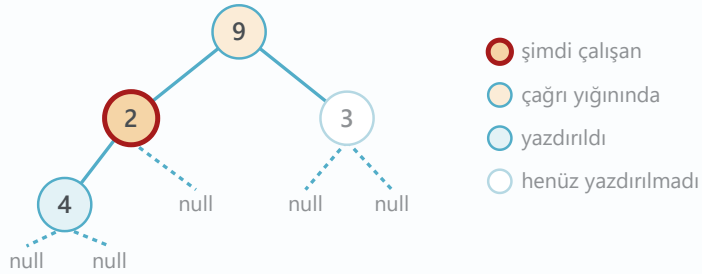
DEĞİŞKENLER

`kok = null`

ÇIKTI

```
4 2
```

Özyinelemeli Kök Ortada ve Çağrı Yığını



`kokOrtada(2)` bitti (satır 8); çerçeve yığından çıkar, `kokOrtada(9)` 'a dönülür.

Adım 10 / 17

```
1  public void kokOrtada(AgacDugumu kok)
   {
2    if (kok == null) {
3      return;
4    }
5    kokOrtada(kok.sol);
6    System.out.print(kok.veri + " ");
7    kokOrtada(kok.sag);
▶  }
```

ÇAĞRI YIĞINI

- `kokOrtada(2)`
- `kokOrtada(9) · s.5`

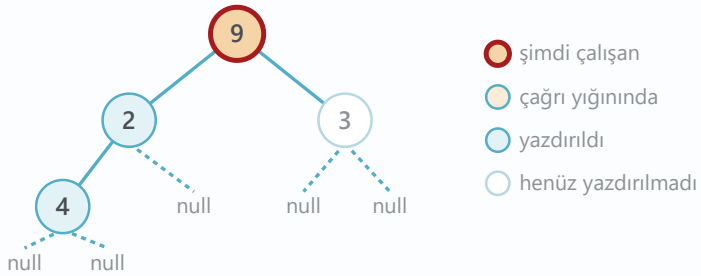
DEĞİŞKENLER

`kok = 2`

ÇIKTI

```
4 2
```

Özyinelemeli Kök Ortada ve Çağrı Yığını



Sol alt ağaç bitti, `kokOrtada(9)`'a dönüldü → **9 yazdırılır**. Sırada sağ alt ağaç var.

Adım 11 / 17

```
1 public void kokOrtada(AgacDugumu kok)
  {
2   if (kok == null) {
3     return;
4   }
5   kokOrtada(kok.sol);
▶   System.out.print(kok.veri + " ");
7   kokOrtada(kok.sag);
8 }
```

ÇAĞRI YIĞINI

`kokOrtada(9)`

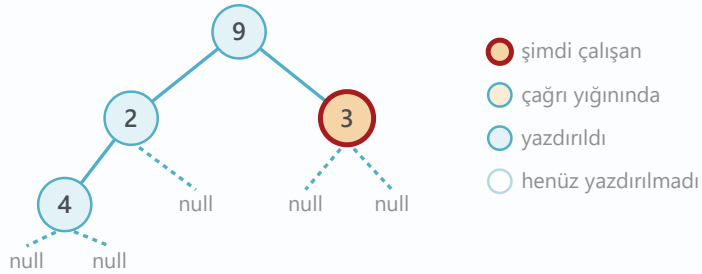
DEĞİŞKENLER

`kok = 9`

ÇIKTI

```
4 2 9
```

Özyinelemeli Kök Ortada ve Çağrı Yığını



`kokOrtada(3)` çağrıldı (9'un sağ çocuğu). 3 henüz yazdırılmaz: **önce sol alt ağaç** gezilmeli.

Adım 12 / 17

```
1 public void kokOrtada(AgacDugumu kok)
  {
2   if (kok == null) {
3     return;
4   }
▶   kokOrtada(kok.sol);
6   System.out.print(kok.veri + " ");
7   kokOrtada(kok.sag);
8 }
```

ÇAĞRI YIĞINI

- `kokOrtada(3)`
- `kokOrtada(9) · s.7`

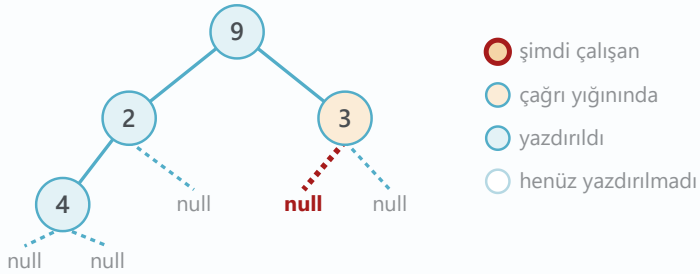
DEĞİŞKENLER

`kok = 3`

ÇIKTI

```
4 2 9
```

Özyinelemeli Kök Ortada ve Çağrı Yığını



`kokOrtada(3.sol)` = `kokOrtada(null)` : **taban durum**, hemen **return**. Boş alt ağaçta yapılacak iş yok.

Adım 13 / 17

```
1 public void kokOrtada(AgacDugumu kok)
  {
2   if (kok == null) {
▶       return;
4   }
5   kokOrtada(kok.sol);
6   System.out.print(kok.veri + " ");
7   kokOrtada(kok.sag);
8 }
```

ÇAĞRI YIĞINI

- `kokOrtada(null)`
- `kokOrtada(3) · s.5`
- `kokOrtada(9) · s.7`

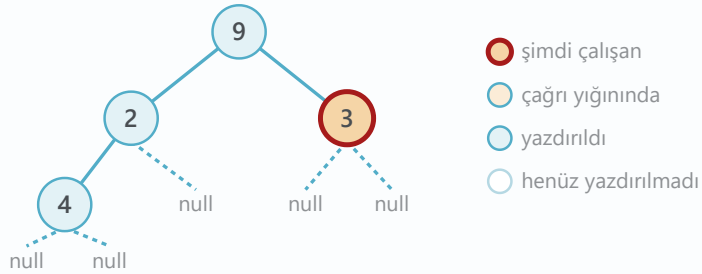
DEĞİŞKENLER

`kok = null`

ÇIKTI

```
4 2 9
```

Özyinelemeli Kök Ortada ve Çağrı Yığını



Sol alt ağaç bitti, `kokOrtada(3)` 'e dönüldü → **3 yazdırılır**. Sırada sağ alt ağaç var.

Adım 14 / 17

```
1 public void kokOrtada(AgacDugumu kok)
  {
2   if (kok == null) {
3     return;
4   }
5   kokOrtada(kok.sol);
▶   System.out.print(kok.veri + " ");
7   kokOrtada(kok.sag);
8 }
```

ÇAĞRI YIĞINI

- `kokOrtada(3)`
- `kokOrtada(9) · s.7`

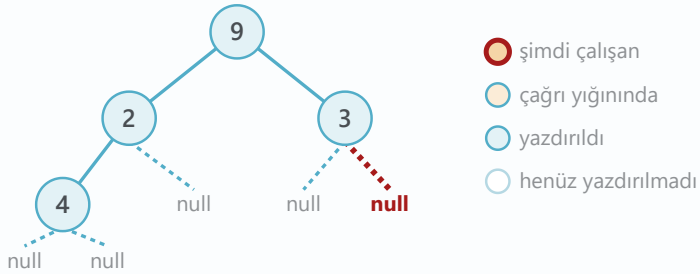
DEĞİŞKENLER

`kok = 3`

ÇIKTI

```
4 2 9 3
```

Özyinelemeli Kök Ortada ve Çağrı Yığını



`kokOrtada(3.sag)` = `kokOrtada(null)` : **taban durum**, hemen **return**. Boş alt ağaçta yapılacak iş yok.

Adım 15 / 17

```
1 public void kokOrtada(AgacDugumu kok)
  {
2   if (kok == null) {
▶       return;
4   }
5   kokOrtada(kok.sol);
6   System.out.print(kok.veri + " ");
7   kokOrtada(kok.sag);
8 }
```

ÇAĞRI YIĞINI

- **`kokOrtada(null)`**
- `kokOrtada(3) · s.7`
- `kokOrtada(9) · s.7`

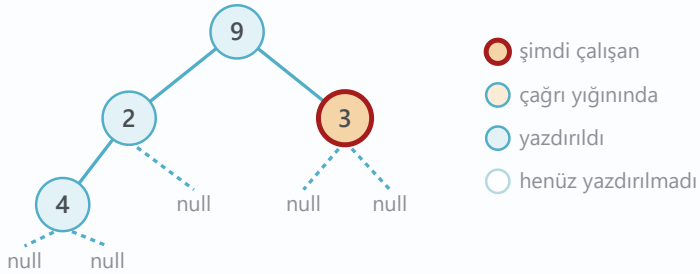
DEĞİŞKENLER

`kok = null`

ÇIKTI

```
4 2 9 3
```

Özyinelemeli Kök Ortada ve Çağrı Yığını



`kokOrtada(3)` bitti (satır 8); çerçeve yığından çıkar, `kokOrtada(9)` 'a dönülür.

Adım 16 / 17

```
1 public void kokOrtada(AgacDugumu kok)
  {
2   if (kok == null) {
3     return;
4   }
5   kokOrtada(kok.sol);
6   System.out.print(kok.veri + " ");
7   kokOrtada(kok.sag);
▶ }
```

ÇAĞRI YIĞINI

- `kokOrtada(3)`
- `kokOrtada(9) · s.7`

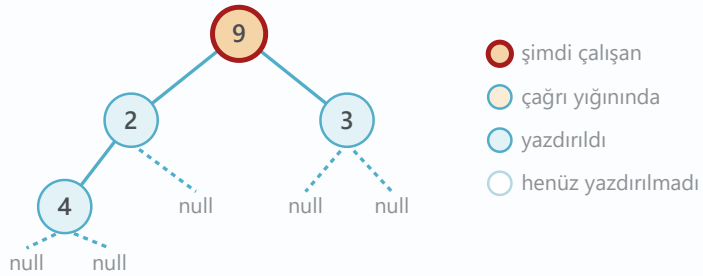
DEĞİŞKENLER

`kok = 3`

ÇIKTI

```
4 2 9 3
```

Özyinelemeli Kök Ortada ve Çağrı Yığını



`kokOrtada(9)` bitti (satır 8); yığın boşalır. **Çıktı: 4 2 9 3** — her düğüm bir kez, en fazla h + 2 = 4 çerçeve.

Adım 17 / 17

```
1 public void kokOrtada(AgacDugumu kok)
  {
2   if (kok == null) {
3     return;
4   }
5   kokOrtada(kok.sol);
6   System.out.print(kok.veri + " ");
7   kokOrtada(kok.sag);
▶ }
```

ÇAĞRI YIĞINI

`kokOrtada(9)`

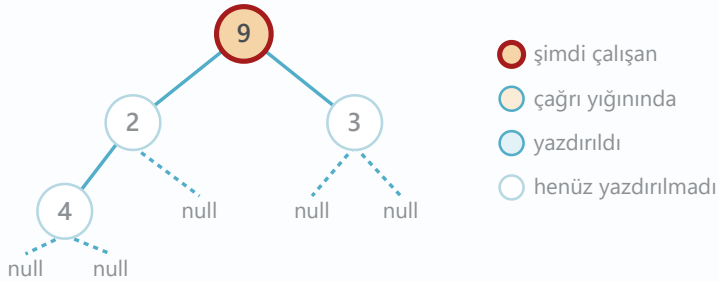
DEĞİŞKENLER

`kok = 9`

ÇIKTI

```
4 2 9 3
```

Özyinelemeli Kök Sonda ve Çağrı Yığını



`kokSonda(9)` çağrıldı (kök). Kök en sona kalır: **önce sol alt ağaç**.

Adım 1 / 17

```
1 public void kokSonda(AgacDugumu kok) {
2   if (kok == null) {
3     return;
4   }
▶   kokSonda(kok.sol);
6   kokSonda(kok.sag);
7   System.out.print(kok.veri + " ");
8 }
```

ÇAĞRI YIĞINI

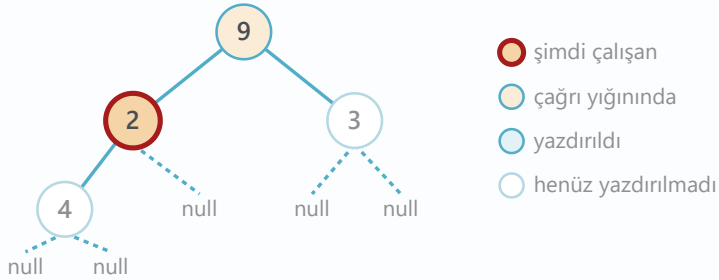
`kokSonda(9)`

DEĞİŞKENLER

`kok = 9`

ÇIKTI

Özyinelemeli Kök Sonda ve Çağrı Yığını



`kokSonda(2)` çağrıldı (9'un sol çocuğu). Kök en sona kalır: **önce sol alt ağaç**.

Adım 2 / 17

```
1 public void kokSonda(AgacDugumu kok) {
2   if (kok == null) {
3     return;
4   }
▶   kokSonda(kok.sol);
6   kokSonda(kok.sag);
7   System.out.print(kok.veri + " ");
8 }
```

ÇAĞRI YIĞINI

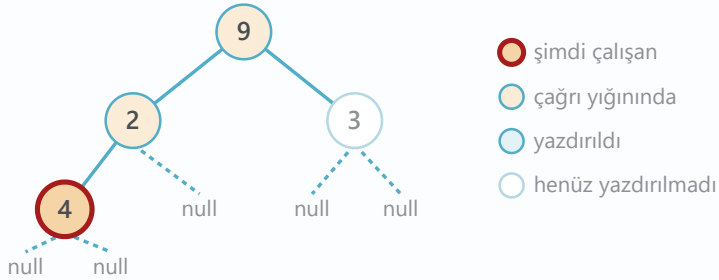
- `kokSonda(2)`
- `kokSonda(9) · s.5`

DEĞİŞKENLER

`kok = 2`

ÇIKTI

Özyinelemeli Kök Sonda ve Çağrı Yığını



`kokSonda(4)` çağrıldı (2'nin sol çocuğu). Kök en sona kalır: **önce sol alt ağaç**.

Adım 3 / 17

```
1 public void kokSonda(AgacDugumu kok) {
2   if (kok == null) {
3     return;
4   }
▶   kokSonda(kok.sol);
6   kokSonda(kok.sag);
7   System.out.print(kok.veri + " ");
8 }
```

ÇAĞRI YIĞINI

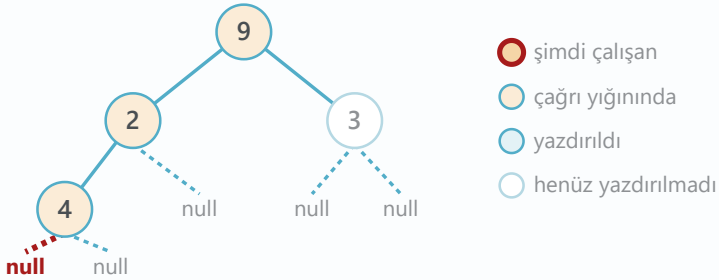
- `kokSonda(4)`
- `kokSonda(2) · s.5`
- `kokSonda(9) · s.5`

DEĞİŞKENLER

`kok = 4`

ÇIKTI

Özyinelemeli Kök Sonda ve Çağrı Yığını



`kokSonda(4.sol)` = `kokSonda(null)`: **taban durum**, hemen **return**. Boş alt ağaçta yapılacak iş yok.

Adım 4 / 17

```
1 public void kokSonda(AgacDugumu kok) {
2   if (kok == null) {
▶     return;
4   }
5   kokSonda(kok.sol);
6   kokSonda(kok.sag);
7   System.out.print(kok.veri + " ");
8 }
```

ÇAĞRI YIĞINI

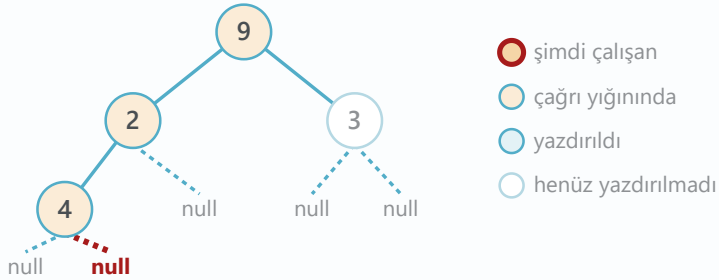
- `kokSonda(null)`
- `kokSonda(4) · s.5`
- `kokSonda(2) · s.5`
- `kokSonda(9) · s.5`

DEĞİŞKENLER

`kok = null`

ÇIKTI

Özyinelemeli Kök Sonda ve Çağrı Yığını



`kokSonda(4.sag)` = `kokSonda(null)` : **taban durum**, hemen **return**. Boş alt ağaçta yapılacak iş yok.

Adım 5 / 17

```
1 public void kokSonda(AgacDugumu kok) {
2   if (kok == null) {
▶     return;
4   }
5   kokSonda(kok.sol);
6   kokSonda(kok.sag);
7   System.out.print(kok.veri + " ");
8 }
```

ÇAĞRI YIĞINI

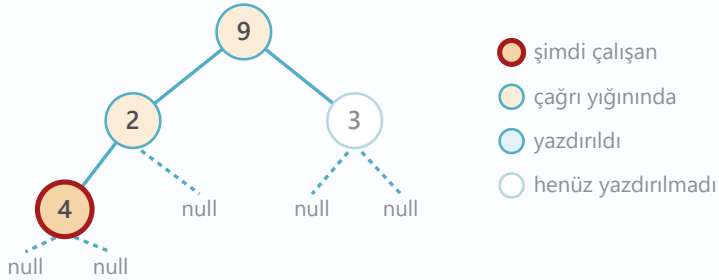
- `kokSonda(null)`
- `kokSonda(4) · s.6`
- `kokSonda(2) · s.5`
- `kokSonda(9) · s.5`

DEĞİŞKENLER

`kok = null`

ÇIKTI

Özyinelemeli Kök Sonda ve Çağrı Yığını



`kokSonda(4)`: iki alt ağaç da bitti → şimdi **4 yazdırılır**.

Adım 6 / 17

```
1 public void kokSonda(AgacDugumu kok) {
2   if (kok == null) {
3     return;
4   }
5   kokSonda(kok.sol);
6   kokSonda(kok.sag);
▶   System.out.print(kok.veri + " ");
8 }
```

ÇAĞRI YIĞINI

- `kokSonda(4)`
- `kokSonda(2) · s.5`
- `kokSonda(9) · s.5`

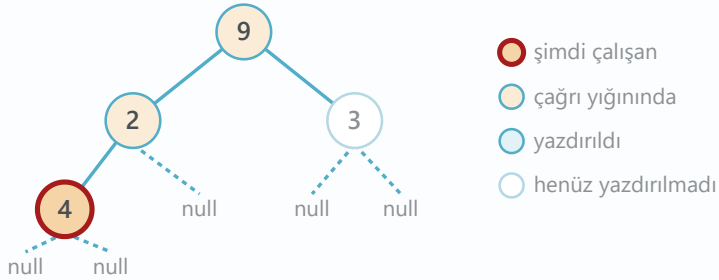
DEĞİŞKENLER

`kok = 4`

ÇIKTI

```
4
```

Özyinelemeli Kök Sonda ve Çağrı Yığını



`kokSonda(4)` bitti (satır 8); çerçeve yığından çıkar, `kokSonda(2)`'ye dönülür.

Adım 7 / 17

```
1 public void kokSonda(AgacDugumu kok) {
2   if (kok == null) {
3     return;
4   }
5   kokSonda(kok.sol);
6   kokSonda(kok.sag);
7   System.out.print(kok.veri + " ");
▶ }
```

ÇAĞRI YIĞINI

- `kokSonda(4)`
- `kokSonda(2) · s.5`
- `kokSonda(9) · s.5`

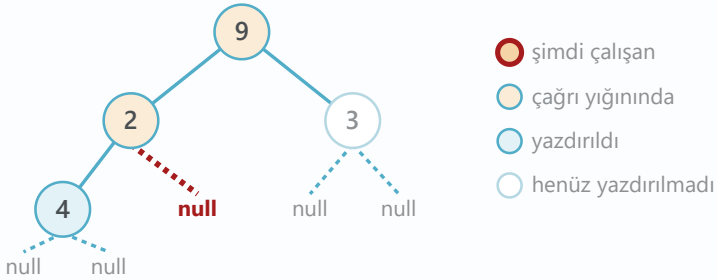
DEĞİŞKENLER

`kok = 4`

ÇIKTI

```
4
```

Özyinelemeli Kök Sonda ve Çağrı Yığını



`kokSonda(2.sag)` = `kokSonda(null)` : **taban durum**, hemen **return**. Boş alt ağaçta yapılacak iş yok.

Adım 8 / 17

```
1 public void kokSonda(AgacDugumu kok) {
2   if (kok == null) {
▶     return;
4   }
5   kokSonda(kok.sol);
6   kokSonda(kok.sag);
7   System.out.print(kok.veri + " ");
8 }
```

ÇAĞRI YIĞINI

- `kokSonda(null)`
- `kokSonda(2) · s.6`
- `kokSonda(9) · s.5`

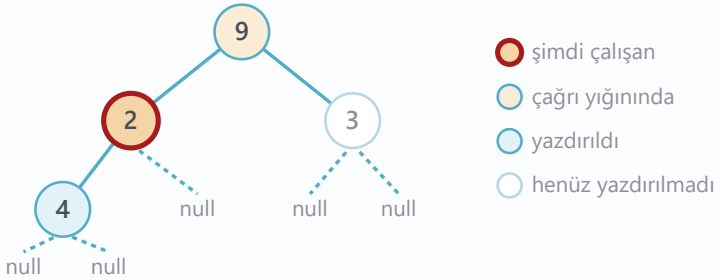
DEĞİŞKENLER

`kok = null`

ÇIKTI

```
4
```

Özyinelemeli Kök Sonda ve Çağrı Yığını



`kokSonda(2)`: iki alt ağaç da bitti → şimdi **2 yazdırılır**.

Adım 9 / 17

```
1 public void kokSonda(AgacDugumu kok) {
2   if (kok == null) {
3     return;
4   }
5   kokSonda(kok.sol);
6   kokSonda(kok.sag);
▶   System.out.print(kok.veri + " ");
8 }
```

ÇAĞRI YIĞINI

- `kokSonda(2)`
- `kokSonda(9) · s.5`

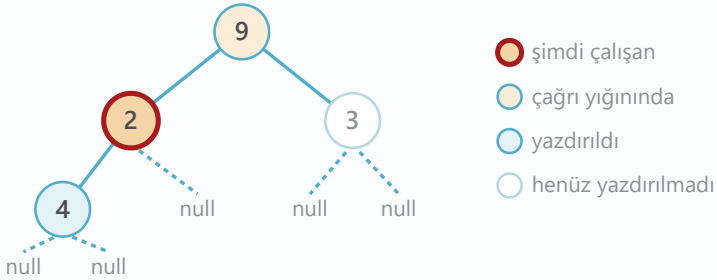
DEĞİŞKENLER

`kok = 2`

ÇIKTI

```
4 2
```

Özyinelemeli Kök Sonda ve Çağrı Yığını



`kokSonda(2)` bitti (satır 8); çerçeve yığından çıkar, `kokSonda(9)`'a dönülür.

Adım 10 / 17

```
1 public void kokSonda(AgacDugumu kok) {
2   if (kok == null) {
3     return;
4   }
5   kokSonda(kok.sol);
6   kokSonda(kok.sag);
7   System.out.print(kok.veri + " ");
▶ }
```

ÇAĞRI YIĞINI

- `kokSonda(2)`
- `kokSonda(9) · s.5`

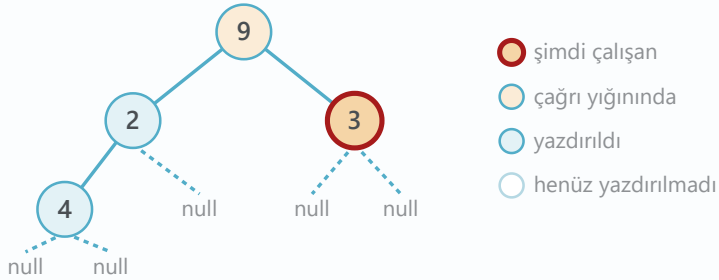
DEĞİŞKENLER

`kok = 2`

ÇIKTI

```
4 2
```

Özyinelemeli Kök Sonda ve Çağrı Yığını



`kokSonda(3)` çağrıldı (9'un sağ çocuğu). Kök en sona kalır: **önce sol alt ağaç**.

Adım 11 / 17

```
1 public void kokSonda(AgacDugumu kok) {
2   if (kok == null) {
3     return;
4   }
▶   kokSonda(kok.sol);
6   kokSonda(kok.sag);
7   System.out.print(kok.veri + " ");
8 }
```

ÇAĞRI YIĞINI

- **kokSonda(3)**
- kokSonda(9) · s.6

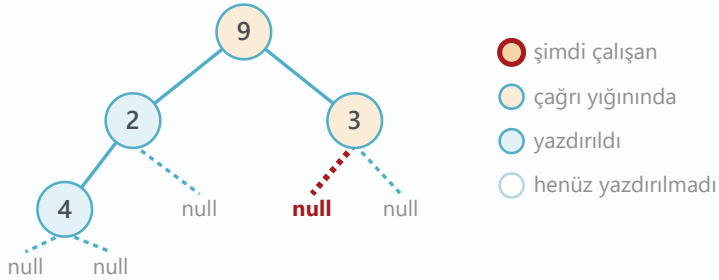
DEĞİŞKENLER

kok = **3**

ÇIKTI

```
4 2
```

Özyinelemeli Kök Sonda ve Çağrı Yığını



`kokSonda(3.sol)` = `kokSonda(null)`: **taban durum**, hemen **return**. Boş alt ağaçta yapılacak iş yok.

Adım 12 / 17

```
1 public void kokSonda(AgacDugumu kok) {
2   if (kok == null) {
▶     return;
4   }
5   kokSonda(kok.sol);
6   kokSonda(kok.sag);
7   System.out.print(kok.veri + " ");
8 }
```

ÇAĞRI YIĞINI

- `kokSonda(null)`
- `kokSonda(3) · s.5`
- `kokSonda(9) · s.6`

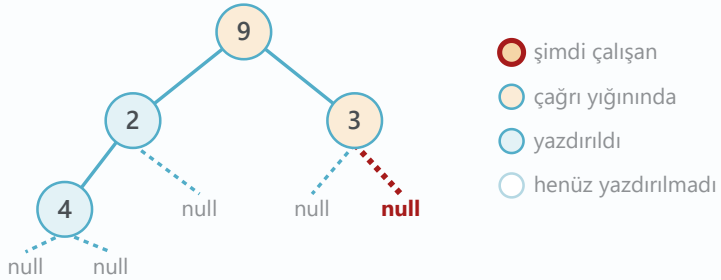
DEĞİŞKENLER

`kok = null`

ÇIKTI

```
4 2
```

Özyinelemeli Kök Sonda ve Çağrı Yığını



`kokSonda(3.sag)` = `kokSonda(null)` : **taban durum**, hemen **return**. Boş alt ağaçta yapılacak iş yok.

Adım 13 / 17

```
1 public void kokSonda(AgacDugumu kok) {
2   if (kok == null) {
▶     return;
4   }
5   kokSonda(kok.sol);
6   kokSonda(kok.sag);
7   System.out.print(kok.veri + " ");
8 }
```

ÇAĞRI YIĞINI

- **kokSonda(null)**
- kokSonda(3) · s.6
- kokSonda(9) · s.6

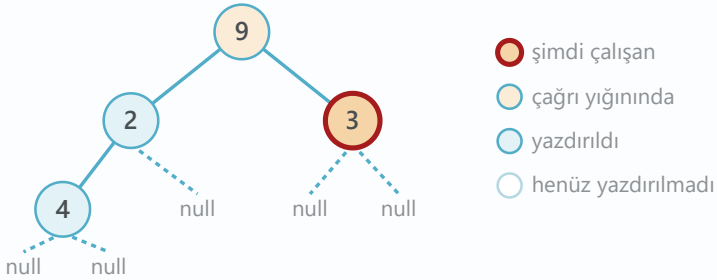
DEĞİŞKENLER

kok = **null**

ÇIKTI

```
4 2
```

Özyinelemeli Kök Sonda ve Çağrı Yığını



`kokSonda(3)` : iki alt ağaç da bitti → şimdi **3 yazdırılır**.

Adım 14 / 17

```
1 public void kokSonda(AgacDugumu kok) {
2   if (kok == null) {
3     return;
4   }
5   kokSonda(kok.sol);
6   kokSonda(kok.sag);
▶   System.out.print(kok.veri + " ");
8 }
```

ÇAĞRI YIĞINI

- `kokSonda(3)`
- `kokSonda(9) · s.6`

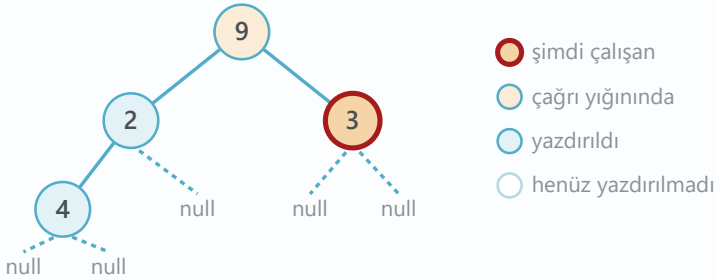
DEĞİŞKENLER

`kok = 3`

ÇIKTI

```
4 2 3
```

Özyinelemeli Kök Sonda ve Çağrı Yığını



`kokSonda(3)` bitti (satır 8); çerçeve yığından çıkar, `kokSonda(9)`'a dönülür.

Adım 15 / 17

```
1 public void kokSonda(AgacDugumu kok) {
2   if (kok == null) {
3     return;
4   }
5   kokSonda(kok.sol);
6   kokSonda(kok.sag);
7   System.out.print(kok.veri + " ");
▶ }
```

ÇAĞRI YIĞINI

- **`kokSonda(3)`**
- `kokSonda(9) · s.6`

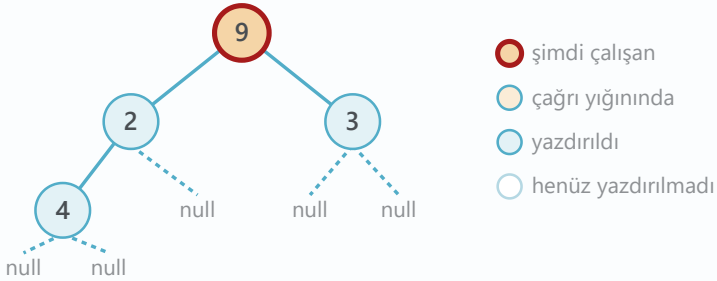
DEĞİŞKENLER

`kok = 3`

ÇIKTI

```
4 2 3
```

Özyinelemeli Kök Sonda ve Çağrı Yığını



`kokSonda(9)`: iki alt ağaç da bitti → şimdi **9 yazdırılır**.

Adım 16 / 17

```
1 public void kokSonda(AgacDugumu kok) {
2   if (kok == null) {
3     return;
4   }
5   kokSonda(kok.sol);
6   kokSonda(kok.sag);
▶   System.out.print(kok.veri + " ");
8 }
```

ÇAĞRI YIĞINI

`kokSonda(9)`

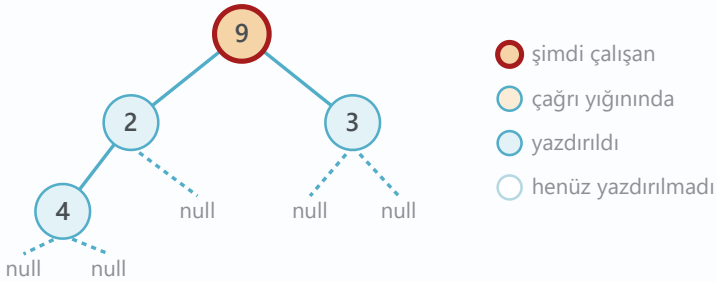
DEĞİŞKENLER

`kok = 9`

ÇIKTI

```
4 2 3 9
```

Özyinelemeli Kök Sonda ve Çağrı Yığını



`kokSonda(9)` bitti (satır 8); yığın boşalır. **Çıktı: 4 2 3 9** — her düğüm bir kez, en fazla h + 2 = 4 çerçeve.

Adım 17 / 17

```
1 public void kokSonda(AgacDugumu kok) {
2   if (kok == null) {
3     return;
4   }
5   kokSonda(kok.sol);
6   kokSonda(kok.sag);
7   System.out.print(kok.veri + " ");
▶ }
```

ÇAĞRI YIĞINI

`kokSonda(9)`

DEĞİŞKENLER

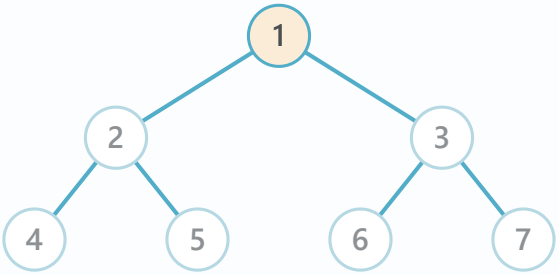
`kok = 9`

ÇIKTI

```
4 2 3 9
```

Yığın ile Kök Önde Gezinme

```
1  public void kokBastaYigin(AgacDugumu kok) {
2    if (kok == null) return;
3    Deque<AgacDugumu> yigin = new ArrayDeque<>
   ();
▶    yigin.push(kok);
5    while (!yigin.isEmpty()) {
6      AgacDugumu gecici = yigin.pop();
7      System.out.print(gecici.veri + " ");
8      if (gecici.sag != null)
9        yigin.push(gecici.sag);  // önce sağ
10     if (gecici.sol != null)
11       yigin.push(gecici.sol);  // sol üstte
12   }
13 }
```



şimdi · yığında · yazdırıldı

Kök yığına itilir. Yığın, özyinelemedeki çağrı yığınının yerini tutar.

Adım 1 / 15

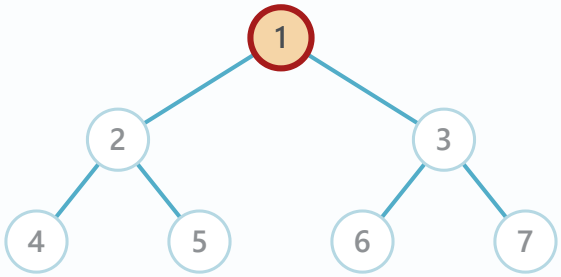
DEĞİŞKENLER

`yigin = [1]` `gecici = –`

ÇIKTI

Yığın ile Kök Önde Gezinme

```
1  public void kokBastaYigin(AgacDugumu kok) {
2    if (kok == null) return;
3    Deque<AgacDugumu> yigin = new ArrayDeque<>
   ();
4    yigin.push(kok);
5    while (!yigin.isEmpty()) {
       AgacDugumu gecici = yigin.pop();
       System.out.print(gecici.veri + " ");
8      if (gecici.sag != null)
9        yigin.push(gecici.sag);  // önce sağ
10     if (gecici.sol != null)
11       yigin.push(gecici.sol);  // sol üstte
12   }
13 }
```



Yığın boş

şimdi yığında yazdırıldı

`pop()` üstteki **1**'i verir; hemen **yazdırılır** (kök önde: önce kök).

Adım 2 / 15

DEĞİŞKENLER

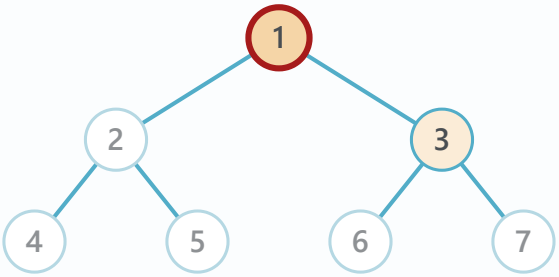
`yigin = []` `gecici = 1`

ÇIKTI

```
1
```

Yığın ile Kök Önde Gezinme

```
1  public void kokBastaYigin(AgacDugumu kok) {
2    if (kok == null) return;
3    Deque<AgacDugumu> yigin = new ArrayDeque<>
   ();
4    yigin.push(kok);
5    while (!yigin.isEmpty()) {
6      AgacDugumu gecici = yigin.pop();
7      System.out.print(gecici.veri + " ");
8      if (gecici.sag != null)
▶        yigin.push(gecici.sag);  // önce sağ
10     if (gecici.sol != null)
11       yigin.push(gecici.sol);  // sol üstte
12   }
13 }
```



şimdi yığında yazdırıldı

Sağ çocuk **3** itilir. Önce sağ itilir ki sol üstte kalıp **önce işlensin** (LIFO).

Adım 3 / 15

DEĞİŞKENLER

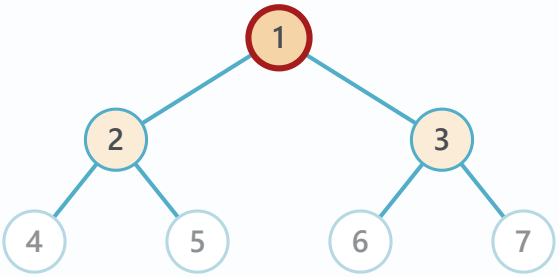
`yigin = [3]` `gecici = 1`

ÇIKTI

```
1
```

Yığın ile Kök Önde Gezinme

```
1  public void kokBastaYigin(AgacDugumu kok) {
2    if (kok == null) return;
3    Deque<AgacDugumu> yigin = new ArrayDeque<>
   ();
4    yigin.push(kok);
5    while (!yigin.isEmpty()) {
6      AgacDugumu gecici = yigin.pop();
7      System.out.print(gecici.veri + " ");
8      if (gecici.sag != null)
9        yigin.push(gecici.sag);  // önce sağ
10     if (gecici.sol != null)
▶        yigin.push(gecici.sol);  // sol üstte
12   }
13 }
```



şimdi · yığında · yazdırıldı

Sol çocuk **2** itilir → yığının üstünde; bir sonraki `pop()` onu verir.

Adım 4 / 15

DEĞİŞKENLER

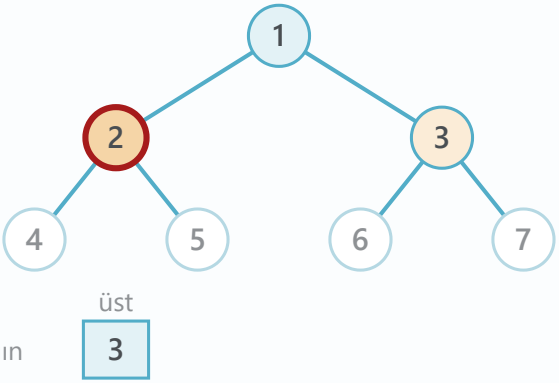
`yigin = [3, 2]` `gecici = 1`

ÇIKTI

```
1
```

Yığın ile Kök Önde Gezinme

```
1  public void kokBastaYigin(AgacDugumu kok) {
2    if (kok == null) return;
3    Deque<AgacDugumu> yigin = new ArrayDeque<>
   ();
4    yigin.push(kok);
5    while (!yigin.isEmpty()) {
       AgacDugumu gecici = yigin.pop();
       System.out.print(gecici.veri + " ");
8      if (gecici.sag != null)
9        yigin.push(gecici.sag);  // önce sağ
10     if (gecici.sol != null)
11       yigin.push(gecici.sol);  // sol üstte
12   }
13 }
```



şimdi yığında yazdırıldı

`pop()` üstteki **2**'yi verir; hemen **yazdırılır** (kök önde: önce kök).

Adım 5 / 15

DEĞİŞKENLER

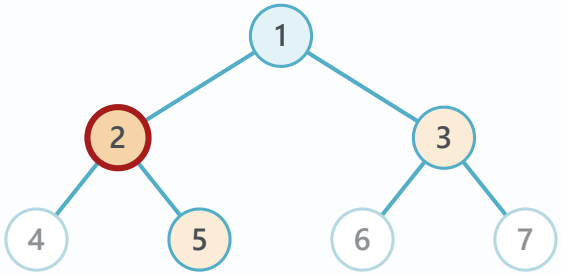
`yigin = [3]` `gecici = 2`

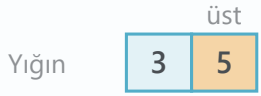
ÇIKTI

```
1 2
```

Yığın ile Kök Önde Gezinme

```
1  public void kokBastaYigin(AgacDugumu kok) {
2    if (kok == null) return;
3    Deque<AgacDugumu> yigin = new ArrayDeque<>
   ();
4    yigin.push(kok);
5    while (!yigin.isEmpty()) {
6      AgacDugumu gecici = yigin.pop();
7      System.out.print(gecici.veri + " ");
8      if (gecici.sag != null)
▶        yigin.push(gecici.sag);  // önce sağ
10     if (gecici.sol != null)
11       yigin.push(gecici.sol);  // sol üstte
12   }
13 }
```





şimdi yığında yazdırıldı

Sağ çocuk **5** itilir. Önce sağ itilir ki sol üstte kalıp **önce işlensin** (LIFO).

Adım 6 / 15

DEĞİŞKENLER

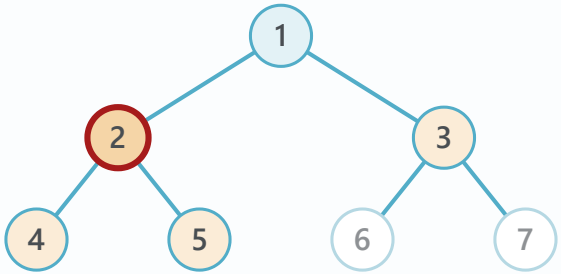
`yigin = [3, 5]` `gecici = 2`

ÇIKTI

```
1 2
```

Yığın ile Kök Önde Gezinme

```
1  public void kokBastaYigin(AgacDugumu kok) {
2    if (kok == null) return;
3    Deque<AgacDugumu> yigin = new ArrayDeque<>
   ();
4    yigin.push(kok);
5    while (!yigin.isEmpty()) {
6      AgacDugumu gecici = yigin.pop();
7      System.out.print(gecici.veri + " ");
8      if (gecici.sag != null)
9        yigin.push(gecici.sag);  // önce sağ
10     if (gecici.sol != null)
▶        yigin.push(gecici.sol);  // sol üstte
12   }
13 }
```



şimdi yığında yazdırıldı

Sol çocuk **4** itilir → yığının üstünde; bir sonraki `pop()` onu verir.

Adım 7 / 15

DEĞİŞKENLER

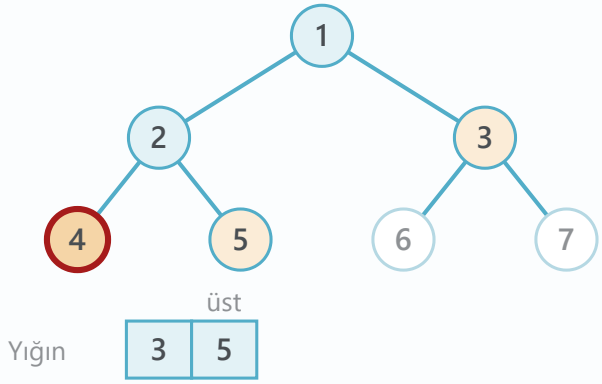
`yigin = [3, 5, 4]` `gecici = 2`

ÇIKTI

```
1 2
```

Yığın ile Kök Önde Gezinme

```
1  public void kokBastaYigin(AgacDugumu kok) {
2    if (kok == null) return;
3    Deque<AgacDugumu> yigin = new ArrayDeque<>
   ();
4    yigin.push(kok);
5    while (!yigin.isEmpty()) {
▶      AgacDugumu gecici = yigin.pop();
▶      System.out.print(gecici.veri + " ");
8      if (gecici.sag != null)
9        yigin.push(gecici.sag);  // önce sağ
10     if (gecici.sol != null)
11       yigin.push(gecici.sol);  // sol üstte
12   }
13 }
```



`pop()` üstteki **4**'ü verir; hemen **yazdırılır** (kök önde: önce kök). 4 yaprak: itilecek çocuk yok.

Adım 8 / 15

DEĞİŞKENLER

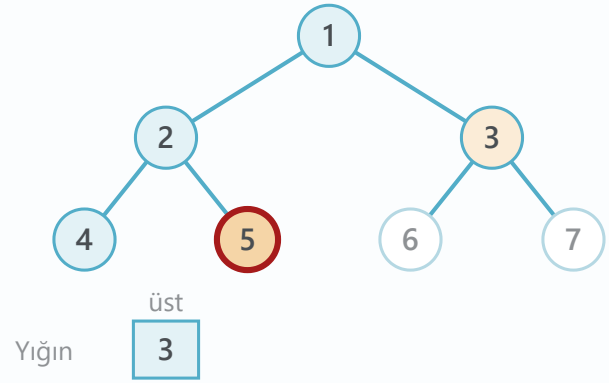
`yigin = [3, 5]` `gecici = 4`

ÇIKTI

```
1 2 4
```

Yığın ile Kök Önde Gezinme

```
1  public void kokBastaYigin(AgacDugumu kok) {
2    if (kok == null) return;
3    Deque<AgacDugumu> yigin = new ArrayDeque<>
   ();
4    yigin.push(kok);
5    while (!yigin.isEmpty()) {
▶      AgacDugumu gecici = yigin.pop();
▶      System.out.print(gecici.veri + " ");
8      if (gecici.sag != null)
9        yigin.push(gecici.sag);  // önce sağ
10     if (gecici.sol != null)
11       yigin.push(gecici.sol);  // sol üstte
12   }
13 }
```



`pop()` üstteki **5**'i verir; hemen **yazdırılır** (kök önde: önce kök). 5 yaprak: itilecek çocuk yok.

Adım 9 / 15

DEĞİŞKENLER

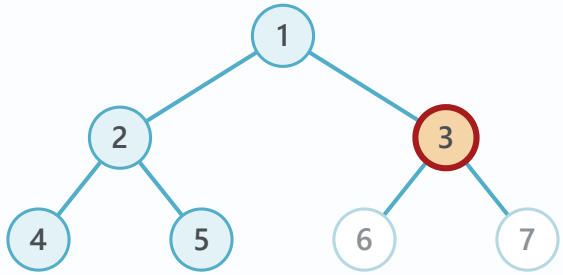
`yigin = [3]` `gecici = 5`

ÇIKTI

```
1 2 4 5
```

Yığın ile Kök Önde Gezinme

```
1  public void kokBastaYigin(AgacDugumu kok) {
2    if (kok == null) return;
3    Deque<AgacDugumu> yigin = new ArrayDeque<>
   ();
4    yigin.push(kok);
5    while (!yigin.isEmpty()) {
       AgacDugumu gecici = yigin.pop();
       System.out.print(gecici.veri + " ");
8      if (gecici.sag != null)
9        yigin.push(gecici.sag);  // önce sağ
10     if (gecici.sol != null)
11       yigin.push(gecici.sol);  // sol üstte
12   }
13 }
```



Yığın boş

şimdi yığında yazdırıldı

`pop()` üstteki **3**'ü verir; hemen **yazdırılır** (kök önde: önce kök).

Adım 10 / 15

DEĞİŞKENLER

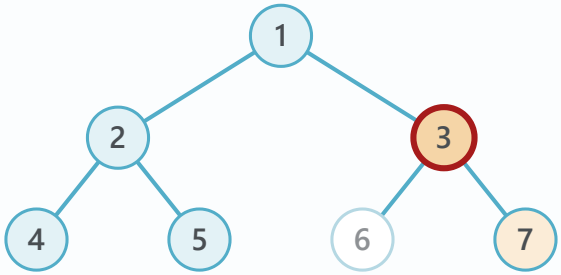
`yigin = []` `gecici = 3`

ÇIKTI

```
1 2 4 5 3
```

Yığın ile Kök Önde Gezinme

```
1  public void kokBastaYigin(AgacDugumu kok) {
2    if (kok == null) return;
3    Deque<AgacDugumu> yigin = new ArrayDeque<>
   ();
4    yigin.push(kok);
5    while (!yigin.isEmpty()) {
6      AgacDugumu gecici = yigin.pop();
7      System.out.print(gecici.veri + " ");
8      if (gecici.sag != null)
▶        yigin.push(gecici.sag);  // önce sağ
10     if (gecici.sol != null)
11       yigin.push(gecici.sol);  // sol üstte
12   }
13 }
```



şimdi · yığında · yazdırıldı

Sağ çocuk **7** itilir. Önce sağ itilir ki sol üstte kalıp **önce işlensin** (LIFO).

Adım 11 / 15

DEĞİŞKENLER

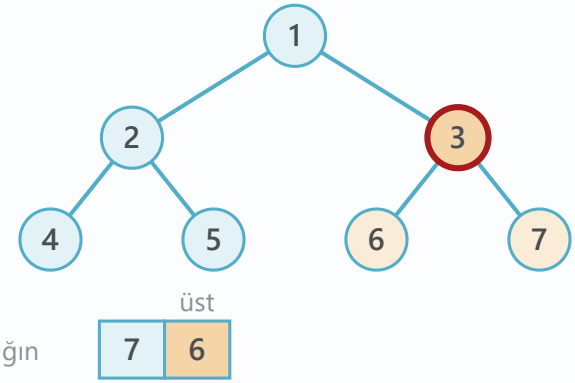
`yigin = [7]` `gecici = 3`

ÇIKTI

```
1 2 4 5 3
```

Yığın ile Kök Önde Gezinme

```
1  public void kokBastaYigin(AgacDugumu kok) {
2    if (kok == null) return;
3    Deque<AgacDugumu> yigin = new ArrayDeque<>
   ();
4    yigin.push(kok);
5    while (!yigin.isEmpty()) {
6      AgacDugumu gecici = yigin.pop();
7      System.out.print(gecici.veri + " ");
8      if (gecici.sag != null)
9        yigin.push(gecici.sag);  // önce sağ
10     if (gecici.sol != null)
▶        yigin.push(gecici.sol);  // sol üstte
12   }
13 }
```



şimdi yığında yazdırıldı

Sol çocuk **6** itilir → yığının üstünde; bir sonraki `pop()` onu verir.

Adım 12 / 15

DEĞİŞKENLER

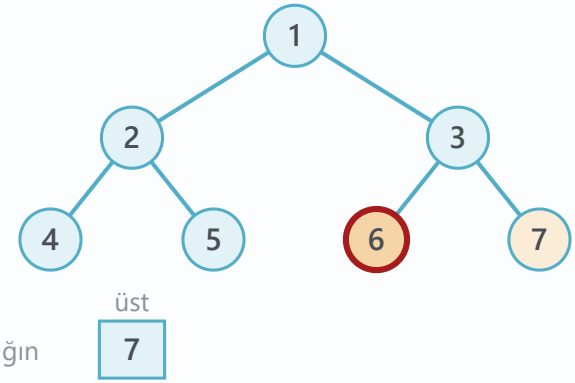
`yigin = [7, 6]` `gecici = 3`

ÇIKTI

```
1 2 4 5 3
```

Yığın ile Kök Önde Gezinme

```
1  public void kokBastaYigin(AgacDugumu kok) {
2    if (kok == null) return;
3    Deque<AgacDugumu> yigin = new ArrayDeque<>
   ();
4    yigin.push(kok);
5    while (!yigin.isEmpty()) {
▶      AgacDugumu gecici = yigin.pop();
▶      System.out.print(gecici.veri + " ");
8      if (gecici.sag != null)
9        yigin.push(gecici.sag);  // önce sağ
10     if (gecici.sol != null)
11       yigin.push(gecici.sol);  // sol üstte
12   }
13 }
```



şimdi yığında yazdırıldı

`pop()` üstteki **6**'yı verir; hemen **yazdırılır** (kök önde: önce kök). 6 yaprak: itilecek çocuk yok.

Adım 13 / 15

DEĞİŞKENLER

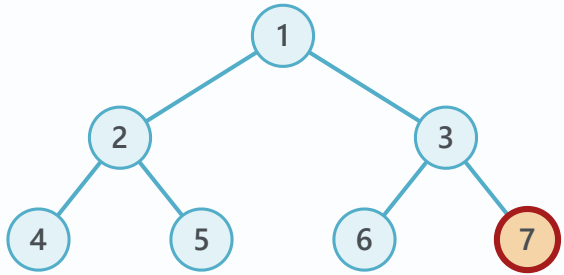
`yigin = [7]` `gecici = 6`

ÇIKTI

```
1 2 4 5 3 6
```

Yığın ile Kök Önde Gezinme

```
1  public void kokBastaYigin(AgacDugumu kok) {
2    if (kok == null) return;
3    Deque<AgacDugumu> yigin = new ArrayDeque<>
   ();
4    yigin.push(kok);
5    while (!yigin.isEmpty()) {
▶      AgacDugumu gecici = yigin.pop();
▶      System.out.print(gecici.veri + " ");
8      if (gecici.sag != null)
9        yigin.push(gecici.sag);  // önce sağ
10     if (gecici.sol != null)
11       yigin.push(gecici.sol);  // sol üstte
12   }
13 }
```



Yığın boş

şimdi yığında yazdırıldı

`pop()` üstteki **7**'yi verir; hemen **yazdırılır** (kök önde: önce kök). 7 yaprak: itilecek çocuk yok.

Adım 14 / 15

DEĞİŞKENLER

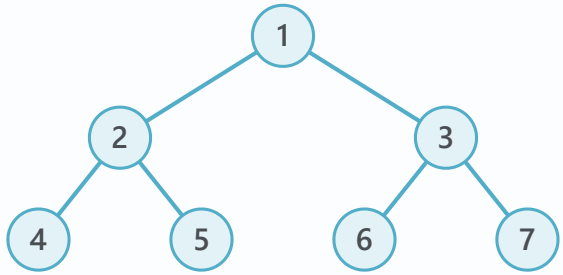
`yigin = []` `gecici = 7`

ÇIKTI

```
1 2 4 5 3 6 7
```

Yığın ile Kök Önde Gezinme

```
1  public void kokBastaYigin(AgacDugumu kok) {
2    if (kok == null) return;
3    Deque<AgacDugumu> yigin = new ArrayDeque<>
   ();
4    yigin.push(kok);
▶    while (!yigin.isEmpty()) {
6      AgacDugumu gecici = yigin.pop();
7      System.out.print(gecici.veri + " ");
8      if (gecici.sag != null)
9        yigin.push(gecici.sag);  // önce sağ
10     if (gecici.sol != null)
11       yigin.push(gecici.sol);  // sol üstte
12   }
13 }
```



Yığın boş

şimdi yığında yazdırıldı

Yığın boş → döngü biter. **Çıktı: 1 2 4 5 3 6 7** — özyinelemeli kök önde ile aynı.

Adım 15 / 15

DEĞİŞKENLER

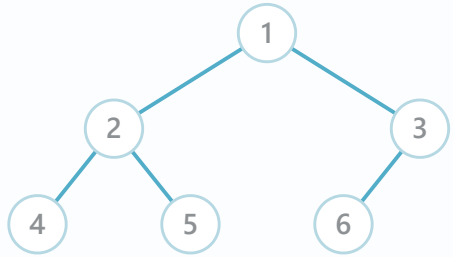
`yigin = []` `gecici = 7`

ÇIKTI

```
1 2 4 5 3 6 7
```

Yığın ile Kök Ortada Gezinme

```
1  public void kokOrtadaYigin(AgacDugumu kok) {
2    if (kok == null) return;
3    Deque<AgacDugumu> yigin = new ArrayDeque<>();
▶    AgacDugumu gecici = kok;
5    while (!yigin.isEmpty() || gecici != null) {
6      if (gecici != null) {
7        yigin.push(gecici);
8        gecici = gecici.sol;
9      } else {
10       gecici = yigin.pop();
11       System.out.print(gecici.veri + " ");
12       gecici = gecici.sag;
13     }
14   }
15 }
```



Yığın boş

şimdi yığında yazdırıldı

`gecici` kökten başlar; yığın boş.

Adım 1 / 14

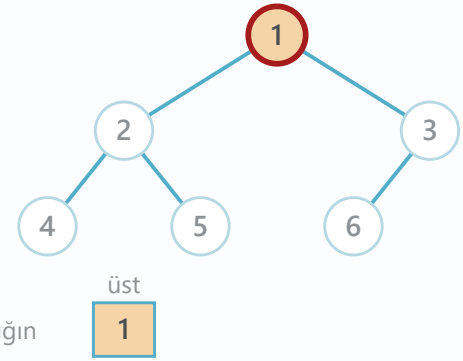
DEĞİŞKENLER

`yigin = []` `gecici = 1`

ÇIKTI

Yığın ile Kök Ortada Gezinme

```
1  public void kokOrtadaYigin(AgacDugumu kok) {
2    if (kok == null) return;
3    Deque<AgacDugumu> yigin = new ArrayDeque<>();
4    AgacDugumu gecici = kok;
5    while (!yigin.isEmpty() || gecici != null) {
6      if (gecici != null) {
▶        yigin.push(gecici);
▶        gecici = gecici.sol;
9      } else {
10       gecici = yigin.pop();
11       System.out.print(gecici.veri + " ");
12       gecici = gecici.sag;
13     }
14   }
15 }
```



şimdi · yığında · yazdırıldı

1 itilir (sonra yazdırılacak), `gecici = 1.sol` = 2. Sola inmeye devam.

Adım 2 / 14

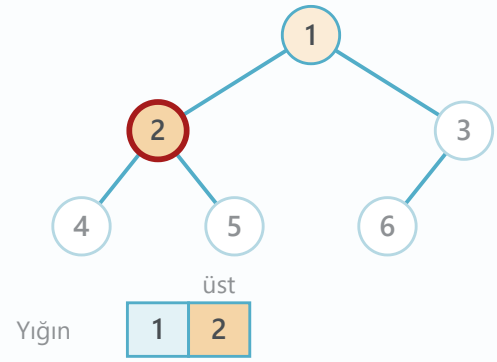
DEĞİŞKENLER

`yigin = [1]` `gecici = 2`

ÇIKTI

Yığın ile Kök Ortada Gezinme

```
1  public void kokOrtadaYigin(AgacDugumu kok) {
2    if (kok == null) return;
3    Deque<AgacDugumu> yigin = new ArrayDeque<>();
4    AgacDugumu gecici = kok;
5    while (!yigin.isEmpty() || gecici != null) {
6      if (gecici != null) {
▶        yigin.push(gecici);
▶        gecici = gecici.sol;
9      } else {
10       gecici = yigin.pop();
11       System.out.print(gecici.veri + " ");
12       gecici = gecici.sag;
13     }
14   }
15 }
```



şimdi · yığında · yazdırıldı

2 itilir (sonra yazdırılacak), `gecici = 2.sol` = 4. Sola inmeye devam.

Adım 3 / 14

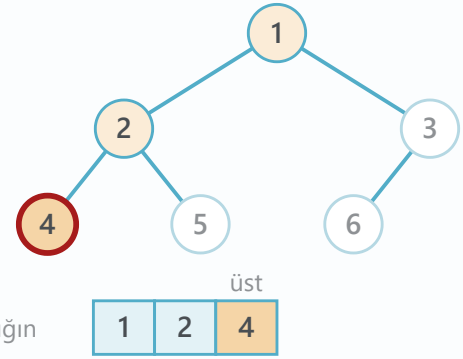
DEĞİŞKENLER

`yigin = [1, 2]` `gecici = 4`

ÇIKTI

Yığın ile Kök Ortada Gezinme

```
1  public void kokOrtadaYigin(AgacDugumu kok) {
2    if (kok == null) return;
3    Deque<AgacDugumu> yigin = new ArrayDeque<>();
4    AgacDugumu gecici = kok;
5    while (!yigin.isEmpty() || gecici != null) {
6      if (gecici != null) {
▶        yigin.push(gecici);
▶        gecici = gecici.sol;
9      } else {
10       gecici = yigin.pop();
11       System.out.print(gecici.veri + " ");
12       gecici = gecici.sag;
13     }
14   }
15 }
```



şimdi · yığında · yazdırıldı

4 itilir (sonra yazdırılacak), `gecici = 4.sol` = null. Sola gidilecek yer kalmadı.

Adım 4 / 14

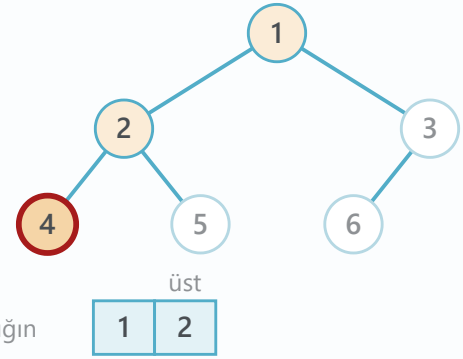
DEĞİŞKENLER

`yigin = [1, 2, 4]` `gecici = null`

ÇIKTI

Yığın ile Kök Ortada Gezinme

```
1  public void kokOrtadaYigin(AgacDugumu kok) {
2    if (kok == null) return;
3    Deque<AgacDugumu> yigin = new ArrayDeque<>();
4    AgacDugumu gecici = kok;
5    while (!yigin.isEmpty() || gecici != null) {
6      if (gecici != null) {
7        yigin.push(gecici);
8        gecici = gecici.sol;
9      } else {
▶        gecici = yigin.pop();
▶        System.out.print(gecici.veri + " ");
▶        gecici = gecici.sag;
13     }
14   }
15 }
```



şimdi · yığında · yazdırıldı

gecici null → **4** çıkarılıp **yazdırılır** (solundakilerin hepsi bitti). `gecici = 4.sag` = null.

Adım 5 / 14

DEĞİŞKENLER

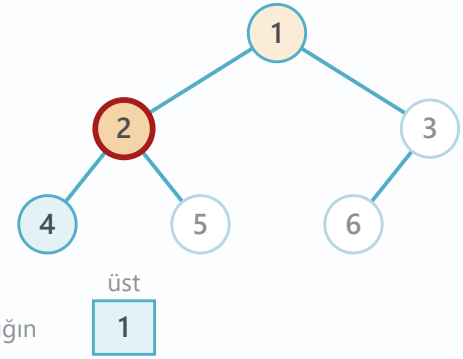
`yigin = [1, 2]` `gecici = null`

ÇIKTI

```
4
```

Yığın ile Kök Ortada Gezinme

```
1  public void kokOrtadaYigin(AgacDugumu kok) {
2    if (kok == null) return;
3    Deque<AgacDugumu> yigin = new ArrayDeque<>();
4    AgacDugumu gecici = kok;
5    while (!yigin.isEmpty() || gecici != null) {
6      if (gecici != null) {
7        yigin.push(gecici);
8        gecici = gecici.sol;
9      } else {
▶        gecici = yigin.pop();
▶        System.out.print(gecici.veri + " ");
▶        gecici = gecici.sag;
13     }
14   }
15 }
```



şimdi · yığında · yazdırıldı

gecici null → **2** çıkarılıp **yazdırılır** (solundakilerin hepsi bitti). `gecici = 2.sag` = 5.

Adım 6 / 14

DEĞİŞKENLER

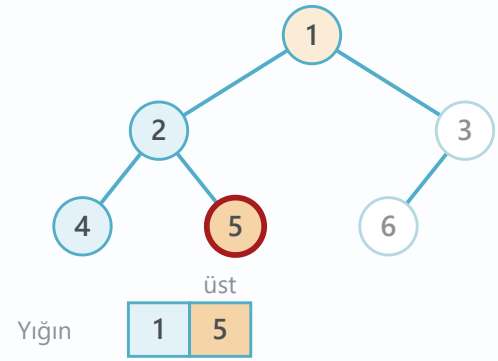
`yigin = [1]` `gecici = 5`

ÇIKTI

```
4 2
```

Yığın ile Kök Ortada Gezinme

```
1  public void kokOrtadaYigin(AgacDugumu kok) {
2    if (kok == null) return;
3    Deque<AgacDugumu> yigin = new ArrayDeque<>();
4    AgacDugumu gecici = kok;
5    while (!yigin.isEmpty() || gecici != null) {
6      if (gecici != null) {
▶        yigin.push(gecici);
▶        gecici = gecici.sol;
9      } else {
10       gecici = yigin.pop();
11       System.out.print(gecici.veri + " ");
12       gecici = gecici.sag;
13     }
14   }
15 }
```



şimdi · yığında · yazdırıldı

5 itilir (sonra yazdırılacak), `gecici = 5.sol` = null. Sola gidilecek yer kalmadı.

Adım 7 / 14

DEĞİŞKENLER

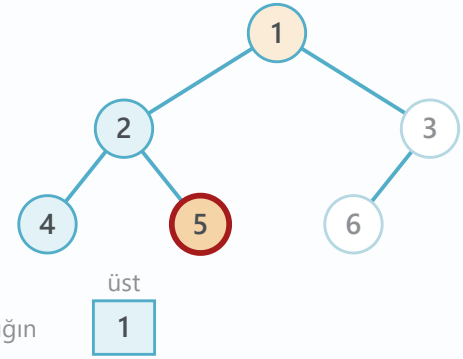
`yigin = [1, 5]` `gecici = null`

ÇIKTI

```
4 2
```

Yığın ile Kök Ortada Gezinme

```
1  public void kokOrtadaYigin(AgacDugumu kok) {
2    if (kok == null) return;
3    Deque<AgacDugumu> yigin = new ArrayDeque<>();
4    AgacDugumu gecici = kok;
5    while (!yigin.isEmpty() || gecici != null) {
6      if (gecici != null) {
7        yigin.push(gecici);
8        gecici = gecici.sol;
9      } else {
▶        gecici = yigin.pop();
▶        System.out.print(gecici.veri + " ");
▶        gecici = gecici.sag;
13     }
14   }
15 }
```



şimdi · yığında · yazdırıldı

gecici null → **5** çıkarılıp **yazdırılır** (solundakilerin hepsi bitti). `gecici = 5.sag` = null.

Adım 8 / 14

DEĞİŞKENLER

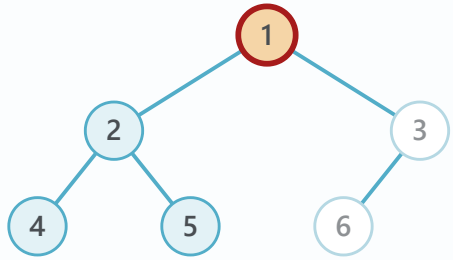
`yigin = [1]` `gecici = null`

ÇIKTI

```
4 2 5
```

Yığın ile Kök Ortada Gezinme

```
1  public void kokOrtadaYigin(AgacDugumu kok) {
2    if (kok == null) return;
3    Deque<AgacDugumu> yigin = new ArrayDeque<>();
4    AgacDugumu gecici = kok;
5    while (!yigin.isEmpty() || gecici != null) {
6      if (gecici != null) {
7        yigin.push(gecici);
8        gecici = gecici.sol;
9      } else {
▶        gecici = yigin.pop();
▶        System.out.print(gecici.veri + " ");
▶        gecici = gecici.sag;
13     }
14   }
15 }
```



Yığın boş

şimdi yığında yazdırıldı

gecici null → **1** çıkarılıp **yazdırılır** (solundakilerin hepsi bitti). `gecici = 1.sag` = 3.

Adım 9 / 14

DEĞİŞKENLER

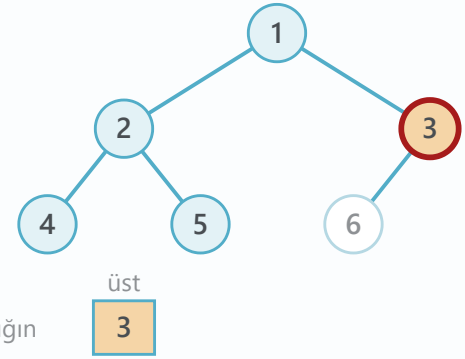
`yigin = []` `gecici = 3`

ÇIKTI

```
4 2 5 1
```

Yığın ile Kök Ortada Gezinme

```
1  public void kokOrtadaYigin(AgacDugumu kok) {
2    if (kok == null) return;
3    Deque<AgacDugumu> yigin = new ArrayDeque<>();
4    AgacDugumu gecici = kok;
5    while (!yigin.isEmpty() || gecici != null) {
6      if (gecici != null) {
▶        yigin.push(gecici);
▶        gecici = gecici.sol;
9      } else {
10       gecici = yigin.pop();
11       System.out.print(gecici.veri + " ");
12       gecici = gecici.sag;
13     }
14   }
15 }
```



şimdi · yığında · yazdırıldı

3 itilir (sonra yazdırılacak), `gecici = 3.sol` = 6. Sola inmeye devam.

Adım 10 / 14

DEĞİŞKENLER

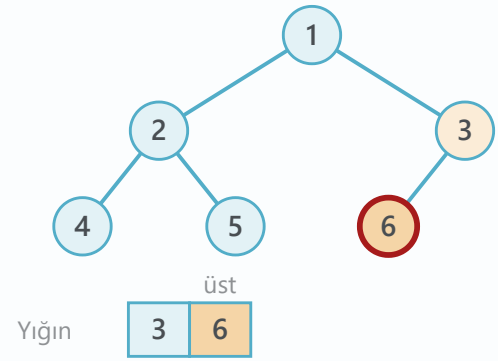
`yigin = [3]` `gecici = 6`

ÇIKTI

```
4 2 5 1
```

Yığın ile Kök Ortada Gezinme

```
1  public void kokOrtadaYigin(AgacDugumu kok) {
2    if (kok == null) return;
3    Deque<AgacDugumu> yigin = new ArrayDeque<>();
4    AgacDugumu gecici = kok;
5    while (!yigin.isEmpty() || gecici != null) {
6      if (gecici != null) {
▶        yigin.push(gecici);
▶        gecici = gecici.sol;
9      } else {
10       gecici = yigin.pop();
11       System.out.print(gecici.veri + " ");
12       gecici = gecici.sag;
13     }
14   }
15 }
```



şimdi · yığında · yazdırıldı

6 itilir (sonra yazdırılacak), `gecici = 6.sol` = null. Sola gidilecek yer kalmadı.

Adım 11 / 14

DEĞİŞKENLER

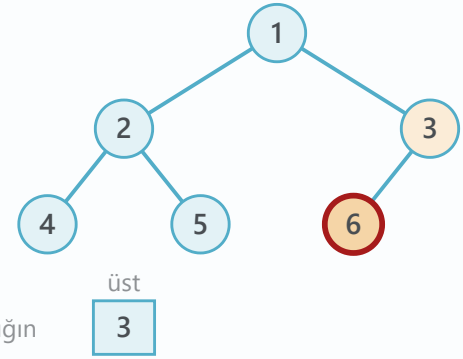
`yigin = [3, 6]` `gecici = null`

ÇIKTI

```
4 2 5 1
```

Yığın ile Kök Ortada Gezinme

```
1  public void kokOrtadaYigin(AgacDugumu kok) {
2    if (kok == null) return;
3    Deque<AgacDugumu> yigin = new ArrayDeque<>();
4    AgacDugumu gecici = kok;
5    while (!yigin.isEmpty() || gecici != null) {
6      if (gecici != null) {
7        yigin.push(gecici);
8        gecici = gecici.sol;
9      } else {
▶        gecici = yigin.pop();
▶        System.out.print(gecici.veri + " ");
▶        gecici = gecici.sag;
13     }
14   }
15 }
```



şimdi · yığında · yazdırıldı

gecici null → **6** çıkarılıp **yazdırılır** (solundakilerin hepsi bitti). `gecici = 6.sag` = null.

Adım 12 / 14

DEĞİŞKENLER

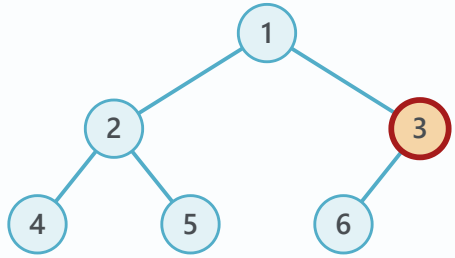
`yigin = [3]` `gecici = null`

ÇIKTI

```
4 2 5 1 6
```

Yığın ile Kök Ortada Gezinme

```
1  public void kokOrtadaYigin(AgacDugumu kok) {
2    if (kok == null) return;
3    Deque<AgacDugumu> yigin = new ArrayDeque<>();
4    AgacDugumu gecici = kok;
5    while (!yigin.isEmpty() || gecici != null) {
6      if (gecici != null) {
7        yigin.push(gecici);
8        gecici = gecici.sol;
9      } else {
▶        gecici = yigin.pop();
▶        System.out.print(gecici.veri + " ");
▶        gecici = gecici.sag;
13     }
14   }
15 }
```



Yığın boş

şimdi yığında yazdırıldı

gecici null → **3** çıkarılıp **yazdırılır** (solundakilerin hepsi bitti). `gecici = 3.sag` = null.

Adım 13 / 14

DEĞİŞKENLER

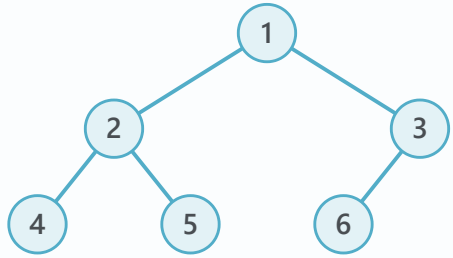
`yigin = []` `gecici = null`

ÇIKTI

```
4 2 5 1 6 3
```

Yığın ile Kök Ortada Gezinme

```
1  public void kokOrtadaYigin(AgacDugumu kok) {
2    if (kok == null) return;
3    Deque<AgacDugumu> yigin = new ArrayDeque<>();
4    AgacDugumu gecici = kok;
▶    while (!yigin.isEmpty() || gecici != null) {
6      if (gecici != null) {
7        yigin.push(gecici);
8        gecici = gecici.sol;
9      } else {
10       gecici = yigin.pop();
11       System.out.print(gecici.veri + " ");
12       gecici = gecici.sag;
13     }
14   }
15 }
```



Yığın boş

şimdi yığında yazdırıldı

Yığın boş ve gecici null → döngü biter. **Çıktı: 4 2 5 1 6 3** (sol → kök → sağ).

Adım 14 / 14

DEĞİŞKENLER

`yigin = []` `gecici = null`

ÇIKTI

```
4 2 5 1 6 3
```

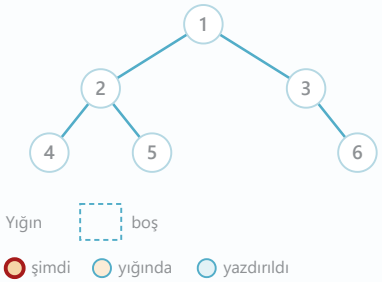
Yığın ile Kök Sonda Gezinme: Kod

```
public void kokSondaYigin(AgacDugumu kok) {
  AgacDugumu simdiki = kok;
  Deque<AgacDugumu> yigin = new ArrayDeque<>();
  while (simdiki != null || !yigin.isEmpty()) {
    if (simdiki != null) {
      yigin.push(simdiki);
      simdiki = simdiki.sol;
    } else {
      AgacDugumu gecici = yigin.peek().sag;
      if (gecici == null) {
        gecici = yigin.pop();
        System.out.print(gecici.veri + " ");
        while (!yigin.isEmpty()
           && gecici == yigin.peek().sag) {
          gecici = yigin.pop();
          System.out.print(gecici.veri + " ");
        }
      } else {
        simdiki = gecici;
      }
    }
  }
}
```

- Kök, iki alt ağacı da bittikten sonra yazdırılmalı; bu yüzden yığındaki düğüm hemen çıkarılmaz, önce `peek` ile bakılır.
- Üstteki düğümün **sağ çocuğu varsa** önce o alt ağaca gidilir.
- Sağ çocuk yoksa düğüm yazdırılır. Az önce yazdırılan düğüm, yığındakinin **sağ çocuğuysa** o ebeveynin de işi bitmiştir → o da çıkarılıp yazdırılır.

Yığın ile Kök Sonda Gezinme: İzleme

```
1  public void kokSondaYigin(AgacDugumu kok) {
▶    AgacDugumu simdiki = kok;
▶    Deque<AgacDugumu> yigin = new ArrayDeque<>();
4    while (simdiki != null || !yigin.isEmpty()) {
5      if (simdiki != null) {
6        yigin.push(simdiki);
7        simdiki = simdiki.sol;
8      } else {
9        AgacDugumu gecici = yigin.peek().sag;
10       if (gecici == null) {
11         gecici = yigin.pop();
12         System.out.print(gecici.veri + " ");
13         while (!yigin.isEmpty()
14             && gecici == yigin.peek().sag) {
15           gecici = yigin.pop();
16           System.out.print(gecici.veri + " ");
17         }
18       } else {
19         simdiki = gecici;
20       }
21     }
22   }
23 }
```



`simdiki` kökten başlar, yığın boş.

Adım 1 / 20

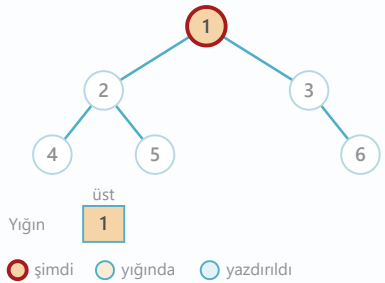
DEĞİŞKENLER

`simdiki = 1` `gecici = –` `yigin = []`

ÇIKTI

Yığın ile Kök Sonda Gezinme: İzleme

```
1  public void kokSondaYigin(AgacDugumu kok) {
2    AgacDugumu simdiki = kok;
3    Deque<AgacDugumu> yigin = new ArrayDeque<>();
4    while (simdiki != null || !yigin.isEmpty()) {
5      if (simdiki != null) {
▶        yigin.push(simdiki);
▶        simdiki = simdiki.sol;
8      } else {
9        AgacDugumu gecici = yigin.peek().sag;
10       if (gecici == null) {
11         gecici = yigin.pop();
12         System.out.print(gecici.veri + " ");
13         while (!yigin.isEmpty()
14             && gecici == yigin.peek().sag) {
15           gecici = yigin.pop();
16           System.out.print(gecici.veri + " ");
17         }
18       } else {
19         simdiki = gecici;
20       }
21     }
22   }
23 }
```



1 itilir, `simdiki = 1.sol` = 2.

Adım 2 / 20

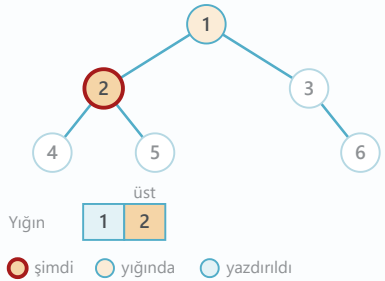
DEĞİŞKENLER

`simdiki = 2` `gecici = –` `yigin = [1]`

ÇIKTI

Yığın ile Kök Sonda Gezinme: İzleme

```
1  public void kokSondaYigin(AgacDugumu kok) {
2    AgacDugumu simdiki = kok;
3    Deque<AgacDugumu> yigin = new ArrayDeque<>();
4    while (simdiki != null || !yigin.isEmpty()) {
5      if (simdiki != null) {
▶        yigin.push(simdiki);
▶        simdiki = simdiki.sol;
8      } else {
9        AgacDugumu gecici = yigin.peek().sag;
10       if (gecici == null) {
11         gecici = yigin.pop();
12         System.out.print(gecici.veri + " ");
13         while (!yigin.isEmpty()
14             && gecici == yigin.peek().sag) {
15           gecici = yigin.pop();
16           System.out.print(gecici.veri + " ");
17         }
18       } else {
19         simdiki = gecici;
20       }
21     }
22   }
23 }
```



2 itilir, `simdiki = 2.sol` = 4.

Adım 3 / 20

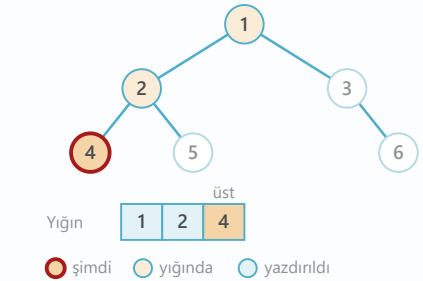
DEĞİŞKENLER

`simdiki = 4` `gecici = –` `yigin = [1, 2]`

ÇIKTI

Yığın ile Kök Sonda Gezinme: İzleme

```
1  public void kokSondaYigin(AgacDugumu kok) {
2    AgacDugumu simdiki = kok;
3    Deque<AgacDugumu> yigin = new ArrayDeque<>();
4    while (simdiki != null || !yigin.isEmpty()) {
5      if (simdiki != null) {
▶        yigin.push(simdiki);
▶        simdiki = simdiki.sol;
8      } else {
9        AgacDugumu gecici = yigin.peek().sag;
10       if (gecici == null) {
11         gecici = yigin.pop();
12         System.out.print(gecici.veri + " ");
13         while (!yigin.isEmpty()
14             && gecici == yigin.peek().sag) {
15           gecici = yigin.pop();
16           System.out.print(gecici.veri + " ");
17         }
18       } else {
19         simdiki = gecici;
20       }
21     }
22   }
23 }
```



4 itilir, `simdiki = 4.sol` = null.

Adım 4 / 20

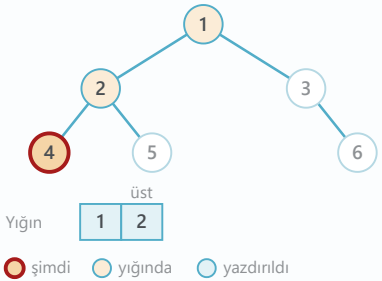
DEĞİŞKENLER

`simdiki = null` `gecici = –` `yigin = [1, 2, 4]`

ÇIKTI

Yığın ile Kök Sonda Gezinme: İzleme

```
1  public void kokSondaYigin(AgacDugumu kok) {
2    AgacDugumu simdiki = kok;
3    Deque<AgacDugumu> yigin = new ArrayDeque<>();
4    while (simdiki != null || !yigin.isEmpty()) {
5      if (simdiki != null) {
6        yigin.push(simdiki);
7        simdiki = simdiki.sol;
8      } else {
▶        AgacDugumu gecici = yigin.peek().sag;
▶        if (gecici == null) {
▶          gecici = yigin.pop();
▶          System.out.print(gecici.veri + " ");
13         while (!yigin.isEmpty()
14             && gecici == yigin.peek().sag) {
15           gecici = yigin.pop();
16           System.out.print(gecici.veri + " ");
17         }
18       } else {
19         simdiki = gecici;
20       }
21     }
22   }
23 }
```



`peek()` = 4, sağ çocuğu yok → iki alt ağacı da bitmiş sayılır: **4 çıkarılıp yazdırılır**.

Adım 5 / 20

DEĞİŞKENLER

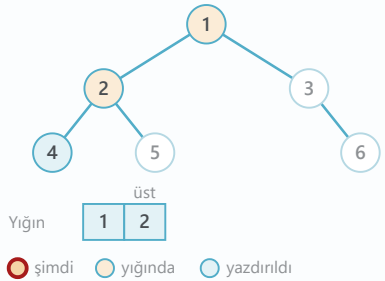
`simdiki = null` `gecici = 4` `yigin = [1, 2]`

ÇIKTI

```
4
```

Yığın ile Kök Sonda Gezinme: İzleme

```
1  public void kokSondaYigin(AgacDugumu kok) {
2    AgacDugumu simdiki = kok;
3    Deque<AgacDugumu> yigin = new ArrayDeque<>();
4    while (simdiki != null || !yigin.isEmpty()) {
5      if (simdiki != null) {
6        yigin.push(simdiki);
7        simdiki = simdiki.sol;
8      } else {
9        AgacDugumu gecici = yigin.peek().sag;
10       if (gecici == null) {
11         gecici = yigin.pop();
12         System.out.print(gecici.veri + " ");
▶          while (!yigin.isEmpty()
▶             && gecici == yigin.peek().sag) {
15           gecici = yigin.pop();
16           System.out.print(gecici.veri + " ");
17         }
18       } else {
19         simdiki = gecici;
20       }
21     }
22   }
23 }
```



4, üstteki 2'nin sağ çocuğu değil (2.sag = 5) → iç döngü durur; 2'nin sağ alt ağacına henüz girilmedi.

Adım 6 / 20

DEĞİŞKENLER

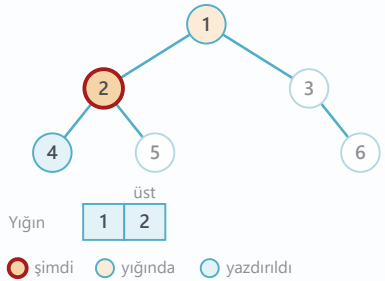
`simdiki = null` `gecici = 4` `yigin = [1, 2]`

ÇIKTI

```
4
```

Yığın ile Kök Sonda Gezinme: İzleme

```
1  public void kokSondaYigin(AgacDugumu kok) {
2    AgacDugumu simdiki = kok;
3    Deque<AgacDugumu> yigin = new ArrayDeque<>();
4    while (simdiki != null || !yigin.isEmpty()) {
5      if (simdiki != null) {
6        yigin.push(simdiki);
7        simdiki = simdiki.sol;
8      } else {
▶        AgacDugumu gecici = yigin.peek().sag;
10       if (gecici == null) {
11         gecici = yigin.pop();
12         System.out.print(gecici.veri + " ");
13         while (!yigin.isEmpty()
14             && gecici == yigin.peek().sag) {
15           gecici = yigin.pop();
16           System.out.print(gecici.veri + " ");
17         }
18       } else {
▶          simdiki = gecici;
20       }
21     }
22   }
23 }
```



`peek()` = 2; sağ çocuğu **5** var → 2 henüz yazdırılamaz, önce `simdiki = 5` ile sağ alt ağaca gidilir.

Adım 7 / 20

DEĞİŞKENLER

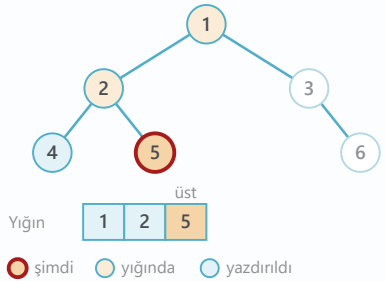
`simdiki = 5` `gecici = 5` `yigin = [1, 2]`

ÇIKTI

```
4
```

Yığın ile Kök Sonda Gezinme: İzleme

```
1  public void kokSondaYigin(AgacDugumu kok) {
2    AgacDugumu simdiki = kok;
3    Deque<AgacDugumu> yigin = new ArrayDeque<>();
4    while (simdiki != null || !yigin.isEmpty()) {
5      if (simdiki != null) {
▶        yigin.push(simdiki);
▶        simdiki = simdiki.sol;
8      } else {
9        AgacDugumu gecici = yigin.peek().sag;
10       if (gecici == null) {
11         gecici = yigin.pop();
12         System.out.print(gecici.veri + " ");
13         while (!yigin.isEmpty()
14             && gecici == yigin.peek().sag) {
15           gecici = yigin.pop();
16           System.out.print(gecici.veri + " ");
17         }
18       } else {
19         simdiki = gecici;
20       }
21     }
22   }
23 }
```



5 itilir, `simdiki = 5.sol` = null.

Adım 8 / 20

DEĞİŞKENLER

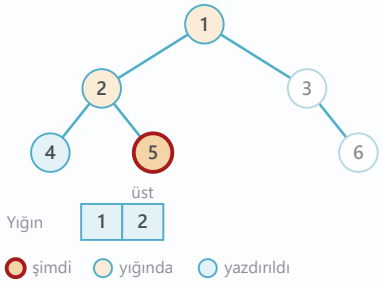
`simdiki = null` `gecici = –` `yigin = [1, 2, 5]`

ÇIKTI

```
4
```

Yığın ile Kök Sonda Gezinme: İzleme

```
1  public void kokSondaYigin(AgacDugumu kok) {
2    AgacDugumu simdiki = kok;
3    Deque<AgacDugumu> yigin = new ArrayDeque<>();
4    while (simdiki != null || !yigin.isEmpty()) {
5      if (simdiki != null) {
6        yigin.push(simdiki);
7        simdiki = simdiki.sol;
8      } else {
▶        AgacDugumu gecici = yigin.peek().sag;
▶        if (gecici == null) {
▶          gecici = yigin.pop();
▶          System.out.print(gecici.veri + " ");
13         while (!yigin.isEmpty()
14             && gecici == yigin.peek().sag) {
15           gecici = yigin.pop();
16           System.out.print(gecici.veri + " ");
17         }
18       } else {
19         simdiki = gecici;
20       }
21     }
22   }
23 }
```



`peek()` = 5, sağ çocuğu yok → iki alt ağacı da bitmiş sayılır: **5 çıkarılıp yazdırılır**.

Adım 9 / 20

DEĞİŞKENLER

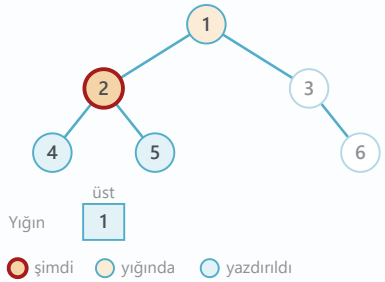
`simdiki = null` `gecici = 5` `yigin = [1, 2]`

ÇIKTI

```
4 5
```

Yığın ile Kök Sonda Gezinme: İzleme

```
1  public void kokSondaYigin(AgacDugumu kok) {
2    AgacDugumu simdiki = kok;
3    Deque<AgacDugumu> yigin = new ArrayDeque<>();
4    while (simdiki != null || !yigin.isEmpty()) {
5      if (simdiki != null) {
6        yigin.push(simdiki);
7        simdiki = simdiki.sol;
8      } else {
9        AgacDugumu gecici = yigin.peek().sag;
10       if (gecici == null) {
11         gecici = yigin.pop();
12         System.out.print(gecici.veri + " ");
13         while (!yigin.isEmpty()
▶              && gecici == yigin.peek().sag) {
▶            gecici = yigin.pop();
▶            System.out.print(gecici.veri + " ");
17         }
18       } else {
19         simdiki = gecici;
20       }
21     }
22   }
23 }
```



Az önce yazdırılan düğüm, 2'nin **sağ** çocuğuydu → 2'nin sağ alt ağacı da bitti: **2 çıkarılıp yazdırılır**.

Adım 10 / 20

DEĞİŞKENLER

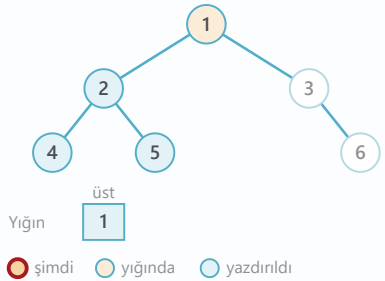
`simdiki = null` `gecici = 2` `yigin = [1]`

ÇIKTI

```
4 5 2
```

Yığın ile Kök Sonda Gezinme: İzleme

```
1  public void kokSondaYigin(AgacDugumu kok) {
2    AgacDugumu simdiki = kok;
3    Deque<AgacDugumu> yigin = new ArrayDeque<>();
4    while (simdiki != null || !yigin.isEmpty()) {
5      if (simdiki != null) {
6        yigin.push(simdiki);
7        simdiki = simdiki.sol;
8      } else {
9        AgacDugumu gecici = yigin.peek().sag;
10       if (gecici == null) {
11         gecici = yigin.pop();
12         System.out.print(gecici.veri + " ");
▶          while (!yigin.isEmpty()
▶             && gecici == yigin.peek().sag) {
15           gecici = yigin.pop();
16           System.out.print(gecici.veri + " ");
17         }
18       } else {
19         simdiki = gecici;
20       }
21     }
22   }
23 }
```



2, üstteki 1'in sağ çocuğu değil (1.sag = 3) → iç döngü durur; 1'in sağ alt ağacına henüz girilmedi.

Adım 11 / 20

DEĞİŞKENLER

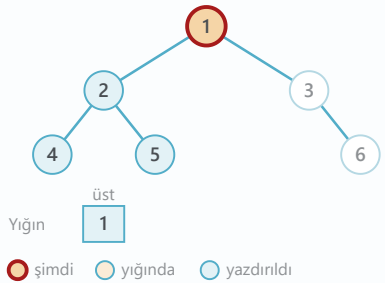
`simdiki = null` `gecici = 2` `yigin = [1]`

ÇIKTI

```
4 5 2
```

Yığın ile Kök Sonda Gezinme: İzleme

```
1  public void kokSondaYigin(AgacDugumu kok) {
2    AgacDugumu simdiki = kok;
3    Deque<AgacDugumu> yigin = new ArrayDeque<>();
4    while (simdiki != null || !yigin.isEmpty()) {
5      if (simdiki != null) {
6        yigin.push(simdiki);
7        simdiki = simdiki.sol;
8      } else {
▶        AgacDugumu gecici = yigin.peek().sag;
10       if (gecici == null) {
11         gecici = yigin.pop();
12         System.out.print(gecici.veri + " ");
13         while (!yigin.isEmpty()
14             && gecici == yigin.peek().sag) {
15           gecici = yigin.pop();
16           System.out.print(gecici.veri + " ");
17         }
18       } else {
▶          simdiki = gecici;
20       }
21     }
22   }
23 }
```



`peek()` = 1; sağ çocuğu **3** var → 1 henüz yazdırılamaz, önce `simdiki = 3` ile sağ alt ağaca gidilir.

Adım 12 / 20

DEĞİŞKENLER

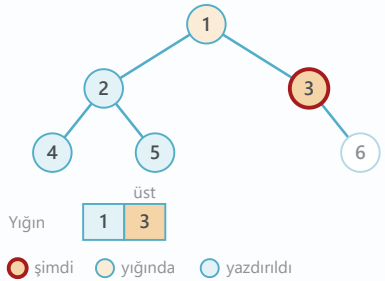
`simdiki = 3` `gecici = 3` `yigin = [1]`

ÇIKTI

```
4 5 2
```

Yığın ile Kök Sonda Gezinme: İzleme

```
1  public void kokSondaYigin(AgacDugumu kok) {
2    AgacDugumu simdiki = kok;
3    Deque<AgacDugumu> yigin = new ArrayDeque<>();
4    while (simdiki != null || !yigin.isEmpty()) {
5      if (simdiki != null) {
▶        yigin.push(simdiki);
▶        simdiki = simdiki.sol;
8      } else {
9        AgacDugumu gecici = yigin.peek().sag;
10       if (gecici == null) {
11         gecici = yigin.pop();
12         System.out.print(gecici.veri + " ");
13         while (!yigin.isEmpty()
14             && gecici == yigin.peek().sag) {
15           gecici = yigin.pop();
16           System.out.print(gecici.veri + " ");
17         }
18       } else {
19         simdiki = gecici;
20       }
21     }
22   }
23 }
```



3 itilir, `simdiki = 3.sol` = null.

Adım 13 / 20

DEĞİŞKENLER

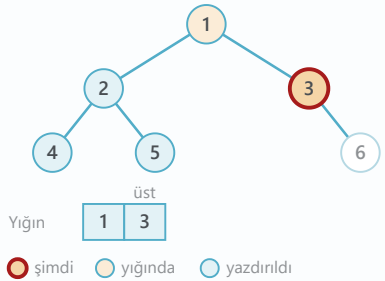
`simdiki = null` `gecici = –` `yigin = [1, 3]`

ÇIKTI

```
4 5 2
```

Yığın ile Kök Sonda Gezinme: İzleme

```
1  public void kokSondaYigin(AgacDugumu kok) {
2    AgacDugumu simdiki = kok;
3    Deque<AgacDugumu> yigin = new ArrayDeque<>();
4    while (simdiki != null || !yigin.isEmpty()) {
5      if (simdiki != null) {
6        yigin.push(simdiki);
7        simdiki = simdiki.sol;
8      } else {
▶        AgacDugumu gecici = yigin.peek().sag;
10       if (gecici == null) {
11         gecici = yigin.pop();
12         System.out.print(gecici.veri + " ");
13         while (!yigin.isEmpty()
14             && gecici == yigin.peek().sag) {
15           gecici = yigin.pop();
16           System.out.print(gecici.veri + " ");
17         }
18       } else {
▶          simdiki = gecici;
20       }
21     }
22   }
23 }
```



`peek()` = 3; sağ çocuğu **6** var → 3 henüz yazdırılamaz, önce `simdiki = 6` ile sağ alt ağaca gidilir.

Adım 14 / 20

DEĞİŞKENLER

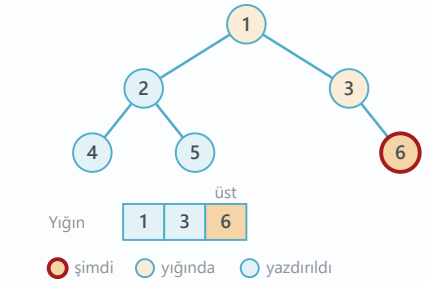
`simdiki = 6` `gecici = 6` `yigin = [1, 3]`

ÇIKTI

```
4 5 2
```

Yığın ile Kök Sonda Gezinme: İzleme

```
1  public void kokSondaYigin(AgacDugumu kok) {
2    AgacDugumu simdiki = kok;
3    Deque<AgacDugumu> yigin = new ArrayDeque<>();
4    while (simdiki != null || !yigin.isEmpty()) {
5      if (simdiki != null) {
▶        yigin.push(simdiki);
▶        simdiki = simdiki.sol;
8      } else {
9        AgacDugumu gecici = yigin.peek().sag;
10       if (gecici == null) {
11         gecici = yigin.pop();
12         System.out.print(gecici.veri + " ");
13         while (!yigin.isEmpty()
14             && gecici == yigin.peek().sag) {
15           gecici = yigin.pop();
16           System.out.print(gecici.veri + " ");
17         }
18       } else {
19         simdiki = gecici;
20       }
21     }
22   }
23 }
```



6 itilir, `simdiki = 6.sol` = null.

Adım 15 / 20

DEĞİŞKENLER

`simdiki = null` `gecici = –`

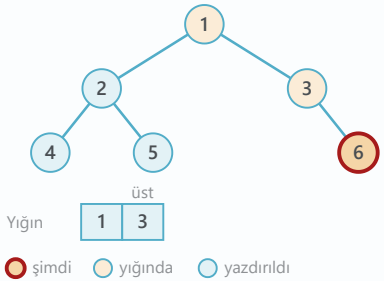
`yigin = [1, 3, 6]`

ÇIKTI

```
4 5 2
```

Yığın ile Kök Sonda Gezinme: İzleme

```
1  public void kokSondaYigin(AgacDugumu kok) {
2    AgacDugumu simdiki = kok;
3    Deque<AgacDugumu> yigin = new ArrayDeque<>();
4    while (simdiki != null || !yigin.isEmpty()) {
5      if (simdiki != null) {
6        yigin.push(simdiki);
7        simdiki = simdiki.sol;
8      } else {
▶        AgacDugumu gecici = yigin.peek().sag;
▶        if (gecici == null) {
▶          gecici = yigin.pop();
▶          System.out.print(gecici.veri + " ");
13         while (!yigin.isEmpty()
14             && gecici == yigin.peek().sag) {
15           gecici = yigin.pop();
16           System.out.print(gecici.veri + " ");
17         }
18       } else {
19         simdiki = gecici;
20       }
21     }
22   }
23 }
```



`peek()` = 6, sağ çocuğu yok → iki alt ağacı da bitmiş sayılır: **6 çıkarılıp yazdırılır**.

Adım 16 / 20

DEĞİŞKENLER

`simdiki = null` `gecici = 6`

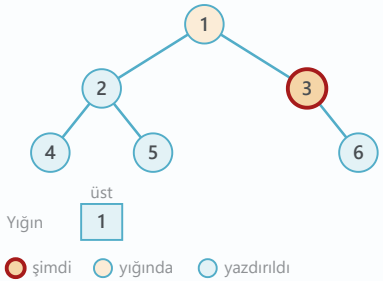
`yigin = [1, 3]`

ÇIKTI

```
4 5 2 6
```

Yığın ile Kök Sonda Gezinme: İzleme

```
1  public void kokSondaYigin(AgacDugumu kok) {
2    AgacDugumu simdiki = kok;
3    Deque<AgacDugumu> yigin = new ArrayDeque<>();
4    while (simdiki != null || !yigin.isEmpty()) {
5      if (simdiki != null) {
6        yigin.push(simdiki);
7        simdiki = simdiki.sol;
8      } else {
9        AgacDugumu gecici = yigin.peek().sag;
10       if (gecici == null) {
11         gecici = yigin.pop();
12         System.out.print(gecici.veri + " ");
13         while (!yigin.isEmpty()
▶              && gecici == yigin.peek().sag) {
▶            gecici = yigin.pop();
▶            System.out.print(gecici.veri + " ");
17         }
18       } else {
19         simdiki = gecici;
20       }
21     }
22   }
23 }
```



Az önce yazdırılan düğüm, 3'ün **sağ** çocuğuydu → 3'ün sağ alt ağacı da bitti: **3 çıkarılıp yazdırılır**.

Adım 17 / 20

DEĞİŞKENLER

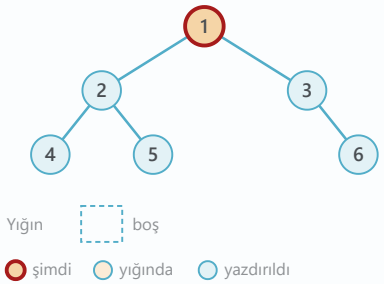
`simdiki = null` `gecici = 3` `yigin = [1]`

ÇIKTI

```
4 5 2 6 3
```

Yığın ile Kök Sonda Gezinme: İzleme

```
1  public void kokSondaYigin(AgacDugumu kok) {
2    AgacDugumu simdiki = kok;
3    Deque<AgacDugumu> yigin = new ArrayDeque<>();
4    while (simdiki != null || !yigin.isEmpty()) {
5      if (simdiki != null) {
6        yigin.push(simdiki);
7        simdiki = simdiki.sol;
8      } else {
9        AgacDugumu gecici = yigin.peek().sag;
10       if (gecici == null) {
11         gecici = yigin.pop();
12         System.out.print(gecici.veri + " ");
13         while (!yigin.isEmpty()
▶              && gecici == yigin.peek().sag) {
▶            gecici = yigin.pop();
▶            System.out.print(gecici.veri + " ");
17         }
18       } else {
19         simdiki = gecici;
20       }
21     }
22   }
23 }
```



Az önce yazdırılan düğüm, 1'in **sağ** çocuğuydu → 1'in sağ alt ağacı da bitti: **1 çıkarılıp yazdırılır**.

Adım 18 / 20

DEĞİŞKENLER

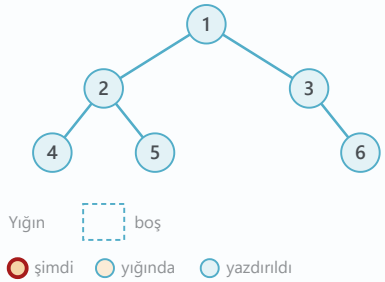
`simdiki = null` `gecici = 1` `yigin = []`

ÇIKTI

```
4 5 2 6 3 1
```

Yığın ile Kök Sonda Gezinme: İzleme

```
1  public void kokSondaYigin(AgacDugumu kok) {
2    AgacDugumu simdiki = kok;
3    Deque<AgacDugumu> yigin = new ArrayDeque<>();
4    while (simdiki != null || !yigin.isEmpty()) {
5      if (simdiki != null) {
6        yigin.push(simdiki);
7        simdiki = simdiki.sol;
8      } else {
9        AgacDugumu gecici = yigin.peek().sag;
10       if (gecici == null) {
11         gecici = yigin.pop();
12         System.out.print(gecici.veri + " ");
▶          while (!yigin.isEmpty()
14             && gecici == yigin.peek().sag) {
15           gecici = yigin.pop();
16           System.out.print(gecici.veri + " ");
17         }
18       } else {
19         simdiki = gecici;
20       }
21     }
22   }
23 }
```



Yığın boşaldı → iç döngü durur.

Adım 19 / 20

DEĞİŞKENLER

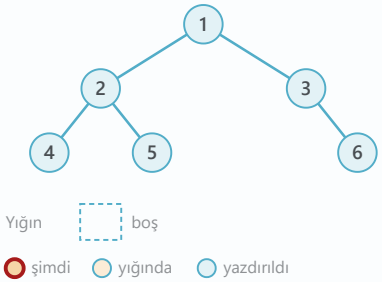
`simdiki = null` `gecici = 1` `yigin = []`

ÇIKTI

```
4 5 2 6 3 1
```

Yığın ile Kök Sonda Gezinme: İzleme

```
1  public void kokSondaYigin(AgacDugumu kok) {
2    AgacDugumu simdiki = kok;
3    Deque<AgacDugumu> yigin = new ArrayDeque<>();
▶    while (simdiki != null || !yigin.isEmpty()) {
5      if (simdiki != null) {
6        yigin.push(simdiki);
7        simdiki = simdiki.sol;
8      } else {
9        AgacDugumu gecici = yigin.peek().sag;
10       if (gecici == null) {
11         gecici = yigin.pop();
12         System.out.print(gecici.veri + " ");
13         while (!yigin.isEmpty()
14             && gecici == yigin.peek().sag) {
15           gecici = yigin.pop();
16           System.out.print(gecici.veri + " ");
17         }
18       } else {
19         simdiki = gecici;
20       }
21     }
22   }
23 }
```



simdiki null ve yığın boş → biter. **Çıktı: 4 5 2 6 3 1** (sol → sağ → kök).

Adım 20 / 20

DEĞİŞKENLER

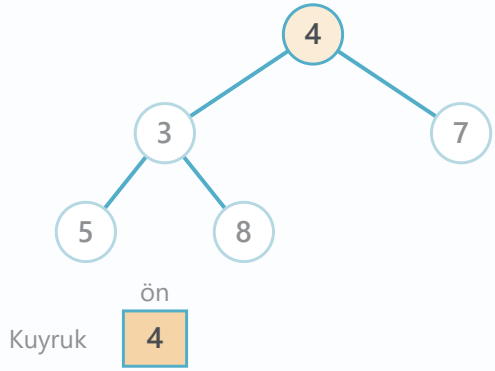
`simdiki = null` `gecici = 1` `yigin = []`

ÇIKTI

```
4 5 2 6 3 1
```

Kuyruk ile Düzey Sırası Gezinme

```
1  public void seviyeSirasi(AgacDugumu kok) {
2    if (kok == null) return;
3    Queue<AgacDugumu> kuyruk = new LinkedList<>
   ();
▶    kuyruk.add(kok);
5    while (!kuyruk.isEmpty()) {
6      AgacDugumu gecici = kuyruk.poll();
7      System.out.print(gecici.veri + " ");
8      if (gecici.sol != null)
9        kuyruk.add(gecici.sol);
10     if (gecici.sag != null)
11       kuyruk.add(gecici.sag);
12   }
13 }
```



şimdi kuyrukta yazdırıldı

Kök kuyruğa eklenir. Kuyruk FIFO: önce eklenen önce çıkar.

Adım 1 / 11

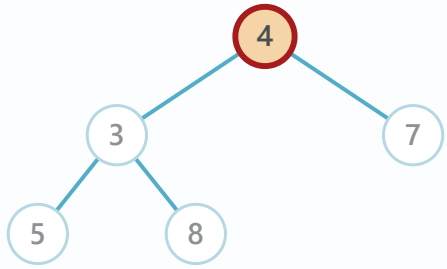
DEĞİŞKENLER

`kuyruk = [4]` `gecici = –`

ÇIKTI

Kuyruk ile Düzey Sırası Gezinme

```
1  public void seviyeSirasi(AgacDugumu kok) {
2    if (kok == null) return;
3    Queue<AgacDugumu> kuyruk = new LinkedList<>
   ();
4    kuyruk.add(kok);
5    while (!kuyruk.isEmpty()) {
▶      AgacDugumu gecici = kuyruk.poll();
▶      System.out.print(gecici.veri + " ");
8      if (gecici.sol != null)
9        kuyruk.add(gecici.sol);
10     if (gecici.sag != null)
11       kuyruk.add(gecici.sag);
12   }
13 }
```



Kuyruk boş

şimdi kuyrukta yazdırıldı

`poll()` öndeki **4**'ü verir ve 4 yazdırılır.

Adım 2 / 11

DEĞİŞKENLER

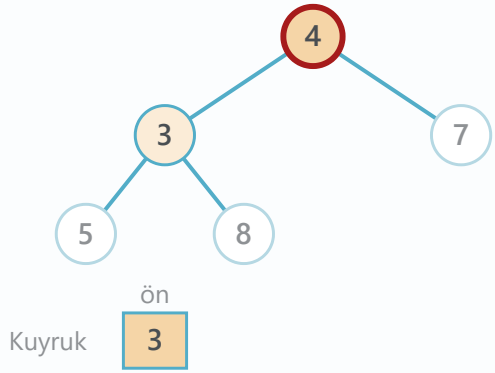
`kuyruk = []` `gecici = 4`

ÇIKTI

```
4
```

Kuyruk ile Düzey Sırası Gezinme

```
1  public void seviyeSirasi(AgacDugumu kok) {
2    if (kok == null) return;
3    Queue<AgacDugumu> kuyruk = new LinkedList<>
   ();
4    kuyruk.add(kok);
5    while (!kuyruk.isEmpty()) {
6      AgacDugumu gecici = kuyruk.poll();
7      System.out.print(gecici.veri + " ");
8      if (gecici.sol != null)
▶        kuyruk.add(gecici.sol);
10     if (gecici.sag != null)
11       kuyruk.add(gecici.sag);
12   }
13 }
```



şimdi · kuyrukta · yazdırıldı

Sol çocuk **3** kuyruğun **arkasına** eklenir; kendi düzeyindekilerden sonra işlenecek.

Adım 3 / 11

DEĞİŞKENLER

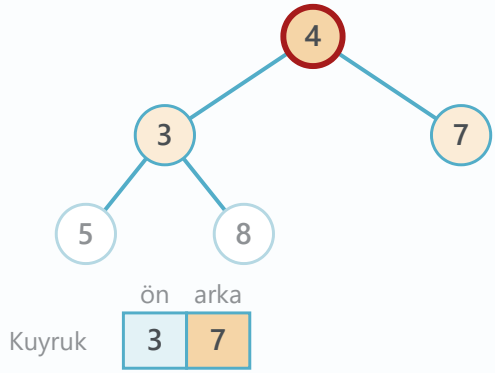
`kuyruk = [3]` `gecici = 4`

ÇIKTI

```
4
```

Kuyruk ile Düzey Sırası Gezinme

```
1  public void seviyeSirasi(AgacDugumu kok) {
2    if (kok == null) return;
3    Queue<AgacDugumu> kuyruk = new LinkedList<>
   ();
4    kuyruk.add(kok);
5    while (!kuyruk.isEmpty()) {
6      AgacDugumu gecici = kuyruk.poll();
7      System.out.print(gecici.veri + " ");
8      if (gecici.sol != null)
9        kuyruk.add(gecici.sol);
10     if (gecici.sag != null)
▶        kuyruk.add(gecici.sag);
12   }
13 }
```



şimdi kuyrukta yazdırıldı

Sağ çocuk **7** arkaya eklenir.

Adım 4 / 11

DEĞİŞKENLER

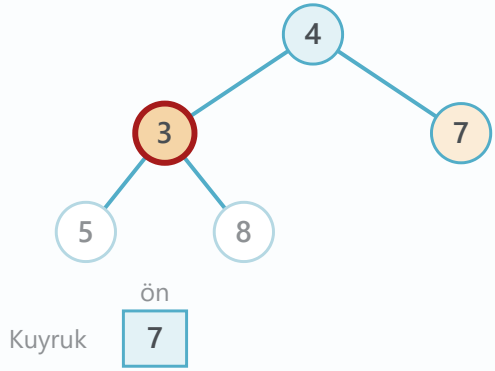
`kuyruk = [3, 7]` `gecici = 4`

ÇIKTI

```
4
```

Kuyruk ile Düzey Sırası Gezinme

```
1  public void seviyeSirasi(AgacDugumu kok) {
2    if (kok == null) return;
3    Queue<AgacDugumu> kuyruk = new LinkedList<>
   ();
4    kuyruk.add(kok);
5    while (!kuyruk.isEmpty()) {
▶      AgacDugumu gecici = kuyruk.poll();
▶      System.out.print(gecici.veri + " ");
8      if (gecici.sol != null)
9        kuyruk.add(gecici.sol);
10     if (gecici.sag != null)
11       kuyruk.add(gecici.sag);
12   }
13 }
```



şimdi · kuyrukta · yazdırıldı

`poll()` öndeki **3**'ü verir ve 3 yazdırılır.

Adım 5 / 11

DEĞİŞKENLER

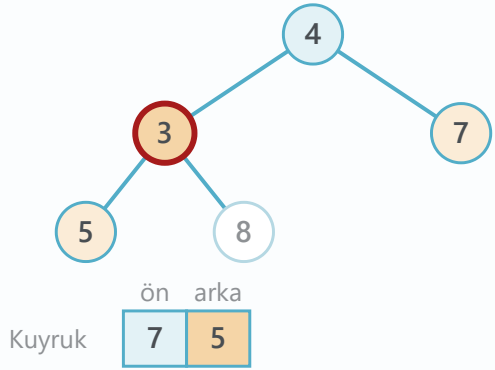
`kuyruk = [7]` `gecici = 3`

ÇIKTI

```
4 3
```

Kuyruk ile Düzey Sırası Gezinme

```
1  public void seviyeSirasi(AgacDugumu kok) {
2    if (kok == null) return;
3    Queue<AgacDugumu> kuyruk = new LinkedList<>
   ();
4    kuyruk.add(kok);
5    while (!kuyruk.isEmpty()) {
6      AgacDugumu gecici = kuyruk.poll();
7      System.out.print(gecici.veri + " ");
8      if (gecici.sol != null)
▶        kuyruk.add(gecici.sol);
10     if (gecici.sag != null)
11       kuyruk.add(gecici.sag);
12   }
13 }
```



şimdi · kuyrukta · yazdırıldı

Sol çocuk **5** kuyruğun **arkasına** eklenir; kendi düzeyindekilerden sonra işlenecek.

Adım 6 / 11

DEĞİŞKENLER

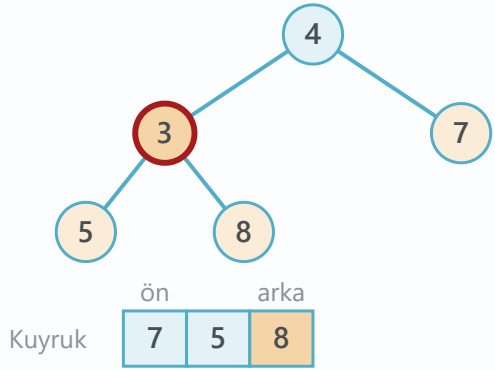
`kuyruk = [7, 5]` `gecici = 3`

ÇIKTI

```
4 3
```

Kuyruk ile Düzey Sırası Gezinme

```
1  public void seviyeSirasi(AgacDugumu kok) {
2    if (kok == null) return;
3    Queue<AgacDugumu> kuyruk = new LinkedList<>
   ();
4    kuyruk.add(kok);
5    while (!kuyruk.isEmpty()) {
6      AgacDugumu gecici = kuyruk.poll();
7      System.out.print(gecici.veri + " ");
8      if (gecici.sol != null)
9        kuyruk.add(gecici.sol);
10     if (gecici.sag != null)
▶        kuyruk.add(gecici.sag);
12   }
13 }
```



Sağ çocuk **8** arkaya eklenir.

Adım 7 / 11

DEĞİŞKENLER

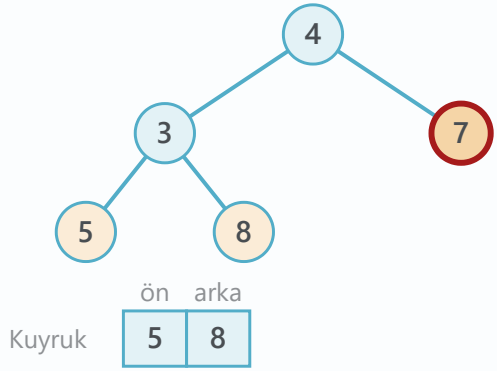
`kuyruk = [7, 5, 8]` `gecici = 3`

ÇIKTI

```
4 3
```

Kuyruk ile Düzey Sırası Gezinme

```
1  public void seviyeSirasi(AgacDugumu kok) {
2    if (kok == null) return;
3    Queue<AgacDugumu> kuyruk = new LinkedList<>
   ();
4    kuyruk.add(kok);
5    while (!kuyruk.isEmpty()) {
▶      AgacDugumu gecici = kuyruk.poll();
▶      System.out.print(gecici.veri + " ");
8      if (gecici.sol != null)
9        kuyruk.add(gecici.sol);
10     if (gecici.sag != null)
11       kuyruk.add(gecici.sag);
12   }
13 }
```



şimdi · kuyrukta · yazdırıldı

`poll()` öndeki **7**'yi verir ve 7 yazdırılır.

Yaprak: eklenecek çocuk yok.

Adım 8 / 11

DEĞİŞKENLER

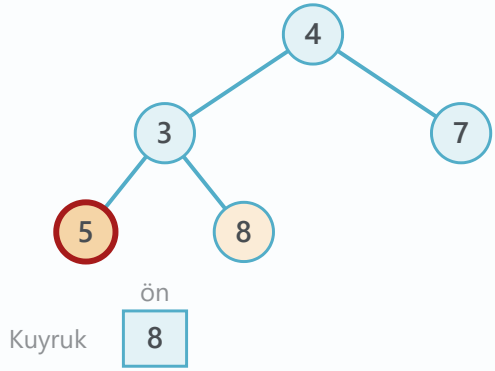
`kuyruk = [5, 8]` `gecici = 7`

ÇIKTI

```
4 3 7
```

Kuyruk ile Düzey Sırası Gezinme

```
1  public void seviyeSirasi(AgacDugumu kok) {
2    if (kok == null) return;
3    Queue<AgacDugumu> kuyruk = new LinkedList<>
   ();
4    kuyruk.add(kok);
5    while (!kuyruk.isEmpty()) {
▶      AgacDugumu gecici = kuyruk.poll();
▶      System.out.print(gecici.veri + " ");
8      if (gecici.sol != null)
9        kuyruk.add(gecici.sol);
10     if (gecici.sag != null)
11       kuyruk.add(gecici.sag);
12   }
13 }
```



şimdi · kuyrukta · yazdırıldı

`poll()` öndeki **5**'i verir ve 5 yazdırılır.

Yaprak: eklenecek çocuk yok.

Adım 9 / 11

DEĞİŞKENLER

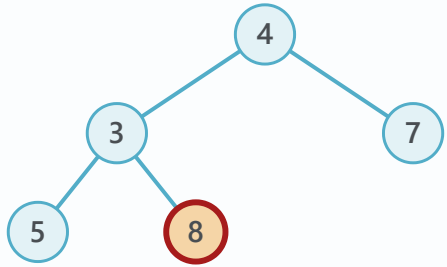
`kuyruk = [8]` `gecici = 5`

ÇIKTI

```
4 3 7 5
```

Kuyruk ile Düzey Sırası Gezinme

```
1  public void seviyeSirasi(AgacDugumu kok) {
2    if (kok == null) return;
3    Queue<AgacDugumu> kuyruk = new LinkedList<>
   ();
4    kuyruk.add(kok);
5    while (!kuyruk.isEmpty()) {
▶      AgacDugumu gecici = kuyruk.poll();
▶      System.out.print(gecici.veri + " ");
8      if (gecici.sol != null)
9        kuyruk.add(gecici.sol);
10     if (gecici.sag != null)
11       kuyruk.add(gecici.sag);
12   }
13 }
```



Kuyruk boş

şimdi kuyrukta yazdırıldı

`poll()` öndeki **8**'i verir ve 8 yazdırılır.

Yaprak: eklenecek çocuk yok.

Adım 10 / 11

DEĞİŞKENLER

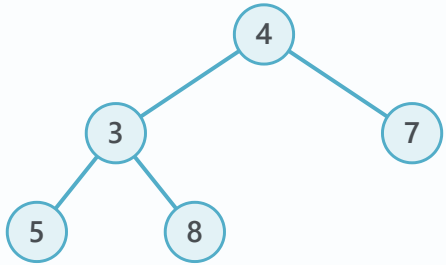
`kuyruk = []` `gecici = 8`

ÇIKTI

```
4 3 7 5 8
```

Kuyruk ile Düzey Sırası Gezinme

```
1  public void seviyeSirasi(AgacDugumu kok) {
2    if (kok == null) return;
3    Queue<AgacDugumu> kuyruk = new LinkedList<>
   ();
4    kuyruk.add(kok);
▶    while (!kuyruk.isEmpty()) {
6      AgacDugumu gecici = kuyruk.poll();
7      System.out.print(gecici.veri + " ");
8      if (gecici.sol != null)
9        kuyruk.add(gecici.sol);
10     if (gecici.sag != null)
11       kuyruk.add(gecici.sag);
12   }
13 }
```



Kuyruk boş

şimdi kuyrukta yazdırıldı

Kuyruk boş → döngü biter. **Çıktı: 4 3 7 5 8** — düzey düzey (kök önde olsaydı: 4 3 5 8 7).

Adım 11 / 11

DEĞİŞKENLER

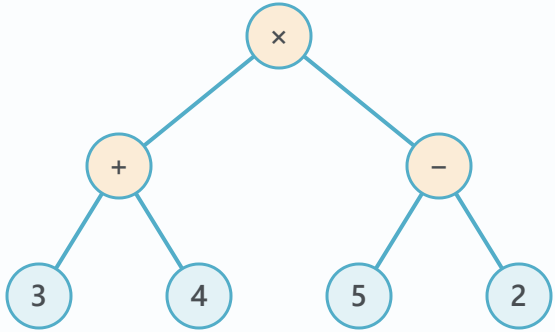
`kuyruk = []` `gecici = 8`

ÇIKTI

```
4 3 7 5 8
```

Uygulama: İfade Ağacı

- İç düğümler **işleçler**, yapraklar **işlenenlerdir**.
- Aynı ağaç, gezinme türüne göre farklı gösterimler verir.
- Değer, kök sonda gezinmeyle hesaplanır: önce iki alt ağacın değeri, sonra işleç.

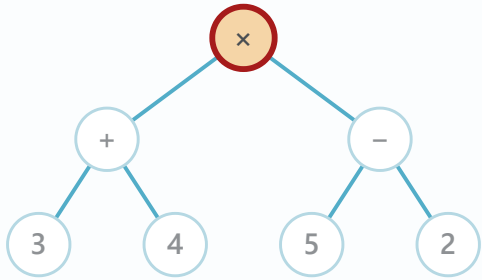


Gezinme	Çıktı	Gösterim
Kök önde	`× + 3 4 − 5 2`	önek (prefix)
Kök ortada	`3 + 4 × 5 − 2` → parantezle: `(3 + 4) × (5 − 2)`	ara ek (infix)
Kök sonda	`3 4 + 5 2 − ×` → 7 × 3 = **21**	sonek (postfix)

Kök ortada çıktısında parantez kaybolur; ifadeyi doğru yazmak için her alt ağaç parantez içine alınmalıdır.

İfade Ağacını Hesaplama: `hesapla()`

```
1  int hesapla(IfadeDugumu d) {
2    if (d.sol == null && d.sag == null)
3      return d.sayi;                // yaprak: işlenen
▶    int a = hesapla(d.sol);
5    int b = hesapla(d.sag);
6    switch (d.islec) {            // iç düğüm: işleç
7      case '+': return a + b;
8      case '-': return a - b;
9      case '*': return a * b;
10     default:  return a / b;
11   }
12 }
```



`hesapla(×)` : iç düğüm (işleç). İşleci uygulamadan önce **sol işlenenin değeri** gerekir → `hesapla(+)` .

Adım 1 / 13

ÇAĞRI YIĞINI

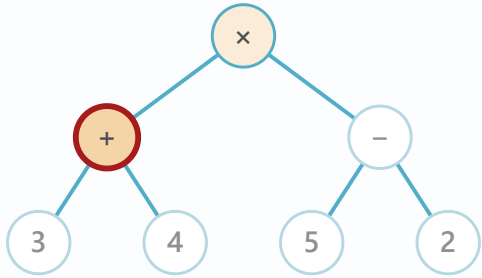
`hesapla(×)`

DEĞİŞKENLER

`d = ×` `a = ?` `b = ?`

İfade Ağacını Hesaplama: `hesapla()`

```
1  int hesapla(IfadeDugumu d) {
2    if (d.sol == null && d.sag == null)
3      return d.sayi;               // yaprak: işlenen
▶    int a = hesapla(d.sol);
5    int b = hesapla(d.sag);
6    switch (d.islec) {             // iç düğüm: işleç
7      case '+': return a + b;
8      case '-': return a - b;
9      case '*': return a * b;
10     default:  return a / b;
11   }
12 }
```



`hesapla(+)` : iç düğüm (işleç). İşleci uygulamadan önce **sol işlenenin değeri** gerekir → `hesapla(3)` .

Adım 2 / 13

ÇAĞRI YIĞINI

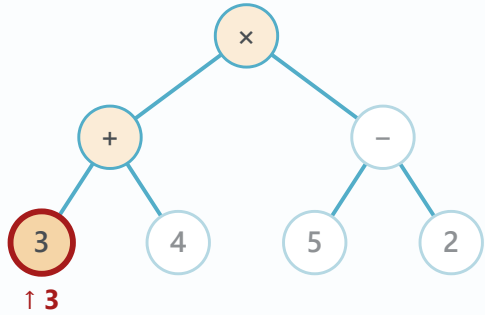
- **`hesapla(+)`**
- `hesapla(×) · s.4`

DEĞİŞKENLER

`d = +` `a = ?` `b = ?`

İfade Ağacını Hesaplama: `hesapla()`

```
1  int hesapla(IfadeDugumu d) {
▶    if (d.sol == null && d.sag == null)
▶      return d.sayi;                // yaprak: işlenen
4    int a = hesapla(d.sol);
5    int b = hesapla(d.sag);
6    switch (d.islec) {            // iç düğüm: işleç
7      case '+': return a + b;
8      case '-': return a - b;
9      case '*': return a * b;
10     default:  return a / b;
11   }
12 }
```



`hesapla(3)` : çocuğu yok → **yaprak**, sayıyı döndürür: **3**.

ÇAĞRI YIĞINI

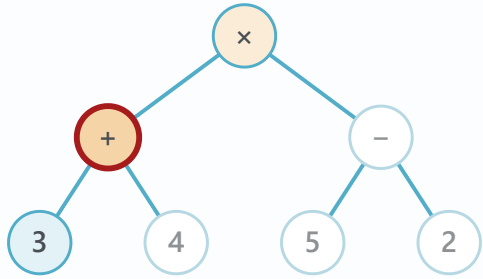
- **`hesapla(3)`**
- `hesapla(+) · s.4`
- `hesapla(×) · s.4`

DEĞİŞKENLER

`d = 3` `dönüş = 3`

İfade Ağacını Hesaplama: `hesapla()`

```
1  int hesapla(IfadeDugumu d) {
2    if (d.sol == null && d.sag == null)
3      return d.sayi;               // yaprak: işlenen
4    int a = hesapla(d.sol);
▶    int b = hesapla(d.sag);
6    switch (d.islec) {             // iç düğüm: işleç
7      case '+': return a + b;
8      case '-': return a - b;
9      case '*': return a * b;
10     default:  return a / b;
11   }
12 }
```



`a = 3` geldi. Şimdi sağ işlenen: `hesapla(4)`.

Adım 4 / 13

ÇAĞRI YIĞINI

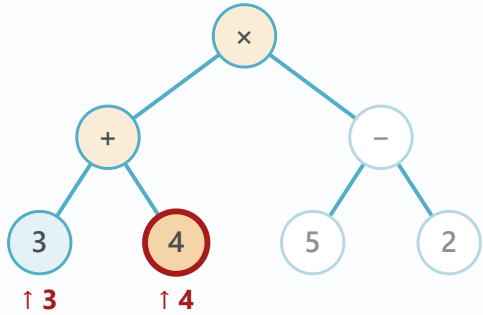
- **`hesapla(+)`**
- `hesapla(×) · s.4`

DEĞİŞKENLER

`d = +` `a = 3` `b = ?`

İfade Ağacını Hesaplama: `hesapla()`

```
1  int hesapla(IfadeDugumu d) {
▶    if (d.sol == null && d.sag == null)
▶      return d.sayi;                // yaprak: işlenen
4    int a = hesapla(d.sol);
5    int b = hesapla(d.sag);
6    switch (d.islec) {              // iç düğüm: işleç
7      case '+': return a + b;
8      case '-': return a - b;
9      case '*': return a * b;
10     default:  return a / b;
11   }
12 }
```



`hesapla(4)` : çocuğu yok → **yaprak**, sayıyı döndürür: **4**.

ÇAĞRI YIĞINI

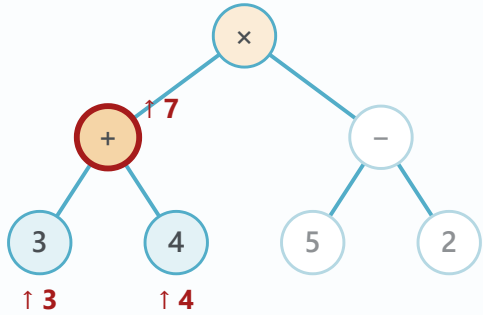
- **`hesapla(4)`**
- `hesapla(+) · s.5`
- `hesapla(×) · s.4`

DEĞİŞKENLER

`d = 4` `dönüş = 4`

İfade Ağacını Hesaplama: `hesapla()`

```
1  int hesapla(IfadeDugumu d) {
2    if (d.sol == null && d.sag == null)
3      return d.sayi;                // yaprak: işlenen
4    int a = hesapla(d.sol);
5    int b = hesapla(d.sag);
▶    switch (d.islec) {            // iç düğüm: işleç
▶      case '+': return a + b;
8      case '-': return a - b;
9      case '*': return a * b;
10     default:  return a / b;
11   }
12 }
```



`b = 4`. İki alt ağaç hazır → işleç uygulanır: 3 + 4 = **7** döner (kök sonda mantığı).

ÇAĞRI YIĞINI

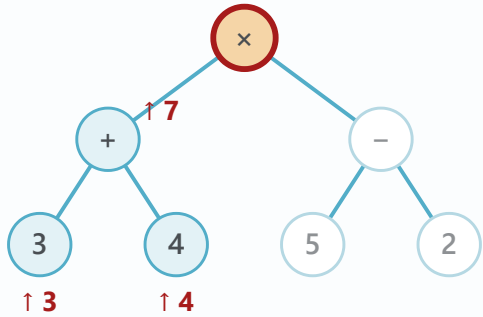
- **hesapla(+)**
- hesapla(×) · s.4

DEĞİŞKENLER

`d = +` `a = 3` `b = 4` `dönüş = 7`

İfade Ağacını Hesaplama: `hesapla()`

```
1  int hesapla(IfadeDugumu d) {
2    if (d.sol == null && d.sag == null)
3      return d.sayi;               // yaprak: işlenen
4    int a = hesapla(d.sol);
▶    int b = hesapla(d.sag);
6    switch (d.islec) {             // iç düğüm: işleç
7      case '+': return a + b;
8      case '-': return a - b;
9      case '*': return a * b;
10     default:  return a / b;
11   }
12 }
```



`a = 7` geldi. Şimdi sağ işlenen: `hesapla(−)`.

Adım 7 / 13

ÇAĞRI YIĞINI

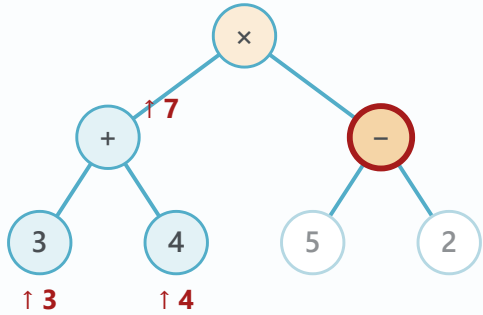
`hesapla(×)`

DEĞİŞKENLER

`d = ×` `a = 7` `b = ?`

İfade Ağacını Hesaplama: `hesapla()`

```
1  int hesapla(IfadeDugumu d) {
2    if (d.sol == null && d.sag == null)
3      return d.sayi;                // yaprak: işlenen
▶    int a = hesapla(d.sol);
5    int b = hesapla(d.sag);
6    switch (d.islec) {            // iç düğüm: işleç
7      case '+': return a + b;
8      case '-': return a - b;
9      case '*': return a * b;
10     default:  return a / b;
11   }
12 }
```



`hesapla(-)` : iç düğüm (işleç). İşleci uygulamadan önce **sol işlenenin değeri** gerekir → `hesapla(5)` .

ÇAĞRI YIĞINI

`hesapla(-)`

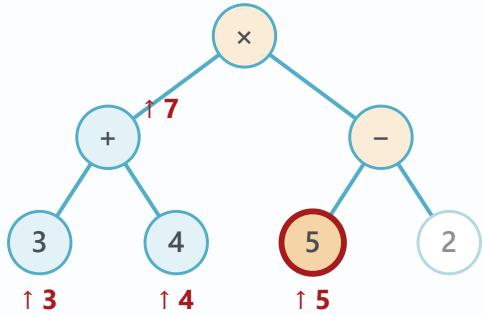
`hesapla(×) · s.5`

DEĞİŞKENLER

`d = −` `a = ?` `b = ?`

İfade Ağacını Hesaplama: `hesapla()`

```
1  int hesapla(IfadeDugumu d) {
▶    if (d.sol == null && d.sag == null)
▶      return d.sayi;                // yaprak: işlenen
4    int a = hesapla(d.sol);
5    int b = hesapla(d.sag);
6    switch (d.islec) {            // iç düğüm: işleç
7      case '+': return a + b;
8      case '-': return a - b;
9      case '*': return a * b;
10     default:  return a / b;
11   }
12 }
```



`hesapla(5)` : çocuğu yok → **yaprak**, sayıyı döndürür: **5**.

ÇAĞRI YIĞINI

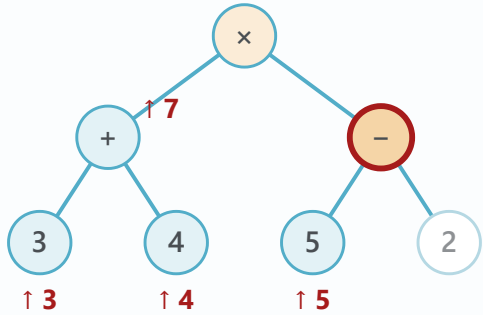
- **`hesapla(5)`**
- `hesapla(−) · s.4`
- `hesapla(×) · s.5`

DEĞİŞKENLER

`d = 5` `dönüş = 5`

İfade Ağacını Hesaplama: `hesapla()`

```
1  int hesapla(IfadeDugumu d) {
2    if (d.sol == null && d.sag == null)
3      return d.sayi;               // yaprak: işlenen
4    int a = hesapla(d.sol);
▶    int b = hesapla(d.sag);
6    switch (d.islec) {             // iç düğüm: işleç
7      case '+': return a + b;
8      case '-': return a - b;
9      case '*': return a * b;
10     default:  return a / b;
11   }
12 }
```



`a = 5` geldi. Şimdi sağ işlenen: `hesapla(2)`.

Adım 10 / 13

ÇAĞRI YIĞINI

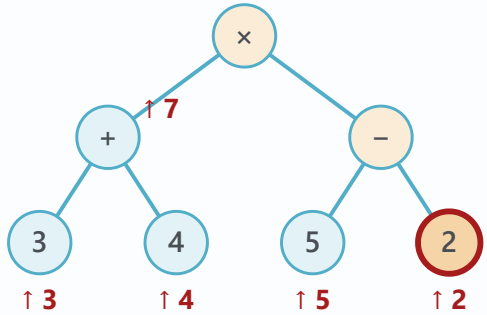
- **`hesapla(−)`**
- `hesapla(×) · s.5`

DEĞİŞKENLER

`d = −` `a = 5` `b = ?`

İfade Ağacını Hesaplama: `hesapla()`

```
1  int hesapla(IfadeDugumu d) {
▶    if (d.sol == null && d.sag == null)
▶      return d.sayi;                // yaprak: işlenen
4    int a = hesapla(d.sol);
5    int b = hesapla(d.sag);
6    switch (d.islec) {            // iç düğüm: işleç
7      case '+': return a + b;
8      case '-': return a - b;
9      case '*': return a * b;
10     default:  return a / b;
11   }
12 }
```



`hesapla(2)` : çocuğu yok → **yaprak**, sayıyı döndürür: **2**.

Adım 11 / 13

ÇAĞRI YIĞINI

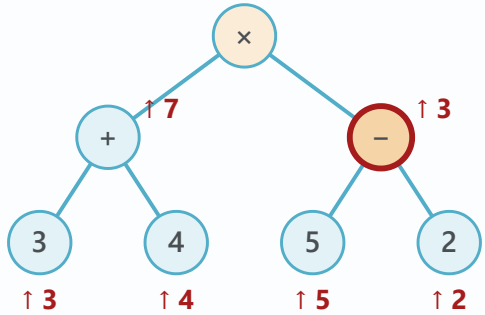
- **`hesapla(2)`**
- `hesapla(−) · s.5`
- `hesapla(×) · s.5`

DEĞİŞKENLER

`d = 2` `dönüş = 2`

İfade Ağacını Hesaplama: `hesapla()`

```
1  int hesapla(IfadeDugumu d) {
2    if (d.sol == null && d.sag == null)
3      return d.sayi;                // yaprak: işlenen
4    int a = hesapla(d.sol);
5    int b = hesapla(d.sag);
▶    switch (d.islec) {            // iç düğüm: işleç
7      case '+': return a + b;
▶      case '-': return a - b;
9      case '*': return a * b;
10     default:  return a / b;
11   }
12 }
```



`b = 2`. İki alt ağaç hazır → işleç uygulanır: 5 − 2 = **3** döner (kök sonda mantığı).

Adım 12 / 13

ÇAĞRI YIĞINI

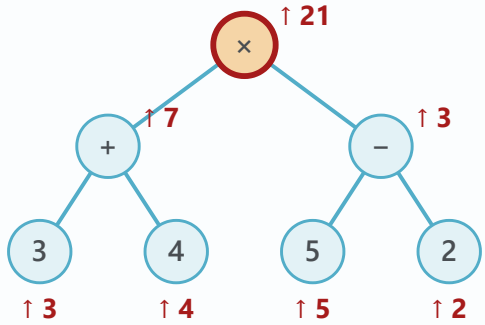
- **hesapla(−)**
- hesapla(×) · s.5

DEĞİŞKENLER

`d = −` `a = 5` `b = 2` `dönüş = 3`

İfade Ağacını Hesaplama: `hesapla()`

```
1  int hesapla(IfadeDugumu d) {
2    if (d.sol == null && d.sag == null)
3      return d.sayi;                // yaprak: işlenen
4    int a = hesapla(d.sol);
5    int b = hesapla(d.sag);
▶    switch (d.islec) {            // iç düğüm: işleç
7      case '+': return a + b;
8      case '-': return a - b;
▶      case '*': return a * b;
10     default:  return a / b;
11   }
12 }
```



`b = 3`. İki alt ağaç hazır → işleç uygulanır: 7 × 3 = **21** döner (kök sonda mantığı). **(3 + 4) × (5 − 2) = 21**

Adım 13 / 13

ÇAĞRI YIĞINI

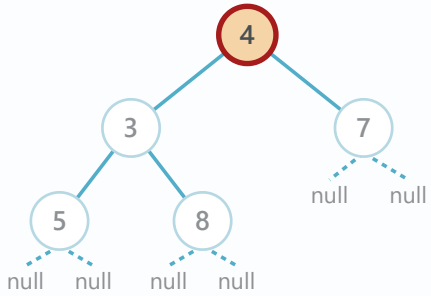
`hesapla(×)`

DEĞİŞKENLER

`d = ×` `a = 7` `b = 3` `dönüş = 21`

İkili Ağaçta En Büyük Değeri Bulma

```
1  public int enBuyukBul(AgacDugumu kok) {
2    if (kok == null) {
3      return Integer.MIN_VALUE;
4    }
▶    int sonuc = kok.veri;
▶    int sol = enBuyukBul(kok.sol);
7    int sag = enBuyukBul(kok.sag);
8    if (sol > sonuc)
9      sonuc = sol;
10   if (sag > sonuc)
11     sonuc = sag;
12   return sonuc;
13 }
```



`enBuyukBul(4)` : **sonuc = 4**; sol alt ağacın en büyüğü sorulur.

Adım 1 / 21

ÇAĞRI YIĞINI

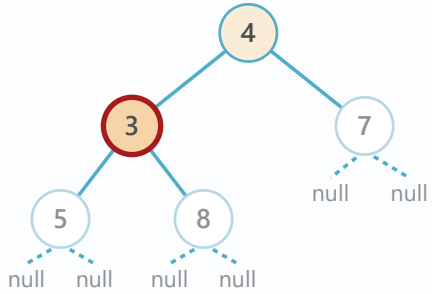
`enBuyukBul(4)`

DEĞİŞKENLER

`kok = 4` `sonuc = 4` `sol = ?` `sag = ?`

İkili Ağaçta En Büyük Değeri Bulma

```
1  public int enBuyukBul(AgacDugumu kok) {
2    if (kok == null) {
3      return Integer.MIN_VALUE;
4    }
▶    int sonuc = kok.veri;
▶    int sol = enBuyukBul(kok.sol);
7    int sag = enBuyukBul(kok.sag);
8    if (sol > sonuc)
9      sonuc = sol;
10   if (sag > sonuc)
11     sonuc = sag;
12   return sonuc;
13 }
```



`enBuyukBul(3)` : **sonuc = 3**; sol alt ağacın en büyüğü sorulur.

Adım 2 / 21

ÇAĞRI YIĞINI

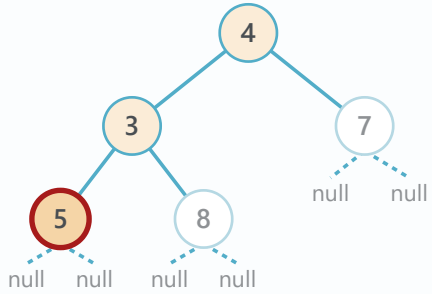
- **enBuyukBul(3)**
- enBuyukBul(4) · s.6

DEĞİŞKENLER

`kok = 3` `sonuc = 3` `sol = ?` `sag = ?`

İkili Ağaçta En Büyük Değeri Bulma

```
1  public int enBuyukBul(AgacDugumu kok) {
2    if (kok == null) {
3      return Integer.MIN_VALUE;
4    }
▶    int sonuc = kok.veri;
▶    int sol = enBuyukBul(kok.sol);
7    int sag = enBuyukBul(kok.sag);
8    if (sol > sonuc)
9      sonuc = sol;
10   if (sag > sonuc)
11     sonuc = sag;
12   return sonuc;
13 }
```



`enBuyukBul(5)` : **sonuc = 5**; sol alt ağacın en büyüğü sorulur.

Adım 3 / 21

ÇAĞRI YIĞINI

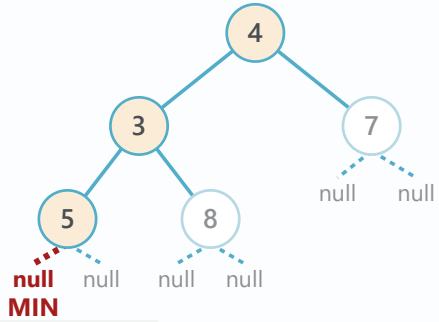
- **enBuyukBul(5)**
- enBuyukBul(3) · s.6
- enBuyukBul(4) · s.6

DEĞİŞKENLER

`kok = 5` `sonuc = 5` `sol = ?` `sag = ?`

İkili Ağaçta En Büyük Değeri Bulma

```
1  public int enBuyukBul(AgacDugumu kok) {
2    if (kok == null) {
▶      return Integer.MIN_VALUE;
4    }
5    int sonuc = kok.veri;
6    int sol = enBuyukBul(kok.sol);
7    int sag = enBuyukBul(kok.sag);
8    if (sol > sonuc)
9      sonuc = sol;
10   if (sag > sonuc)
11     sonuc = sag;
12   return sonuc;
13 }
```



`enBuyukBul(5.sol)` : boş alt ağaç → **Integer.MIN_VALUE** döner; hiçbir değerden büyük olamaz, karşılaştırmayı bozmaz.

Adım 4 / 21

ÇAĞRI YIĞINI

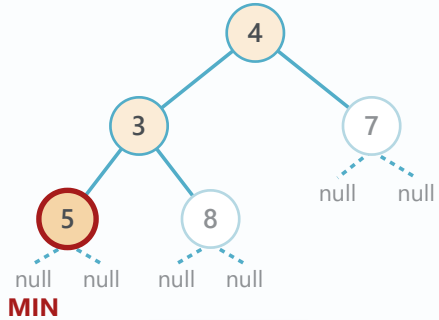
- **enBuyukBul(null)**
- enBuyukBul(5) · s.6
- enBuyukBul(3) · s.6
- enBuyukBul(4) · s.6

DEĞİŞKENLER

`kok = null` `dönüş = MIN`

İkili Ağaçta En Büyük Değeri Bulma

```
1  public int enBuyukBul(AgacDugumu kok) {
2    if (kok == null) {
3      return Integer.MIN_VALUE;
4    }
5    int sonuc = kok.veri;
6    int sol = enBuyukBul(kok.sol);
▶    int sag = enBuyukBul(kok.sag);
8    if (sol > sonuc)
9      sonuc = sol;
10   if (sag > sonuc)
11     sonuc = sag;
12   return sonuc;
13 }
```



Sol sonuç **MIN** geldi (`sol = MIN`); şimdi sağ alt ağaç.

Adım 5 / 21

ÇAĞRI YIĞINI

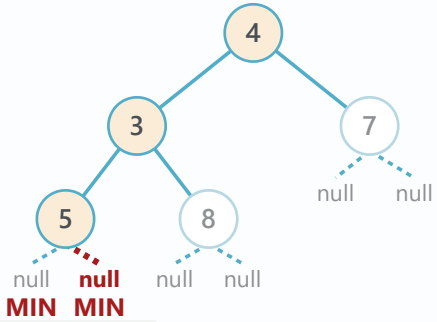
- **enBuyukBul(5)**
- enBuyukBul(3) · s.6
- enBuyukBul(4) · s.6

DEĞİŞKENLER

`kok = 5` `sonuc = 5` `sol = MIN` `sag = ?`

İkili Ağaçta En Büyük Değeri Bulma

```
1  public int enBuyukBul(AgacDugumu kok) {
2    if (kok == null) {
▶      return Integer.MIN_VALUE;
4    }
5    int sonuc = kok.veri;
6    int sol = enBuyukBul(kok.sol);
7    int sag = enBuyukBul(kok.sag);
8    if (sol > sonuc)
9      sonuc = sol;
10   if (sag > sonuc)
11     sonuc = sag;
12   return sonuc;
13 }
```



`enBuyukBul(5.sag)` : boş alt ağaç → **Integer.MIN_VALUE** döner; hiçbir değerden büyük olamaz, karşılaştırmayı bozmaz.

Adım 6 / 21

ÇAĞRI YIĞINI

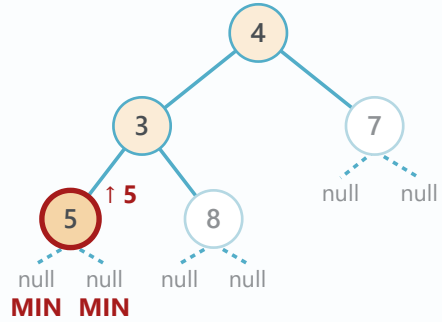
- **enBuyukBul(null)**
- enBuyukBul(5) · s.7
- enBuyukBul(3) · s.6
- enBuyukBul(4) · s.6

DEĞİŞKENLER

`kok = null` `dönüş = MIN`

İkili Ağaçta En Büyük Değeri Bulma

```
1  public int enBuyukBul(AgacDugumu kok) {
2    if (kok == null) {
3      return Integer.MIN_VALUE;
4    }
5    int sonuc = kok.veri;
6    int sol = enBuyukBul(kok.sol);
7    int sag = enBuyukBul(kok.sag);
▶    if (sol > sonuc)
▶      sonuc = sol;
▶    if (sag > sonuc)
▶      sonuc = sag;
▶    return sonuc;
13 }
```



Karşılaştırmalar: sol = MIN ≤ 5; sag = MIN ≤ 5. **return 5**.

ÇAĞRI YIĞINI

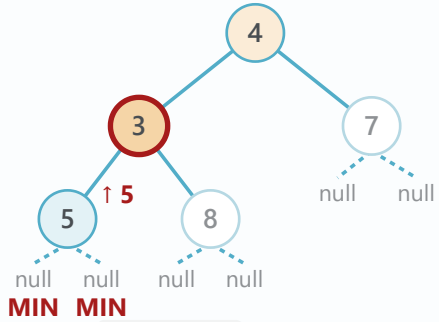
- **enBuyukBul(5)**
- enBuyukBul(3) · s.6
- enBuyukBul(4) · s.6

DEĞİŞKENLER

`kok = 5` `sonuc = 5` `sol = MIN` `sag = MIN`

İkili Ağaçta En Büyük Değeri Bulma

```
1  public int enBuyukBul(AgacDugumu kok) {
2    if (kok == null) {
3      return Integer.MIN_VALUE;
4    }
5    int sonuc = kok.veri;
6    int sol = enBuyukBul(kok.sol);
▶    int sag = enBuyukBul(kok.sag);
8    if (sol > sonuc)
9      sonuc = sol;
10   if (sag > sonuc)
11     sonuc = sag;
12   return sonuc;
13 }
```



Sol sonuç **5** geldi (`sol = 5`); şimdi sağ alt ağaç.

ÇAĞRI YIĞINI

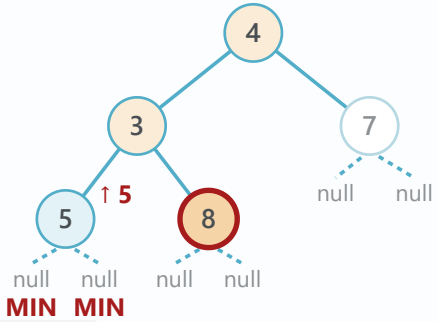
- **enBuyukBul(3)**
- enBuyukBul(4) · s.6

DEĞİŞKENLER

`kok = 3` `sonuc = 3` `sol = 5` `sag = ?`

İkili Ağaçta En Büyük Değeri Bulma

```
1  public int enBuyukBul(AgacDugumu kok) {
2    if (kok == null) {
3      return Integer.MIN_VALUE;
4    }
▶    int sonuc = kok.veri;
▶    int sol = enBuyukBul(kok.sol);
7    int sag = enBuyukBul(kok.sag);
8    if (sol > sonuc)
9      sonuc = sol;
10   if (sag > sonuc)
11     sonuc = sag;
12   return sonuc;
13 }
```



`enBuyukBul(8)` : **sonuc = 8**; sol alt ağacın en büyüğü sorulur.

ÇAĞRI YIĞINI

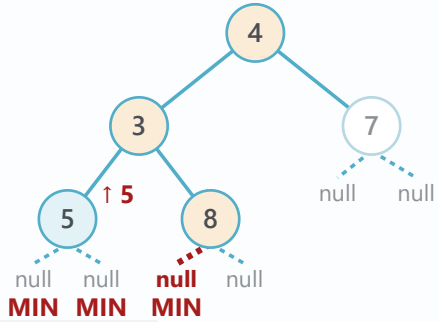
- **enBuyukBul(8)**
- enBuyukBul(3) · s.7
- enBuyukBul(4) · s.6

DEĞİŞKENLER

`kok = 8` `sonuc = 8` `sol = ?` `sag = ?`

İkili Ağaçta En Büyük Değeri Bulma

```
1  public int enBuyukBul(AgacDugumu kok) {
2    if (kok == null) {
▶      return Integer.MIN_VALUE;
4    }
5    int sonuc = kok.veri;
6    int sol = enBuyukBul(kok.sol);
7    int sag = enBuyukBul(kok.sag);
8    if (sol > sonuc)
9      sonuc = sol;
10   if (sag > sonuc)
11     sonuc = sag;
12   return sonuc;
13 }
```



`enBuyukBul(8.sol)` : boş alt ağaç → **Integer.MIN_VALUE** döner; hiçbir değerden büyük olamaz, karşılaştırmayı bozmaz.

Adım 10 / 21

ÇAĞRI YIĞINI

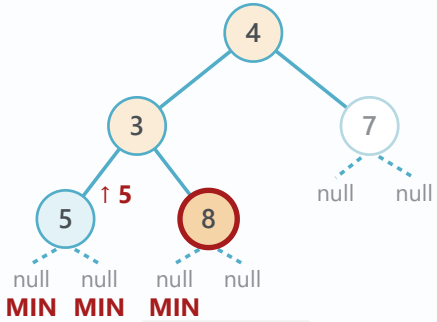
- **enBuyukBul(null)**
- enBuyukBul(8) · s.6
- enBuyukBul(3) · s.7
- enBuyukBul(4) · s.6

DEĞİŞKENLER

`kok = null` `dönüş = MIN`

İkili Ağaçta En Büyük Değeri Bulma

```
1  public int enBuyukBul(AgacDugumu kok) {
2    if (kok == null) {
3      return Integer.MIN_VALUE;
4    }
5    int sonuc = kok.veri;
6    int sol = enBuyukBul(kok.sol);
▶    int sag = enBuyukBul(kok.sag);
8    if (sol > sonuc)
9      sonuc = sol;
10   if (sag > sonuc)
11     sonuc = sag;
12   return sonuc;
13 }
```



Sol sonuç **MIN** geldi (`sol = MIN`); şimdi sağ alt ağaç.

ÇAĞRI YIĞINI

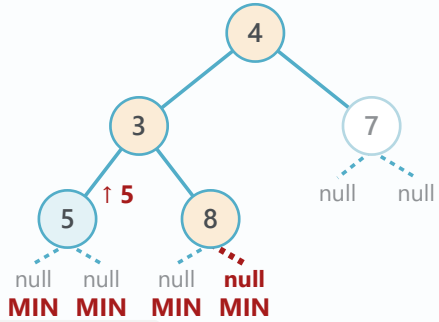
- **enBuyukBul(8)**
- enBuyukBul(3) · s.7
- enBuyukBul(4) · s.6

DEĞİŞKENLER

`kok = 8` `sonuc = 8` `sol = MIN` `sag = ?`

İkili Ağaçta En Büyük Değeri Bulma

```
1  public int enBuyukBul(AgacDugumu kok) {
2    if (kok == null) {
▶      return Integer.MIN_VALUE;
4    }
5    int sonuc = kok.veri;
6    int sol = enBuyukBul(kok.sol);
7    int sag = enBuyukBul(kok.sag);
8    if (sol > sonuc)
9      sonuc = sol;
10   if (sag > sonuc)
11     sonuc = sag;
12   return sonuc;
13 }
```



`enBuyukBul(8.sag)` : boş alt ağaç → **Integer.MIN_VALUE** döner; hiçbir değerden büyük olamaz, karşılaştırmayı bozmaz.

Adım 12 / 21

ÇAĞRI YIĞINI

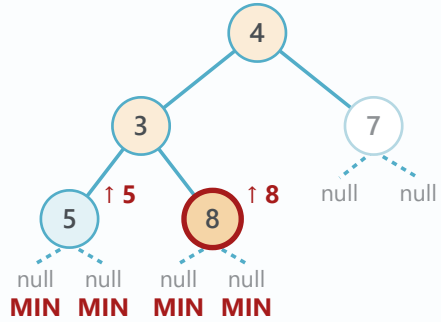
- **enBuyukBul(null)**
- enBuyukBul(8) · s.7
- enBuyukBul(3) · s.7
- enBuyukBul(4) · s.6

DEĞİŞKENLER

`kok = null` `dönüş = MIN`

İkili Ağaçta En Büyük Değeri Bulma

```
1  public int enBuyukBul(AgacDugumu kok) {
2    if (kok == null) {
3      return Integer.MIN_VALUE;
4    }
5    int sonuc = kok.veri;
6    int sol = enBuyukBul(kok.sol);
7    int sag = enBuyukBul(kok.sag);
▶    if (sol > sonuc)
▶      sonuc = sol;
▶    if (sag > sonuc)
▶      sonuc = sag;
▶    return sonuc;
13 }
```



Karşılaştırmalar: sol = MIN ≤ 8; sag = MIN ≤ 8. **return 8**.

Adım 13 / 21

ÇAĞRI YIĞINI

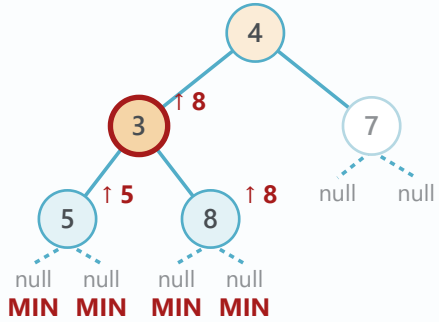
- `enBuyukBul(8)`
- `enBuyukBul(3) · s.7`
- `enBuyukBul(4) · s.6`

DEĞİŞKENLER

`kok = 8` `sonuc = 8` `sol = MIN` `sag = MIN`

İkili Ağaçta En Büyük Değeri Bulma

```
1  public int enBuyukBul(AgacDugumu kok) {
2    if (kok == null) {
3      return Integer.MIN_VALUE;
4    }
5    int sonuc = kok.veri;
6    int sol = enBuyukBul(kok.sol);
7    int sag = enBuyukBul(kok.sag);
▶    if (sol > sonuc)
▶      sonuc = sol;
▶    if (sag > sonuc)
▶      sonuc = sag;
▶    return sonuc;
13 }
```



Karşılaştırmalar: sol = 5 > 3 → sonuc = 5; sag = 8 > 5 → sonuc = 8. **return 8**.

Adım 14 / 21

ÇAĞRI YIĞINI

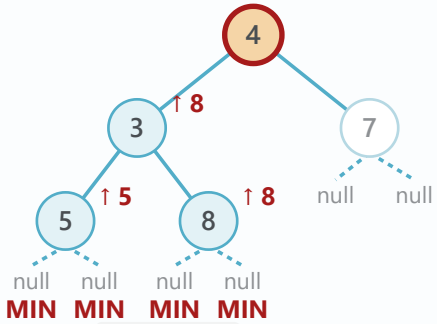
- **enBuyukBul(3)**
- enBuyukBul(4) · s.6

DEĞİŞKENLER

`kok = 3` `sonuc = 8` `sol = 5` `sag = 8`

İkili Ağaçta En Büyük Değeri Bulma

```
1  public int enBuyukBul(AgacDugumu kok) {
2    if (kok == null) {
3      return Integer.MIN_VALUE;
4    }
5    int sonuc = kok.veri;
6    int sol = enBuyukBul(kok.sol);
▶   int sag = enBuyukBul(kok.sag);
8    if (sol > sonuc)
9      sonuc = sol;
10   if (sag > sonuc)
11     sonuc = sag;
12   return sonuc;
13 }
```



Sol sonuç **8** geldi (`sol = 8`); şimdi sağ alt ağaç.

Adım 15 / 21

ÇAĞRI YIĞINI

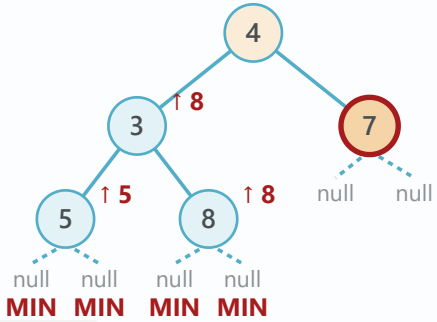
`enBuyukBul(4)`

DEĞİŞKENLER

`kok = 4` `sonuc = 4` `sol = 8` `sag = ?`

İkili Ağaçta En Büyük Değeri Bulma

```
1  public int enBuyukBul(AgacDugumu kok) {
2    if (kok == null) {
3      return Integer.MIN_VALUE;
4    }
▶    int sonuc = kok.veri;
▶    int sol = enBuyukBul(kok.sol);
7    int sag = enBuyukBul(kok.sag);
8    if (sol > sonuc)
9      sonuc = sol;
10   if (sag > sonuc)
11     sonuc = sag;
12   return sonuc;
13 }
```



`enBuyukBul(7)` : **sonuc = 7**; sol alt ağacın en büyüğü sorulur.

Adım 16 / 21

ÇAĞRI YIĞINI

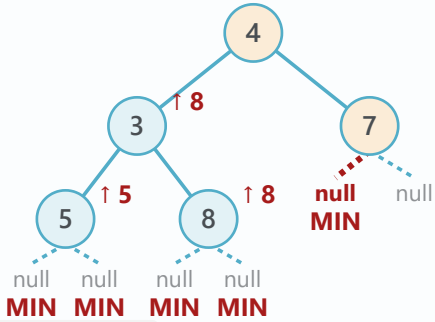
- **enBuyukBul(7)**
- enBuyukBul(4) · s.7

DEĞİŞKENLER

`kok = 7` `sonuc = 7` `sol = ?` `sag = ?`

İkili Ağaçta En Büyük Değeri Bulma

```
1  public int enBuyukBul(AgacDugumu kok) {
2    if (kok == null) {
▶      return Integer.MIN_VALUE;
4    }
5    int sonuc = kok.veri;
6    int sol = enBuyukBul(kok.sol);
7    int sag = enBuyukBul(kok.sag);
8    if (sol > sonuc)
9      sonuc = sol;
10   if (sag > sonuc)
11     sonuc = sag;
12   return sonuc;
13 }
```



`enBuyukBul(7.sol)` : boş alt ağaç → **Integer.MIN_VALUE** döner; hiçbir değerden büyük olamaz, karşılaştırmayı bozmaz.

Adım 17 / 21

ÇAĞRI YIĞINI

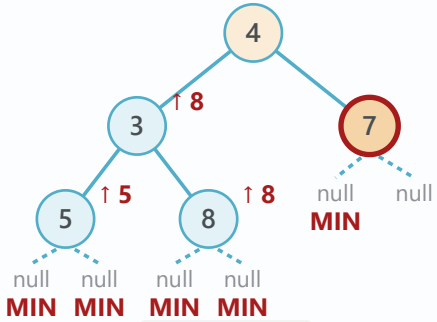
- **`enBuyukBul(null)`**
- `enBuyukBul(7) · s.6`
- `enBuyukBul(4) · s.7`

DEĞİŞKENLER

`kok = null` `dönüş = MIN`

İkili Ağaçta En Büyük Değeri Bulma

```
1  public int enBuyukBul(AgacDugumu kok) {
2    if (kok == null) {
3      return Integer.MIN_VALUE;
4    }
5    int sonuc = kok.veri;
6    int sol = enBuyukBul(kok.sol);
▶    int sag = enBuyukBul(kok.sag);
8    if (sol > sonuc)
9      sonuc = sol;
10   if (sag > sonuc)
11     sonuc = sag;
12   return sonuc;
13 }
```



Sol sonuç **MIN** geldi (`sol = MIN`); şimdi sağ alt ağaç.

ÇAĞRI YIĞINI

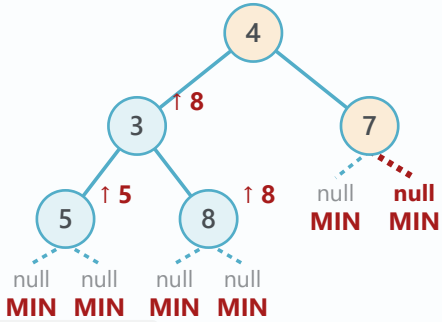
- **enBuyukBul(7)**
- enBuyukBul(4) · s.7

DEĞİŞKENLER

`kok = 7` `sonuc = 7` `sol = MIN` `sag = ?`

İkili Ağaçta En Büyük Değeri Bulma

```
1  public int enBuyukBul(AgacDugumu kok) {
2    if (kok == null) {
▶      return Integer.MIN_VALUE;
4    }
5    int sonuc = kok.veri;
6    int sol = enBuyukBul(kok.sol);
7    int sag = enBuyukBul(kok.sag);
8    if (sol > sonuc)
9      sonuc = sol;
10   if (sag > sonuc)
11     sonuc = sag;
12   return sonuc;
13 }
```



`enBuyukBul(7.sag)` : boş alt ağaç → **Integer.MIN_VALUE** döner; hiçbir değerden büyük olamaz, karşılaştırmayı bozmaz.

Adım 19 / 21

ÇAĞRI YIĞINI

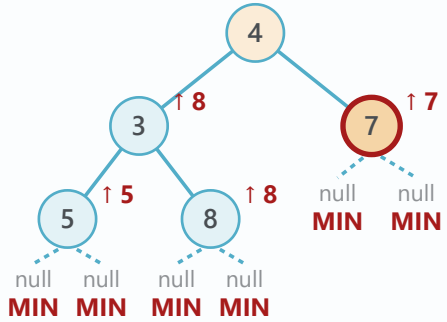
- **enBuyukBul(null)**
- enBuyukBul(7) · s.7
- enBuyukBul(4) · s.7

DEĞİŞKENLER

`kok = null` `dönüş = MIN`

İkili Ağaçta En Büyük Değeri Bulma

```
1  public int enBuyukBul(AgacDugumu kok) {
2    if (kok == null) {
3      return Integer.MIN_VALUE;
4    }
5    int sonuc = kok.veri;
6    int sol = enBuyukBul(kok.sol);
7    int sag = enBuyukBul(kok.sag);
▶    if (sol > sonuc)
▶      sonuc = sol;
▶    if (sag > sonuc)
▶      sonuc = sag;
▶    return sonuc;
13 }
```



Karşılaştırmalar: sol = MIN ≤ 7; sag = MIN ≤ 7. **return 7**.

Adım 20 / 21

ÇAĞRI YIĞINI

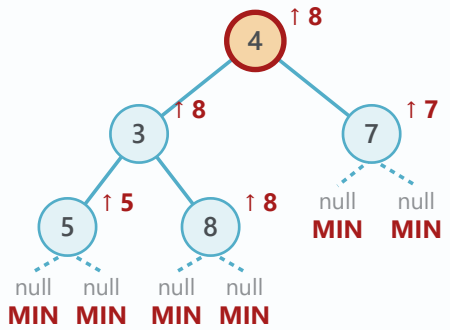
- **enBuyukBul(7)**
- enBuyukBul(4) · s.7

DEĞİŞKENLER

`kok = 7` `sonuc = 7` `sol = MIN` `sag = MIN`

İkili Ağaçta En Büyük Değeri Bulma

```
1  public int enBuyukBul(AgacDugumu kok) {
2    if (kok == null) {
3      return Integer.MIN_VALUE;
4    }
5    int sonuc = kok.veri;
6    int sol = enBuyukBul(kok.sol);
7    int sag = enBuyukBul(kok.sag);
▶    if (sol > sonuc)
▶      sonuc = sol;
▶    if (sag > sonuc)
▶      sonuc = sag;
▶    return sonuc;
13 }
```



Karşılaştırmalar: sol = 8 > 4 → sonuc = 8; sag = 7 ≤ 8. **return 8**. **Sonuç: 8**

Adım 21 / 21

ÇAĞRI YIĞINI

`enBuyukBul(4)`

DEĞİŞKENLER

`kok = 4` `sonuc = 8` `sol = 8` `sag = 7`

Özyinelemeli Ağaç Fonksiyonları

```
int boyut(AgacDugumu k) {            // düğüm sayısı
    if (k == null) return 0;
    return 1 + boyut(k.sol) + boyut(k.sag);
}

int yukseklik(AgacDugumu k) {        // kenar sayısı
    if (k == null) return -1;        // boş ağaç: -1
    return 1 + Math.max(yukseklik(k.sol),
                        yukseklik(k.sag));
}

int yaprakSayisi(AgacDugumu k) {
    if (k == null) return 0;
    if (k.sol == null && k.sag == null) return 1;
    return yaprakSayisi(k.sol) + yaprakSayisi(k.sag);
}
```

Ortak kalıp

Boş ağaç için taban durum; aksi halde iki alt ağacın sonucunu birleştir (kök sonda mantığı).

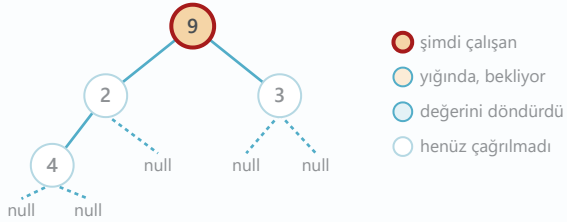
9 → (2 → 4), 3 ağacında

```
boyut = 4
yukseklik = 2
yaprakSayisi = 2 (4 ve 3)
```

Maliyet

Her düğüm bir kez işlenir: O(n) zaman, O(h) yığın alanı.

Kod İzleme: `boyut()` — değerler yukarı taşınır



`boyut(9)` (kök): null değil → önce **sol alt ağaç** için `boyut(2)` çağrılır.

Adım 1 / 17

```
1 int boyut(AgacDugumu k) {            // düğüm sayısı
2     if (k == null) return 0;
▶     return 1 + boyut(k.sol) + boyut(k.sag);
4 }
```

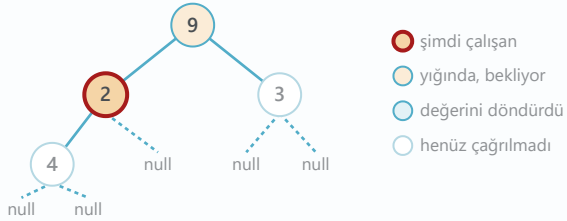
ÇAĞRI YIĞINI

`boyut(9)`

DEĞİŞKENLER

`k = 9` `sol = ?` `sag = ?`

Kod İzleme: `boyut()` — değerler yukarı taşınır



`boyut(2)` (9'un sol çocuğu): null değil → önce **sol alt ağaç** için `boyut(4)` çağrılır.

Adım 2 / 17

```
1 int boyut(AgacDugumu k) {            // düğüm sayısı
2     if (k == null) return 0;
▶     return 1 + boyut(k.sol) + boyut(k.sag);
4 }
```

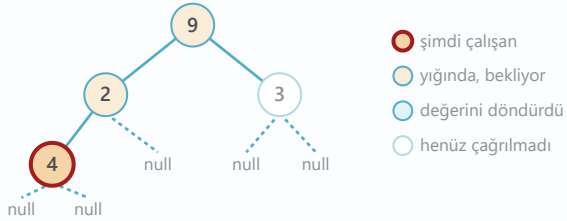
ÇAĞRI YIĞINI

- `boyut(2)`
- `boyut(9) · s.3`

DEĞİŞKENLER

`k = 2` `sol = ?` `sag = ?`

Kod İzleme: `boyut()` — değerler yukarı taşınır



`boyut(4)` (2'nin sol çocuğu): null değil → önce **sol alt ağaç** için `boyut(null)` çağrılır.

Adım 3 / 17

```
1 int boyut(AgacDugumu k) {            // düğüm sayısı
2     if (k == null) return 0;
▶     return 1 + boyut(k.sol) + boyut(k.sag);
4 }
```

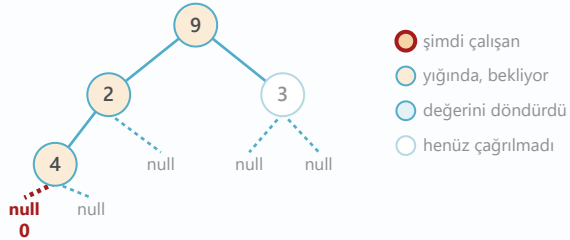
ÇAĞRI YIĞINI

- `boyut(4)`
- `boyut(2) · s.3`
- `boyut(9) · s.3`

DEĞİŞKENLER

`k = 4` `sol = ?` `sag = ?`

Kod İzleme: `boyut()` — değerler yukarı taşınır



`boyut(4.sol)` = `boyut(null)` : boş ağaç → **return 0**.

Adım 4 / 17

```
1 int boyut(AgacDugumu k) {            // düğüm sayısı
▶     if (k == null) return 0;
3     return 1 + boyut(k.sol) + boyut(k.sag);
4 }
```

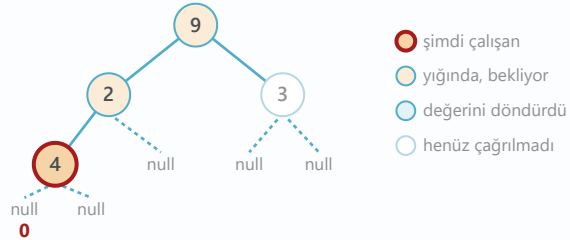
ÇAĞRI YIĞINI

- **`boyut(null)`**
- `boyut(4) · s.3`
- `boyut(2) · s.3`
- `boyut(9) · s.3`

DEĞİŞKENLER

`k = null` `dönüş = 0`

Kod İzleme: `boyut()` — değerler yukarı taşınır



Sol sonuç **0** geldi. Şimdi sağ alt ağaç: `boyut(null)`.

Adım 5 / 17

```
1 int boyut(AgacDugumu k) {            // düğüm sayısı
2     if (k == null) return 0;
▶     return 1 + boyut(k.sol) + boyut(k.sag);
4 }
```

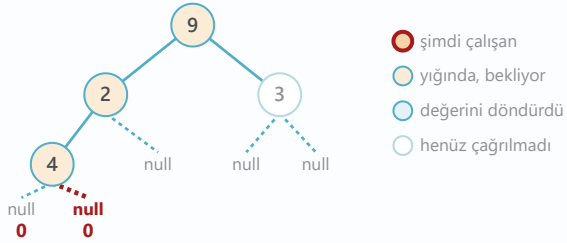
ÇAĞRI YIĞINI

- **`boyut(4)`**
- `boyut(2) · s.3`
- `boyut(9) · s.3`

DEĞİŞKENLER

`k = 4` `sol = 0` `sag = ?`

Kod İzleme: `boyut()` — değerler yukarı taşınır



`boyut(4.sag)` = `boyut(null)` : boş ağaç → **return 0**.

Adım 6 / 17

```
1 int boyut(AgacDugumu k) {            // düğüm sayısı
▶     if (k == null) return 0;
3     return 1 + boyut(k.sol) + boyut(k.sag);
4 }
```

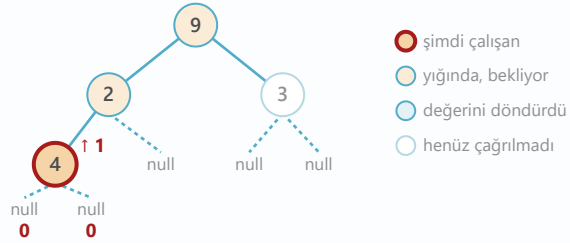
ÇAĞRI YIĞINI

- **`boyut(null)`**
- `boyut(4) · s.3`
- `boyut(2) · s.3`
- `boyut(9) · s.3`

DEĞİŞKENLER

`k = null` `dönüş = 0`

Kod İzleme: `boyut()` — değerler yukarı taşınır



`boyut(4)` = 1 + 0 + 0 = **1** döner. Değer 2'nin çerçevesine taşınır.

Adım 7 / 17

```
1 int boyut(AgacDugumu k) {            // düğüm sayısı
2     if (k == null) return 0;
▶     return 1 + boyut(k.sol) + boyut(k.sag);
4 }
```

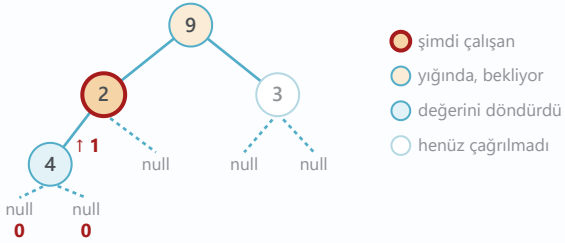
ÇAĞRI YIĞINI

- **boyut(4)**
- boyut(2) · s.3
- boyut(9) · s.3

DEĞİŞKENLER

`k = 4` `sol = 0` `sag = 0` `dönüş = 1`

Kod İzleme: `boyut()` — değerler yukarı taşınır



Sol sonuç **1** geldi. Şimdi sağ alt ağaç: `boyut(null)`.

Adım 8 / 17

```
1 int boyut(AgacDugumu k) {            // düğüm sayısı
2     if (k == null) return 0;
▶     return 1 + boyut(k.sol) + boyut(k.sag);
4 }
```

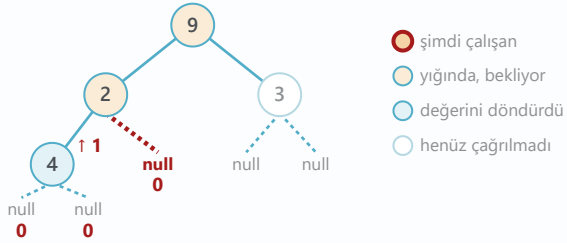
ÇAĞRI YIĞINI

- **boyut(2)**
- boyut(9) · s.3

DEĞİŞKENLER

`k = 2` `sol = 1` `sag = ?`

Kod İzleme: `boyut()` — değerler yukarı taşınır



`boyut(2.sag)` = `boyut(null)` : boş ağaç → **return 0**.

Adım 9 / 17

```
1 int boyut(AgacDugumu k) {            // düğüm sayısı
▶     if (k == null) return 0;
3     return 1 + boyut(k.sol) + boyut(k.sag);
4 }
```

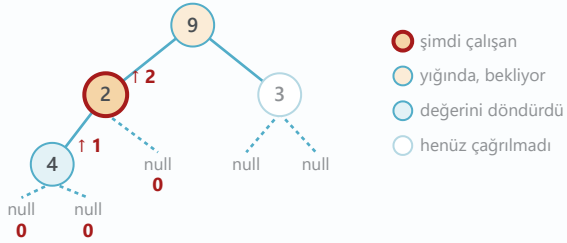
ÇAĞRI YIĞINI

- **`boyut(null)`**
- `boyut(2) · s.3`
- `boyut(9) · s.3`

DEĞİŞKENLER

`k = null` `dönüş = 0`

Kod İzleme: `boyut()` — değerler yukarı taşınır



`boyut(2)` = 1 + 1 + 0 = **2** döner. Değer 9'un çerçevesine taşınır.

Adım 10 / 17

```
1 int boyut(AgacDugumu k) {            // düğüm sayısı
2     if (k == null) return 0;
▶     return 1 + boyut(k.sol) + boyut(k.sag);
4 }
```

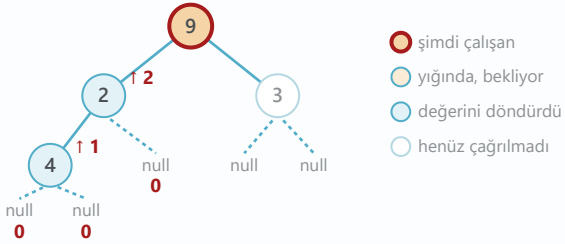
ÇAĞRI YIĞINI

- **boyut(2)**
- boyut(9) · s.3

DEĞİŞKENLER

`k = 2` `sol = 1` `sag = 0` `dönüş = 2`

Kod İzleme: `boyut()` — değerler yukarı taşınır



Sol sonuç **2** geldi. Şimdi sağ alt ağaç: `boyut(3)`.

Adım 11 / 17

```
1 int boyut(AgacDugumu k) {            // düğüm sayısı
2     if (k == null) return 0;
▶     return 1 + boyut(k.sol) + boyut(k.sag);
4 }
```

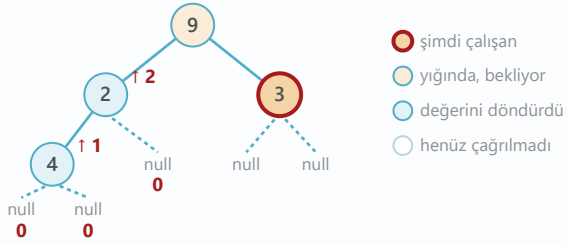
ÇAĞRI YIĞINI

`boyut(9)`

DEĞİŞKENLER

`k = 9` `sol = 2` `sag = ?`

Kod İzleme: `boyut()` — değerler yukarı taşınır



`boyut(3)` (9'un sağ çocuğu): null değil → önce **sol alt ağaç** için `boyut(null)` çağrılır.

Adım 12 / 17

```
1 int boyut(AgacDugumu k) {            // düğüm sayısı
2     if (k == null) return 0;
▶     return 1 + boyut(k.sol) + boyut(k.sag);
4 }
```

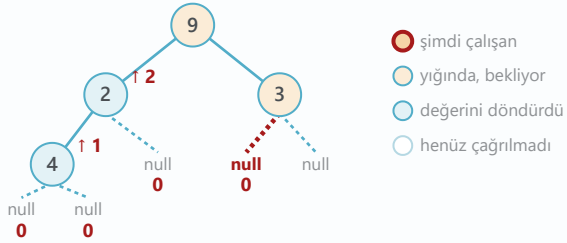
ÇAĞRI YIĞINI

- **boyut(3)**
- boyut(9) · s.3

DEĞİŞKENLER

`k = 3` `sol = ?` `sag = ?`

Kod İzleme: `boyut()` — değerler yukarı taşınır



`boyut(3.sol)` = `boyut(null)` : boş ağaç → **return 0**.

Adım 13 / 17

```
1 int boyut(AgacDugumu k) {            // düğüm sayısı
▶     if (k == null) return 0;
3     return 1 + boyut(k.sol) + boyut(k.sag);
4 }
```

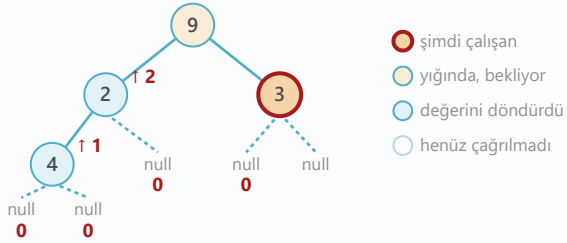
ÇAĞRI YIĞINI

- **`boyut(null)`**
- `boyut(3) · s.3`
- `boyut(9) · s.3`

DEĞİŞKENLER

`k = null` `dönüş = 0`

Kod İzleme: `boyut()` — değerler yukarı taşınır



Sol sonuç **0** geldi. Şimdi sağ alt ağaç: `boyut(null)`.

Adım 14 / 17

```
1 int boyut(AgacDugumu k) {            // düğüm sayısı
2     if (k == null) return 0;
▶     return 1 + boyut(k.sol) + boyut(k.sag);
4 }
```

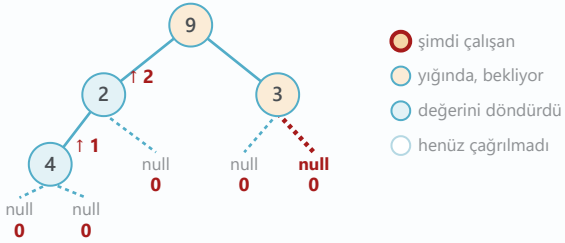
ÇAĞRI YIĞINI

- **boyut(3)**
- boyut(9) · s.3

DEĞİŞKENLER

`k = 3` `sol = 0` `sag = ?`

Kod İzleme: `boyut()` — değerler yukarı taşınır



`boyut(3.sag)` = `boyut(null)` : boş ağaç → **return 0**.

Adım 15 / 17

```
1 int boyut(AgacDugumu k) {            // düğüm sayısı
▶     if (k == null) return 0;
3     return 1 + boyut(k.sol) + boyut(k.sag);
4 }
```

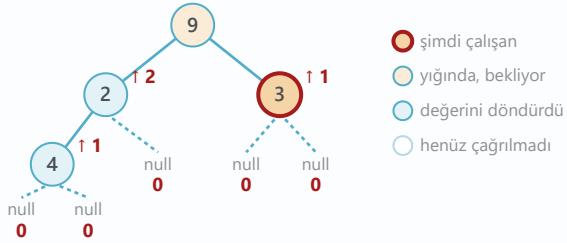
ÇAĞRI YIĞINI

- **`boyut(null)`**
- `boyut(3) · s.3`
- `boyut(9) · s.3`

DEĞİŞKENLER

`k = null` `dönüş = 0`

Kod İzleme: `boyut()` — değerler yukarı taşınır



`boyut(3)` = 1 + 0 + 0 = **1** döner. Değer 9'un çerçevesine taşınır.

Adım 16 / 17

```
1 int boyut(AgacDugumu k) {            // düğüm sayısı
2     if (k == null) return 0;
▶     return 1 + boyut(k.sol) + boyut(k.sag);
4 }
```

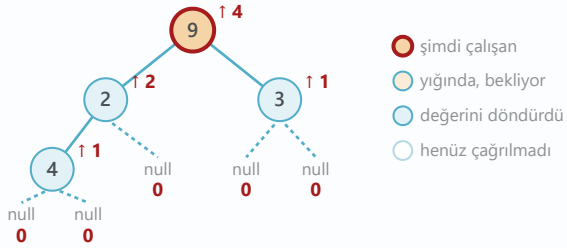
ÇAĞRI YIĞINI

- **boyut(3)**
- boyut(9) · s.3

DEĞİŞKENLER

`k = 3` `sol = 0` `sag = 0` `dönüş = 1`

Kod İzleme: `boyut()` — değerler yukarı taşınır



`boyut(9)` = 1 + 2 + 1 = **4** döner. **Sonuç: 4**

Adım 17 / 17

```
1 int boyut(AgacDugumu k) {            // düğüm sayısı
2     if (k == null) return 0;
▶     return 1 + boyut(k.sol) + boyut(k.sag);
4 }
```

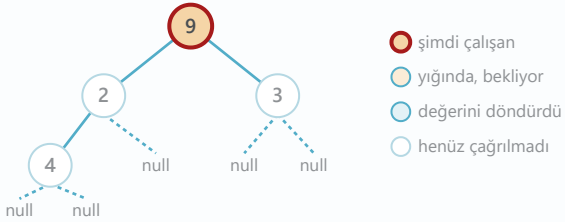
ÇAĞRI YIĞINI

`boyut(9)`

DEĞİŞKENLER

`k = 9` `sol = 2` `sag = 1` `dönüş = 4`

Kod İzleme: `yukseklik()`



`yukseklik(9)` (kök): null değil → önce **sol alt ağaç** için `yukseklik(2)` çağrılır.

Adım 1 / 17

```
1 int yukseklik(AgacDugumu k) {        // kenar sayısı
2     if (k == null) return -1;        // boş ağaç: -1
▶     return 1 + Math.max(yukseklik(k.sol),
4                         yukseklik(k.sag));
5 }
```

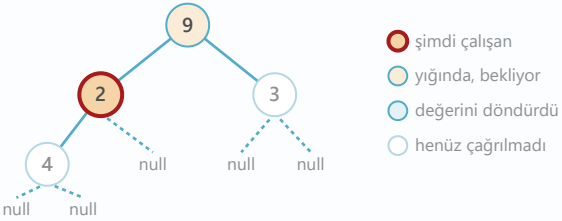
ÇAĞRI YIĞINI

`yukseklik(9)`

DEĞİŞKENLER

`k = 9` `sol = ?` `sag = ?`

Kod İzleme: `yukseklik()`



`yukseklik(2)` (9'un sol çocuğu): null değil → önce **sol alt ağaç** için `yukseklik(4)` çağrılır.

Adım 2 / 17

```
1 int yukseklik(AgacDugumu k) {        // kenar sayısı
2     if (k == null) return -1;        // boş ağaç: -1
▶     return 1 + Math.max(yukseklik(k.sol),
4                         yukseklik(k.sag));
5 }
```

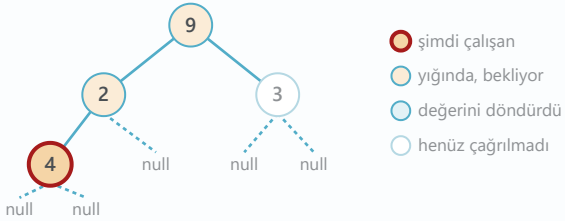
ÇAĞRI YIĞINI

- **`yukseklik(2)`**
- `yukseklik(9) · s.3`

DEĞİŞKENLER

`k = 2` `sol = ?` `sag = ?`

Kod İzleme: `yukseklik()`



`yukseklik(4)` (2'nin sol çocuğu): null değil → önce **sol alt ağaç** için `yukseklik(null)` çağrılır.

Adım 3 / 17

```
1 int yukseklik(AgacDugumu k) {        // kenar sayısı
2     if (k == null) return -1;        // boş ağaç: -1
▶     return 1 + Math.max(yukseklik(k.sol),
4                         yukseklik(k.sag));
5 }
```

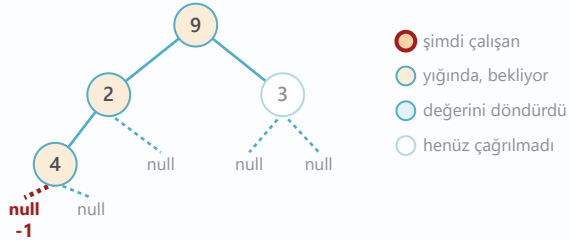
ÇAĞRI YIĞINI

- **`yukseklik(4)`**
- `yukseklik(2) · s.3`
- `yukseklik(9) · s.3`

DEĞİŞKENLER

`k = 4` `sol = ?` `sag = ?`

Kod İzleme: `yukseklik()`



`yukseklik(4.sol)` = `yukseklik(null)`: boş ağaç → **return -1**. Boş ağacın yüksekliği −1 kabul edilir; böylece yaprak 1 + max(−1, −1) = 0 olur.

Adım 4 / 17

```
1 int yukseklik(AgacDugumu k) {        // kenar sayısı
▶     if (k == null) return -1;        // boş ağaç: -1
3     return 1 + Math.max(yukseklik(k.sol),
4                         yukseklik(k.sag));
5 }
```

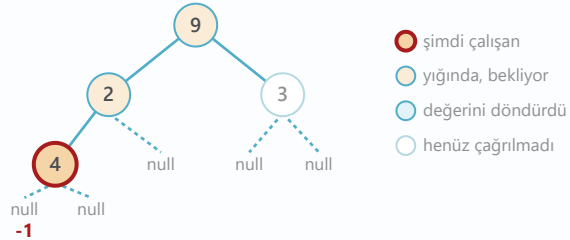
ÇAĞRI YIĞINI

- `yukseklik(null)`
- `yukseklik(4) · s.3`
- `yukseklik(2) · s.3`
- `yukseklik(9) · s.3`

DEĞİŞKENLER

`k = null` `dönüş = -1`

Kod İzleme: `yukseklik()`



Sol sonuç **-1** geldi. Şimdi sağ alt ağaç: `yukseklik(null)`.

Adım 5 / 17

```
1 int yukseklik(AgacDugumu k) {          // kenar sayısı
2     if (k == null) return -1;          // boş ağaç: -1
3     return 1 + Math.max(yukseklik(k.sol),
                          yukseklik(k.sag));
5 }
```

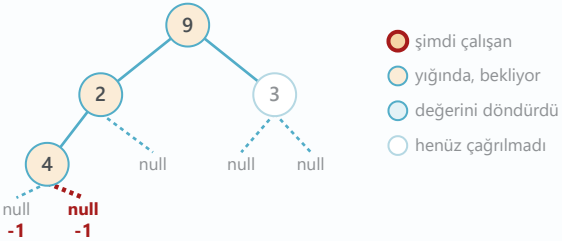
ÇAĞRI YIĞINI

- **`yukseklik(4)`**
- `yukseklik(2) · s.3`
- `yukseklik(9) · s.3`

DEĞİŞKENLER

`k = 4` `sol = -1` `sag = ?`

Kod İzleme: `yukseklik()`



`yukseklik(4.sag)` = `yukseklik(null)` : boş ağaç → **return -1**.

Adım 6 / 17

```
1 int yukseklik(AgacDugumu k) {          // kenar sayısı
▶     if (k == null) return -1;          // boş ağaç: -1
3     return 1 + Math.max(yukseklik(k.sol),
4                         yukseklik(k.sag));
5 }
```

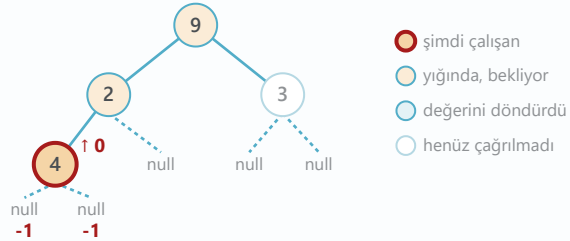
ÇAĞRI YIĞINI

- `yukseklik(null)`
- `yukseklik(4) · s.4`
- `yukseklik(2) · s.3`
- `yukseklik(9) · s.3`

DEĞİŞKENLER

`k = null` `dönüş = -1`

Kod İzleme: `yukseklik()`



`yukseklik(4)` = 1 + max(-1, -1) = **0** döner. Değer 2'nin çerçevesine taşınır.

Adım 7 / 17

```
1 int yukseklik(AgacDugumu k) {          // kenar sayısı
2     if (k == null) return -1;          // boş ağaç: -1
3     return 1 + Math.max(yukseklik(k.sol),
                          yukseklik(k.sag));
5 }
```

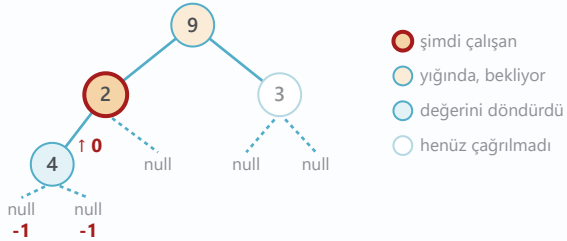
ÇAĞRI YIĞINI

- **`yukseklik(4)`**
- `yukseklik(2) · s.3`
- `yukseklik(9) · s.3`

DEĞİŞKENLER

`k = 4` `sol = -1` `sag = -1` `dönüş = 0`

Kod İzleme: `yukseklik()`



Sol sonuç **0** geldi. Şimdi sağ alt ağaç: `yukseklik(null)`.

Adım 8 / 17

```
1 int yukseklik(AgacDugumu k) {        // kenar sayısı
2     if (k == null) return -1;        // boş ağaç: -1
3     return 1 + Math.max(yukseklik(k.sol),
▶                         yukseklik(k.sag));
5 }
```

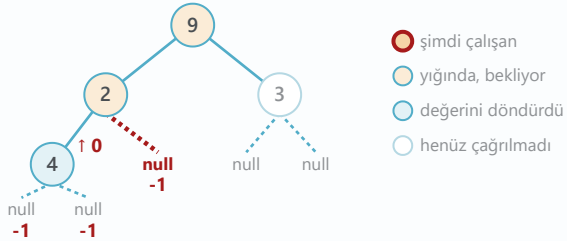
ÇAĞRI YIĞINI

- **`yukseklik(2)`**
- `yukseklik(9) · s.3`

DEĞİŞKENLER

`k = 2` `sol = 0` `sag = ?`

Kod İzleme: `yukseklik()`



`yukseklik(2.sag)` = `yukseklik(null)` : boş ağaç → **return -1**.

Adım 9 / 17

```
1 int yukseklik(AgacDugumu k) {          // kenar sayısı
▶     if (k == null) return -1;          // boş ağaç: -1
3     return 1 + Math.max(yukseklik(k.sol),
4                         yukseklik(k.sag));
5 }
```

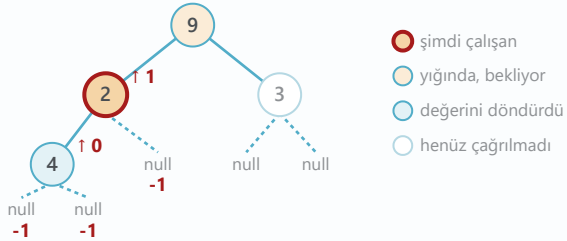
ÇAĞRI YIĞINI

- **`yukseklik(null)`**
- `yukseklik(2) · s.4`
- `yukseklik(9) · s.3`

DEĞİŞKENLER

`k = null` `dönüş = -1`

Kod İzleme: `yukseklik()`



`yukseklik(2)` = 1 + max(0, -1) = **1** döner. Değer 9'un çerçevesine taşınır.

Adım 10 / 17

```
1 int yukseklik(AgacDugumu k) {        // kenar sayısı
2     if (k == null) return -1;        // boş ağaç: -1
3     return 1 + Math.max(yukseklik(k.sol),
                          yukseklik(k.sag));
5 }
```

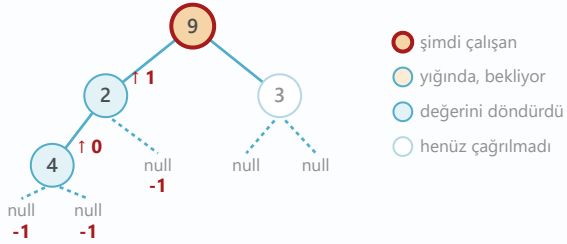
ÇAĞRI YIĞINI

- **`yukseklik(2)`**
- `yukseklik(9) · s.3`

DEĞİŞKENLER

`k = 2` `sol = 0` `sag = -1` `dönüş = 1`

Kod İzleme: `yukseklik()`



Sol sonuç **1** geldi. Şimdi sağ alt ağaç: `yukseklik(3)`.

Adım 11 / 17

```
1 int yukseklik(AgacDugumu k) {        // kenar sayısı
2     if (k == null) return -1;        // boş ağaç: -1
3     return 1 + Math.max(yukseklik(k.sol),
▶                         yukseklik(k.sag));
5 }
```

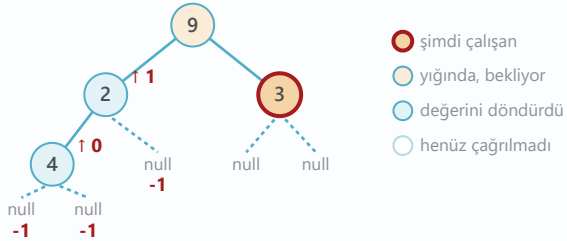
ÇAĞRI YIĞINI

`yukseklik(9)`

DEĞİŞKENLER

`k = 9` `sol = 1` `sag = ?`

Kod İzleme: `yukseklik()`



`yukseklik(3)` (9'un sağ çocuğu): null değil → önce **sol alt ağaç** için `yukseklik(null)` çağrılır.

Adım 12 / 17

```
1 int yukseklik(AgacDugumu k) {        // kenar sayısı
2     if (k == null) return -1;        // boş ağaç: -1
▶     return 1 + Math.max(yukseklik(k.sol),
4                         yukseklik(k.sag));
5 }
```

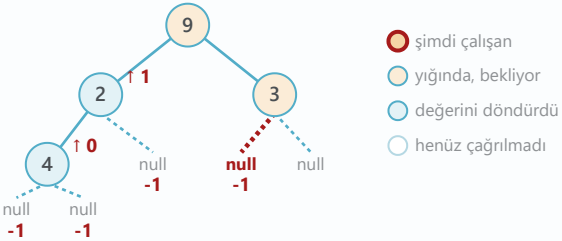
ÇAĞRI YIĞINI

- **`yukseklik(3)`**
- `yukseklik(9) · s.4`

DEĞİŞKENLER

`k = 3` `sol = ?` `sag = ?`

Kod İzleme: `yukseklik()`



`yukseklik(3.sol)` = `yukseklik(null)` : boş ağaç → **return -1**.

Adım 13 / 17

```
1 int yukseklik(AgacDugumu k) {          // kenar sayısı
▶     if (k == null) return -1;          // boş ağaç: -1
3     return 1 + Math.max(yukseklik(k.sol),
4                         yukseklik(k.sag));
5 }
```

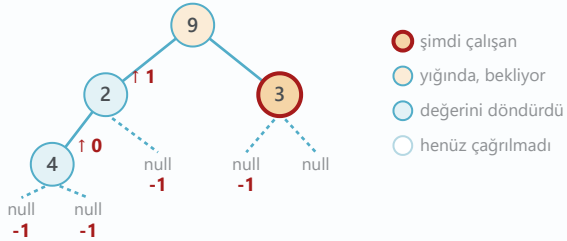
ÇAĞRI YIĞINI

- `yukseklik(null)`
- `yukseklik(3)` · s.3
- `yukseklik(9)` · s.4

DEĞİŞKENLER

`k = null` `dönüş = -1`

Kod İzleme: `yukseklik()`



Sol sonuç **-1** geldi. Şimdi sağ alt ağaç: `yukseklik(null)`.

Adım 14 / 17

```
1 int yukseklik(AgacDugumu k) {          // kenar sayısı
2     if (k == null) return -1;          // boş ağaç: -1
3     return 1 + Math.max(yukseklik(k.sol),
                          yukseklik(k.sag));
5 }
```

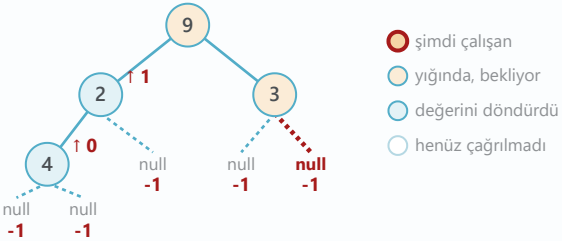
ÇAĞRI YIĞINI

- **yukseklik(3)**
- yukseklik(9) · s.4

DEĞİŞKENLER

`k = 3` `sol = −1` `sag = ?`

Kod İzleme: `yukseklik()`



`yukseklik(3.sag)` = `yukseklik(null)` : boş ağaç → **return -1**.

Adım 15 / 17

```
1 int yukseklik(AgacDugumu k) {          // kenar sayısı
▶     if (k == null) return -1;          // boş ağaç: -1
3     return 1 + Math.max(yukseklik(k.sol),
4                         yukseklik(k.sag));
5 }
```

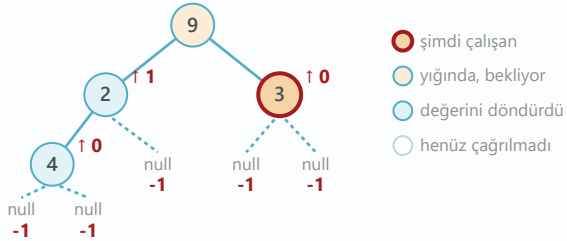
ÇAĞRI YIĞINI

- **`yukseklik(null)`**
- `yukseklik(3) · s.4`
- `yukseklik(9) · s.4`

DEĞİŞKENLER

`k = null` `dönüş = -1`

Kod İzleme: `yukseklik()`



`yukseklik(3)` = 1 + max(-1, -1) = **0** döner. Değer 9'un çerçevesine taşınır.

Adım 16 / 17

```
1 int yukseklik(AgacDugumu k) {          // kenar sayısı
2     if (k == null) return -1;          // boş ağaç: -1
3     return 1 + Math.max(yukseklik(k.sol),
                          yukseklik(k.sag));
5 }
```

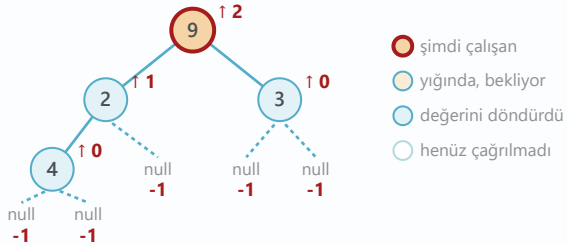
ÇAĞRI YIĞINI

- **`yukseklik(3)`**
- `yukseklik(9) · s.4`

DEĞİŞKENLER

`k = 3` `sol = -1` `sag = -1` `dönüş = 0`

Kod İzleme: `yukseklik()`



`yukseklik(9)` = 1 + max(1, 0) = **2** döner. **Sonuç: 2**

Adım 17 / 17

```
1 int yukseklik(AgacDugumu k) {        // kenar sayısı
2     if (k == null) return -1;        // boş ağaç: -1
3     return 1 + Math.max(yukseklik(k.sol),
                          yukseklik(k.sag));
5 }
```

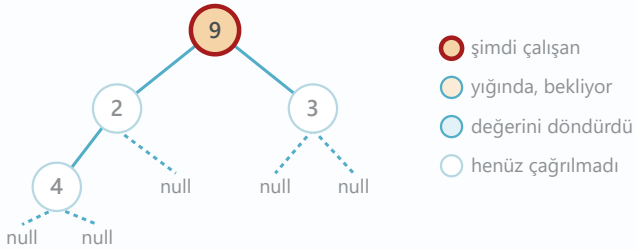
ÇAĞRI YIĞINI

`yukseklik(9)`

DEĞİŞKENLER

`k = 9` `sol = 1` `sag = 0` `dönüş = 2`

Kod İzleme: `yaprakSayisi()`



`yaprakSayisi(9)` (kök): yaprak değil → sol ve sağ alt ağaçların yaprakları toplanacak; önce sol.

Adım 1 / 7

```
1 int yaprakSayisi(AgacDugumu k) {
2     if (k == null) return 0;
3     if (k.sol == null && k.sag == null) return 1;
▶     return yaprakSayisi(k.sol) + yaprakSayisi(k.sag);
5 }
```

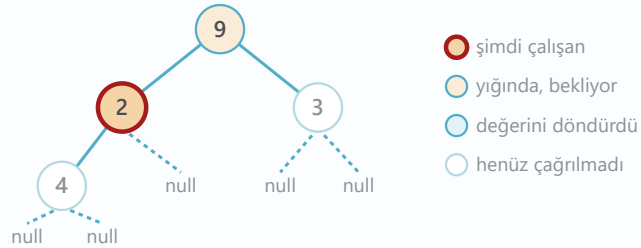
ÇAĞRI YIĞINI

`yaprakSayisi(9)`

DEĞİŞKENLER

`k = 9` `sol = ?` `sag = ?`

Kod İzleme: `yaprakSayisi()`



`yaprakSayisi(2)` (9'un sol çocuğu): yaprak değil → sol ve sağ alt ağaçların yaprakları toplanacak; önce sol.

Adım 2 / 7

```
1 int yaprakSayisi(AgacDugumu k) {
2     if (k == null) return 0;
3     if (k.sol == null && k.sag == null) return 1;
▶     return yaprakSayisi(k.sol) + yaprakSayisi(k.sag);
5 }
```

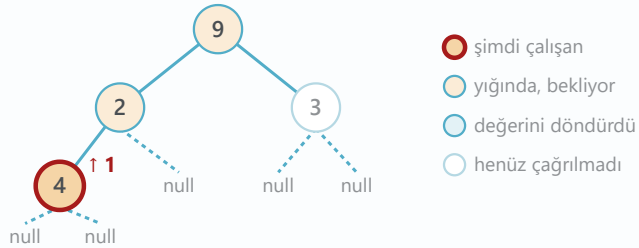
ÇAĞRI YIĞINI

- `yaprakSayisi(2)`
- `yaprakSayisi(9) · s.4`

DEĞİŞKENLER

`k = 2` `sol = ?` `sag = ?`

Kod İzleme: `yaprakSayisi()`



`yaprakSayisi(4)` (2'nin sol çocuğu): iki çocuğu da null → **yaprak**, return 1. Alt çağrı yapılmaz.

Adım 3 / 7

```
1 int yaprakSayisi(AgacDugumu k) {
2     if (k == null) return 0;
▶     if (k.sol == null && k.sag == null) return 1;
4     return yaprakSayisi(k.sol) + yaprakSayisi(k.sag);
5 }
```

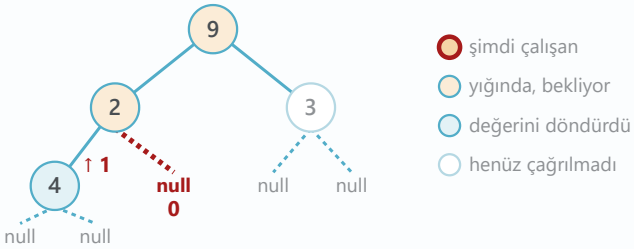
ÇAĞRI YIĞINI

- `yaprakSayisi(4)`
- `yaprakSayisi(2) · s.4`
- `yaprakSayisi(9) · s.4`

DEĞİŞKENLER

`k = 4` `dönüş = 1`

Kod İzleme: `yaprakSayisi()`



`yaprakSayisi(2.sag)` = `yaprakSayisi(null)` : boş ağaç → **return 0**.

Adım 4 / 7

```
1 int yaprakSayisi(AgacDugumu k) {
▶     if (k == null) return 0;
3     if (k.sol == null && k.sag == null) return 1;
4     return yaprakSayisi(k.sol) + yaprakSayisi(k.sag);
5 }
```

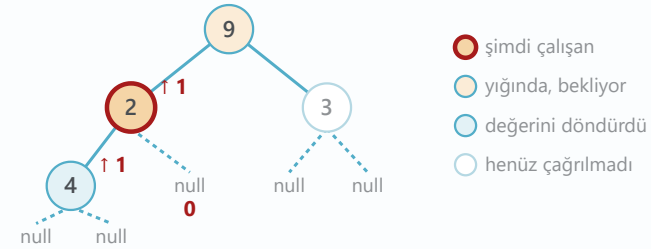
ÇAĞRI YIĞINI

- `yaprakSayisi(null)`
- `yaprakSayisi(2) · s.4`
- `yaprakSayisi(9) · s.4`

DEĞİŞKENLER

`k = null` `dönüş = 0`

Kod İzleme: `yaprakSayisi()`



`yaprakSayisi(2)` = 1 + 0 = **1** döner.

Adım 5 / 7

```
1 int yaprakSayisi(AgacDugumu k) {
2     if (k == null) return 0;
3     if (k.sol == null && k.sag == null) return 1;
▶     return yaprakSayisi(k.sol) + yaprakSayisi(k.sag);
5 }
```

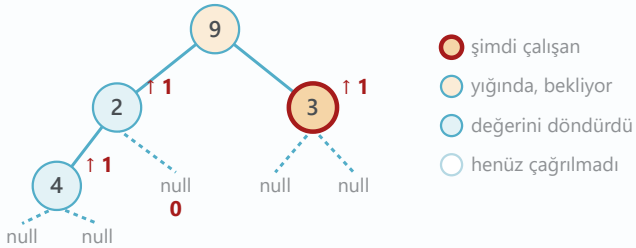
ÇAĞRI YIĞINI

- **`yaprakSayisi(2)`**
- `yaprakSayisi(9) · s.4`

DEĞİŞKENLER

`k = 2` `sol = 1` `sag = 0` `dönüş = 1`

Kod İzleme: `yaprakSayisi()`



`yaprakSayisi(3)` (9'un sağ çocuğu): iki çocuğu da null → **yaprak**, return 1. Alt çağrı yapılmaz.

Adım 6 / 7

```
1 int yaprakSayisi(AgacDugumu k) {
2     if (k == null) return 0;
▶     if (k.sol == null && k.sag == null) return 1;
4     return yaprakSayisi(k.sol) + yaprakSayisi(k.sag);
5 }
```

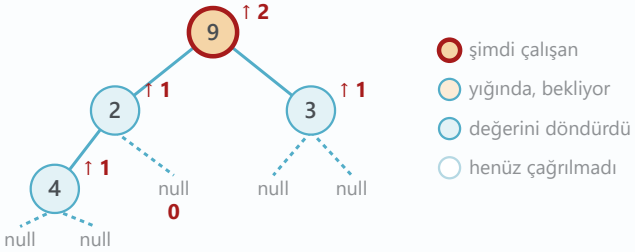
ÇAĞRI YIĞINI

- **`yaprakSayisi(3)`**
- `yaprakSayisi(9) · s.4`

DEĞİŞKENLER

`k = 3` `dönüş = 1`

Kod İzleme: `yaprakSayisi()`



`yaprakSayisi(9)` = 1 + 1 = **2** döner. **Sonuç: 2**

Adım 7 / 7

```
1 int yaprakSayisi(AgacDugumu k) {
2     if (k == null) return 0;
3     if (k.sol == null && k.sag == null) return 1;
▶     return yaprakSayisi(k.sol) + yaprakSayisi(k.sag);
5 }
```

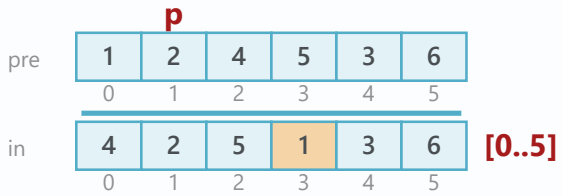
ÇAĞRI YIĞINI

`yaprakSayisi(9)`

DEĞİŞKENLER

`k = 9` `sol = 1` `sag = 1` `dönüş = 2`

Kod İzleme: kök önde + kök ortada'dan ağacı kurma



1

pre[0] = **1** → yeni düğüm (kök). in'de **m = 3**: [0..2] sol alt ağaç, [4..5] sağ alt ağaç olur.

Adım 1 / 10

```
1 int p = 0; // pre[] içinde sıradaki kök
2 AgacDugumu kur(int[] pre, int[] in, int bas, int son) {
▶     if (bas > son) return null;       // boş aralık
▶     AgacDugumu k = new AgacDugumu(pre[p++]);
▶     int m = bas; while (in[m] != k.veri) m++;
6     k.sol = kur(pre, in, bas, m - 1); // soldakiler
7     k.sag = kur(pre, in, m + 1, son); // sağdakiler
8     return k;
9 }
```

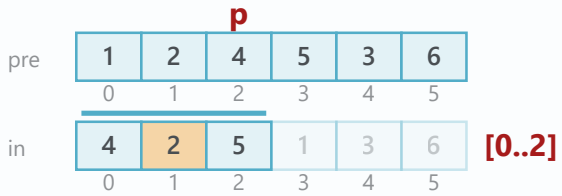
ÇAĞRI YIĞINI

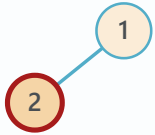
- `kur(0,5)`
- `main()`

DEĞİŞKENLER

`bas = 0` `son = 5` `k = düğüm(1)` `m = 3`

Kod İzleme: kök önde + kök ortada'dan ağacı kurma





pre[1] = **2** → yeni düğüm (1'in sol çocuğu). in'de **m = 1**: [0..0] sol alt ağaç, [2..2] sağ alt ağaç olur.

Adım 2 / 10

```
1 int p = 0; // pre[] içinde sıradaki kök
2 AgacDugumu kur(int[] pre, int[] in, int bas, int son) {
▶     if (bas > son) return null;       // boş aralık
▶     AgacDugumu k = new AgacDugumu(pre[p++]);
▶     int m = bas; while (in[m] != k.veri) m++;
6     k.sol = kur(pre, in, bas, m - 1); // soldakiler
7     k.sag = kur(pre, in, m + 1, son); // sağdakiler
8     return k;
9 }
```

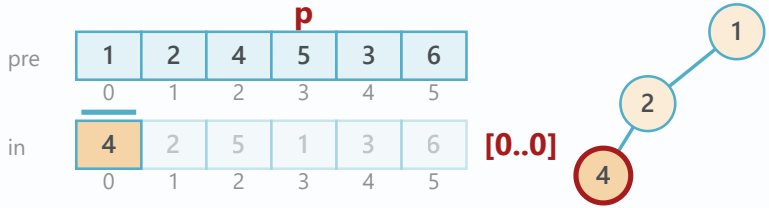
ÇAĞRI YIĞINI

- `kur(0,2)`
- `kur(0,5)`
- `main()`

DEĞİŞKENLER

`bas = 0` `son = 2` `k = düğüm(2)` `m = 1`

Kod İzleme: kök önde + kök ortada'dan ağacı kurma



pre[2] = **4** → yeni düğüm (2'nin sol çocuğu); m = 0. Sol ve sağ aralık **boş** → iki çağrı null döner: 4 **yaprak**.


```
1 int p = 0; // pre[] içinde sıradaki kök
2 AgacDugumu kur(int[] pre, int[] in, int bas, int son) {
▶     if (bas > son) return null;       // boş aralık
▶     AgacDugumu k = new AgacDugumu(pre[p++]);
▶     int m = bas; while (in[m] != k.veri) m++;
▶     k.sol = kur(pre, in, bas, m - 1); // soldakiler
▶     k.sag = kur(pre, in, m + 1, son); // sağdakiler
▶     return k;
9 }
```

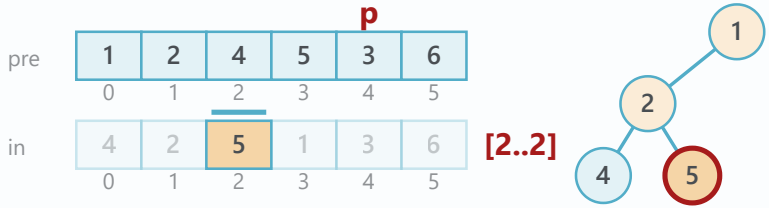
ÇAĞRI YIĞINI

- **kur(0,0)**
- kur(0,2)
- kur(0,5)
- main()

DEĞİŞKENLER

bas = 0 | son = 0 | k = **düğüm(4)** | m = 0

Kod İzleme: kök önde + kök ortada'dan ağacı kurma



pre[3] = **5** → yeni düğüm (2'nin sağ çocuğu); m = 2. Sol ve sağ aralık **boş** → iki çağrı null döner: 5 **yaprak**.

Adım 4 / 10

```
1 int p = 0; // pre[] içinde sıradaki kök
2 AgacDugumu kur(int[] pre, int[] in, int bas, int son) {
▶     if (bas > son) return null;       // boş aralık
▶     AgacDugumu k = new AgacDugumu(pre[p++]);
▶     int m = bas; while (in[m] != k.veri) m++;
▶     k.sol = kur(pre, in, bas, m - 1); // soldakiler
▶     k.sag = kur(pre, in, m + 1, son); // sağdakiler
▶     return k;
9 }
```

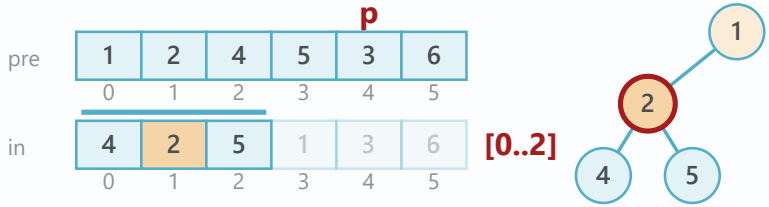
ÇAĞRI YIĞINI

- **kur(2,2)**
- kur(0,2)
- kur(0,5)
- main()

DEĞİŞKENLER

bas = **2** | son = **2** | k = **düğüm(5)** | m = **2**

Kod İzleme: kök önde + kök ortada'dan ağacı kurma



2'nin alt ağaçları kuruldu → **return düğüm(2)**; 1.sol = 2 yapılır.

Adım 5 / 10

```
1 int p = 0; // pre[] içinde sıradaki kök
2 AgacDugumu kur(int[] pre, int[] in, int bas, int son) {
3     if (bas > son) return null;       // boş aralık
4     AgacDugumu k = new AgacDugumu(pre[p++]);
5     int m = bas; while (in[m] != k.veri) m++;
6     k.sol = kur(pre, in, bas, m - 1); // soldakiler
7     k.sag = kur(pre, in, m + 1, son); // sağdakiler
▶     return k;
9 }
```

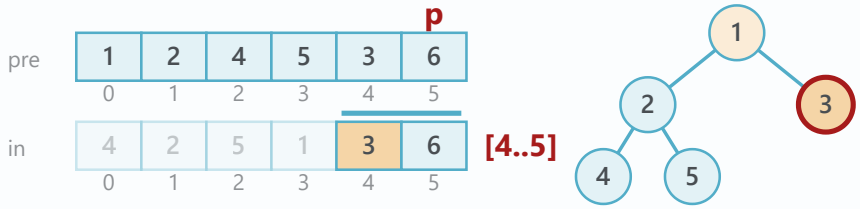
ÇAĞRI YIĞINI

- `kur(0,2)`
- `kur(0,5)`
- `main()`

DEĞİŞKENLER

`bas = 0` `son = 2` `k = düğüm(2)` `m = 1`

Kod İzleme: kök önde + kök ortada'dan ağacı kurma



pre[4] = **3** → yeni düğüm (1'in sağ çocuğu). in'de **m = 4**: [4..3] sol alt ağaç, [5..5] sağ alt ağaç olur.

Adım 6 / 10

```
1 int p = 0; // pre[] içinde sıradaki kök
2 AgacDugumu kur(int[] pre, int[] in, int bas, int son) {
▶     if (bas > son) return null;       // boş aralık
▶     AgacDugumu k = new AgacDugumu(pre[p++]);
▶     int m = bas; while (in[m] != k.veri) m++;
6     k.sol = kur(pre, in, bas, m - 1); // soldakiler
7     k.sag = kur(pre, in, m + 1, son); // sağdakiler
8     return k;
9 }
```

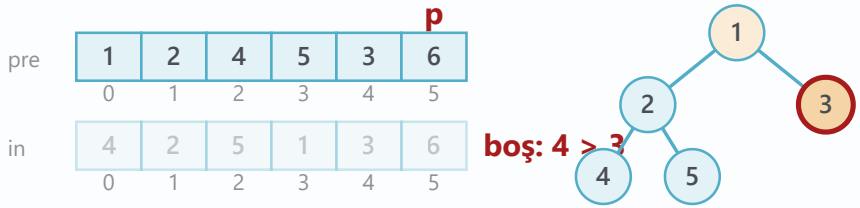
ÇAĞRI YIĞINI

- **kur(4,5)**
- kur(0,5)
- main()

DEĞİŞKENLER

bas = 4 | son = 5 | k = düğüm(3) | m = 4

Kod İzleme: kök önde + kök ortada'dan ağacı kurma



Sol aralık [4..3] boş → `kur` hemen **null** döner: 3'ün sol çocuğu yok.

Adım 7 / 10

```
1 int p = 0; // pre[] içinde sıradaki kök
2 AgacDugumu kur(int[] pre, int[] in, int bas, int son) {
▶     if (bas > son) return null;       // boş aralık
4     AgacDugumu k = new AgacDugumu(pre[p++]);
5     int m = bas; while (in[m] != k.veri) m++;
▶     k.sol = kur(pre, in, bas, m - 1); // soldakiler
7     k.sag = kur(pre, in, m + 1, son); // sağdakiler
8     return k;
9 }
```

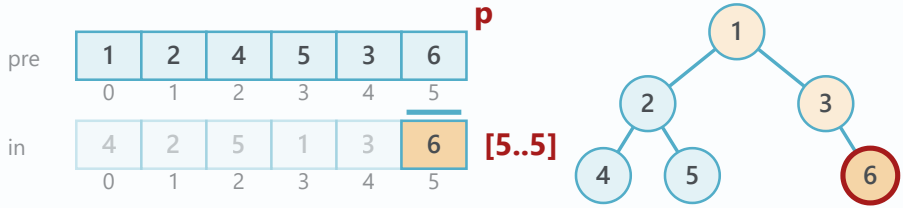
ÇAĞRI YIĞINI

- **kur(4,3)**
- kur(4,5)
- kur(0,5)
- main()

DEĞİŞKENLER

`bas = 4` `son = 5` `k = düğüm(3)` `m = 4`

Kod İzleme: kök önde + kök ortada'dan ağacı kurma



pre[5] = **6** → yeni düğüm (3'ün sağ çocuğu); m = 5. Sol ve sağ aralık **boş** → iki çağrı null döner: 6 **yaprak**.

Adım 8 / 10

```
1 int p = 0; // pre[] içinde sıradaki kök
2 AgacDugumu kur(int[] pre, int[] in, int bas, int son) {
▶     if (bas > son) return null;       // boş aralık
▶     AgacDugumu k = new AgacDugumu(pre[p++]);
▶     int m = bas; while (in[m] != k.veri) m++;
▶     k.sol = kur(pre, in, bas, m - 1); // soldakiler
▶     k.sag = kur(pre, in, m + 1, son); // sağdakiler
▶     return k;
9 }
```

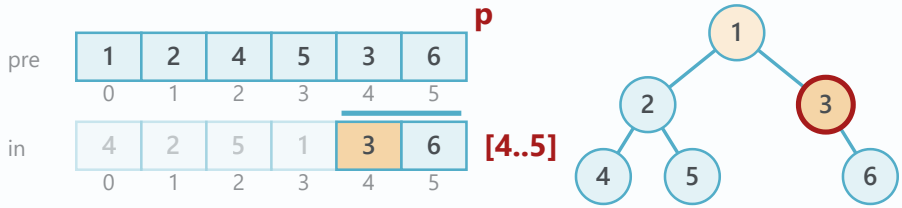
ÇAĞRI YIĞINI

- **kur(5,5)**
- kur(4,5)
- kur(0,5)
- main()

DEĞİŞKENLER

bas = **5** | son = **5** | k = **düğüm(6)** | m = **5**

Kod İzleme: kök önde + kök ortada'dan ağacı kurma



3'ün alt ağaçları kuruldu → **return düğüm(3)**; 1.sag = 3 yapılır.


```
1 int p = 0; // pre[] içinde sıradaki kök
2 AgacDugumu kur(int[] pre, int[] in, int bas, int son) {
3     if (bas > son) return null;       // boş aralık
4     AgacDugumu k = new AgacDugumu(pre[p++]);
5     int m = bas; while (in[m] != k.veri) m++;
6     k.sol = kur(pre, in, bas, m - 1); // soldakiler
7     k.sag = kur(pre, in, m + 1, son); // sağdakiler
▶     return k;
9 }
```

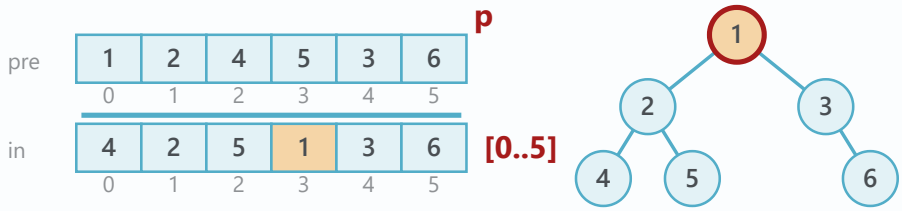
ÇAĞRI YIĞINI

- `kur(4,5)`
- `kur(0,5)`
- `main()`

DEĞİŞKENLER

`bas = 4` `son = 5` `k = düğüm(3)` `m = 4`

Kod İzleme: kök önde + kök ortada'dan ağacı kurma



1'in alt ağaçları kuruldu → **return düğüm(1)**; kök olarak `kok` 'a atanır. Sonuç: 1(2(4, 5), 3(—, 6)). m araması yüzünden O(n^2); in[] indisleri bir eşlemede (map) tutulursa O(n).

Adım 10 / 10

```
1 int p = 0; // pre[] içinde sıradaki kök
2 AgacDugumu kur(int[] pre, int[] in, int bas, int son) {
3     if (bas > son) return null;       // boş aralık
4     AgacDugumu k = new AgacDugumu(pre[p++]);
5     int m = bas; while (in[m] != k.veri) m++;
6     k.sol = kur(pre, in, bas, m - 1); // soldakiler
7     k.sag = kur(pre, in, m + 1, son); // sağdakiler
▶     return k;
9 }
```

ÇAĞRI YIĞINI

- `kur(0,5)`
- `main()`

DEĞİŞKENLER

`bas = 0` `son = 5` `k = düğüm(1)` `m = 3`

Gezinmelerin Maliyeti

Yöntem	Zaman	Ek bellek	Neden?
Özyinelemeli pre/in/post	O(n)	O(h)	Çağrı yığınında en fazla h + 2 çerçeve (null çağrısı dahil)
Yığın ile pre/in/post	O(n)	O(h)	Açık yığın, özyinelemenin yerini tutar
Kuyruk ile düzey sırası	O(n)	O(w)	w: en geniş düzeydeki düğüm sayısı

h ne kadar?

Dengeli ağaçta h ≈ $\log_2 n$; çarpık ağaçta h = n − 1. Çok derin ağaçlarda özyineleme `StackOverflowError` verebilir → yığınlı sürüm tercih edilir.

w ne kadar?

Mükemmel ağaçta son düzeyde ≈ n/2 düğüm vardır; düzey sırası geniş ağaçlarda daha çok bellek harcar.

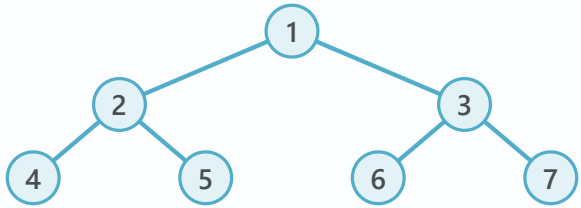
Laboratuvar: Yığın ve Kuyrukla Gezinme

Yığın ile kök önde: 1–7 tam ağaç

Girdi (düzey sırasıyla değerler, boş yer için -) `1, 2, 3, 4, 5, 6, 7`

KOD · **adım 1 / 39**

```
 1 public void kokBastaYigin(AgacDugumu kok) {
 2   if (kok == null) return;
 3   Deque<AgacDugumu> yigin = new ArrayDeque<…
 4   yigin.push(kok);
 5   while (!yigin.isEmpty()) {
 6     AgacDugumu gecici = yigin.pop();
 7     System.out.print(gecici.veri + " ");
 8     if (gecici.sag != null)
 9       yigin.push(gecici.sag);  // önce sağ
10     if (gecici.sol != null)
11       yigin.push(gecici.sol);  // sol üstte
12   }
13 }
```



yigin boş

`gecici = –` `yigin.size() = 0`

ÇIKTI –

kokBastaYigin, kök 1 ile çağrılır; kök null değil.

ÖZET

Özet

Konu	Hatırlanacaklar
Terimler	kök, ebeveyn/çocuk, yaprak, iç düğüm; n düğüm → n − 1 kenar
Ölçüler	derece = çocuk sayısı; derinlik = kökten kenar sayısı; yükseklik = en uzun aşağı yol
İkili ağaç	dolu: 0/2 çocuk · tam: son düzey soldan dolar · mükemmel: tüm düzeyler dolu
Gösterim	dizi: indis bağıntıları · bağlı: sol/sağ referansları
Gezinme	kök önde K-S-S · kök ortada S-K-S · kök sonda S-S-K · düzey sırası kuyrukla
Maliyet	tüm gezinmeler O(n); ek bellek O(h) (yığın) veya O(w) (kuyruk)

ÖZET

Etkileşimli Simülatörler

Genel Ağaç Simülatörü

Çocuk sayısı sınırsız ağaçta düğüm ekleme/silme ve gezinmeler.

İkili Arama Ağacı Gezinme Simülatörü

İkili ağaç üzerinde ekleme ve kök önde / ortada / sonda gezinmeleri izleme.

ÖZET

Sık Yapılan Hatalar

Taban durumu unutmak

`k.sol`'a erişmeden önce `k == null` denetlenmezse `NullPointerException`. Her özyinelemeli ağaç metodu boş ağaçla başlar.

Yükseklik tanımını karıştırmak

Kenar sayısıyla: yaprak 0, boş ağaç −1. Düğüm sayısıyla: yaprak 1, boş ağaç 0. Bir tanım seçilip tutarlı kullanılmalıdır.

Dolu ≠ tam

Dolu: her düğümde 0 ya da 2 çocuk. Tam: son düzey hariç dolu, son düzey soldan. Biri diğerini gerektirmez.

Tek gezinmeden ağaç kurmak

Tek bir sıra ağacı belirlemez. **Kök ortada** + (kök önde ya da kök sonda) gerekir; kök önde + kök sonda tek çocuklu düğümlerde belirsizdir.

İndis formülünü karıştırmak

1 tabanlı: $2i$, $2i + 1$, $\lfloor i/2 \rfloor$. 0 tabanlı: $2i + 1$, $2i + 2$, $\lfloor (i - 1)/2 \rfloor$.

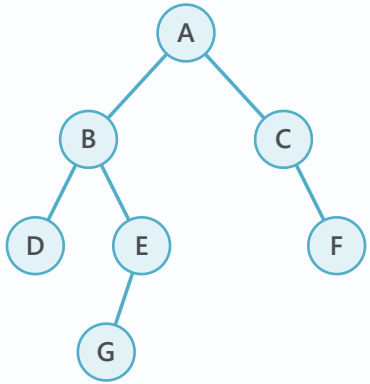
Derin ağaçta özyineleme

Çarpık ağaçta $h = n - 1$: yüz binlerce çerçeve `StackOverflowError` verir → yığınlı ya da kuyruklu sürüm.

ÖZET

Örnek Problemler

1. Sağdaki ağacın kök önde, kök ortada, kök sonda ve düzey sırası çıktılarını yazın.
2. Ağacın yüksekliği nedir? Yapraklar hangileri? E'nin derinliği kaçtır?
3. Bu ağaç dolu mu, tam mı, mükemmel mi?
4. 1 tabanlı dizi gösteriminde F ve G hangi indislerde durur? Dizi en az kaç hücre olmalı?
5. Yüksekliği 4 olan bir ikili ağaçta en fazla kaç düğüm olabilir?
6. Kök önde: A B D E C F, kök ortada: D B E A C F. Ağacı çizin.



▶ **Cevapları göster**

ÖZET

Alıştırmalar

1. `int icDugumSayisi(AgacDugumu k)`: iç düğümleri sayan metodu yazın.
2. `void ayna(AgacDugumu k)` : ağacın aynadaki görüntüsünü (her düğümde sol ↔ sağ) oluşturun.
3. `boolean esitMi(AgacDugumu a, AgacDugumu b)`: iki ağaç şekil ve değerce aynı mı?
4. `int seviyedeki(AgacDugumu k, int d)`: d derinliğindeki düğüm sayısını bulun.
5. Kök sonda: D E B F C A, kök ortada: D B E A F C. Ağacı çizin.
6. n düğümlü dolu ikili ağaçta kaç yaprak vardır? n çift olabilir mi?

▶ **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 9: Ağaçlar

Sonraki sunum: Bölüm 10: Arama Ağaçları · Çalışma soruları · Simülatörler: Genel Ağaç Simülatörü, İkili Arama Ağacı Gezinme Simülatörü · Terim sözlüğü