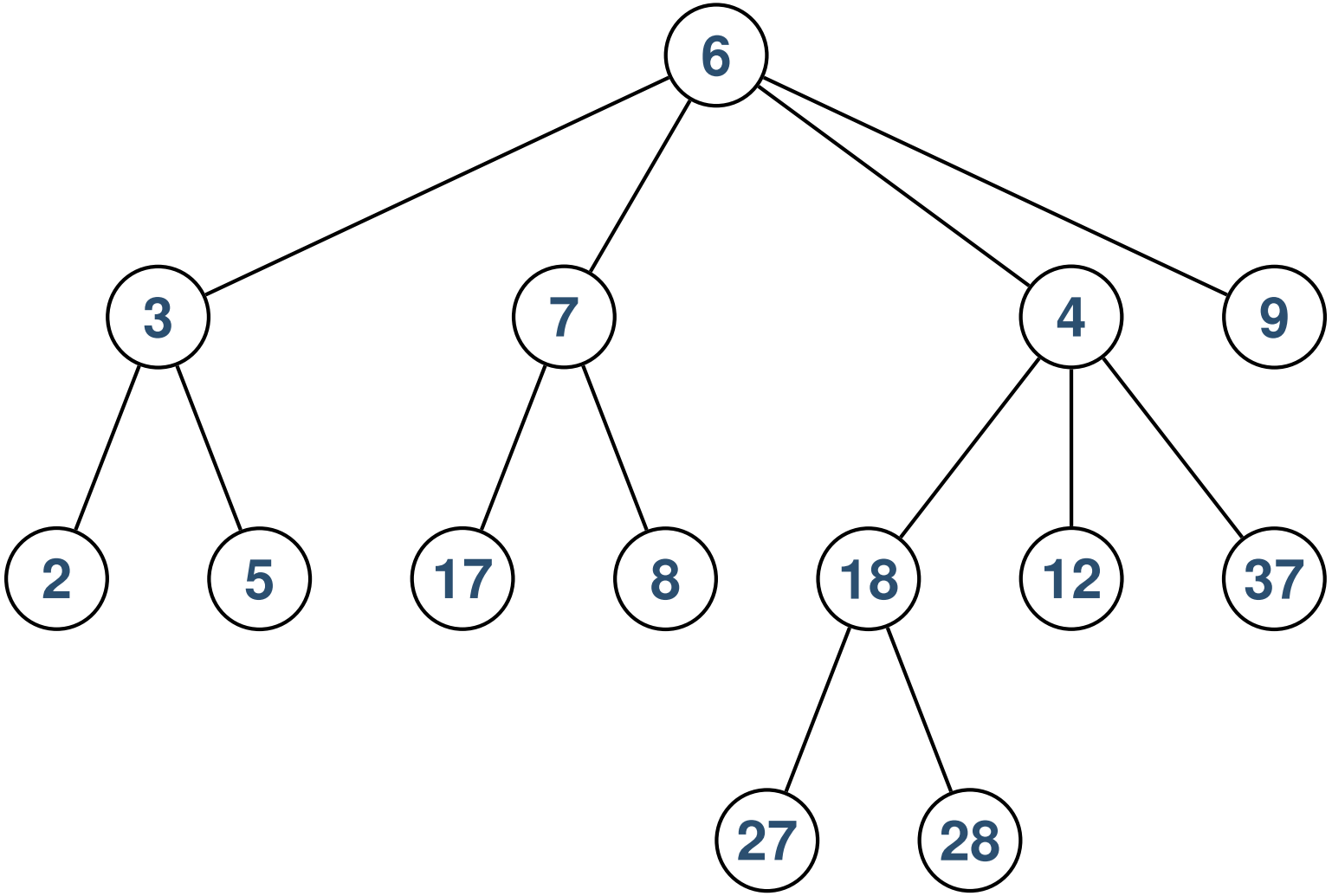
Bölüm 5. Ağaç

Olcay Taner Yıldız

2014

Giriş

14 düğümden oluşan bir ağaç yapısı

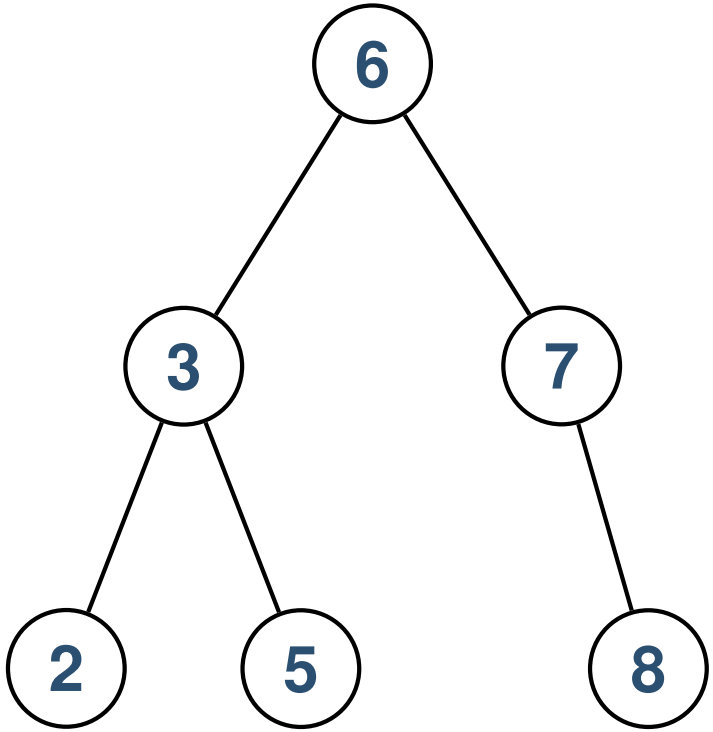


Tanımlar

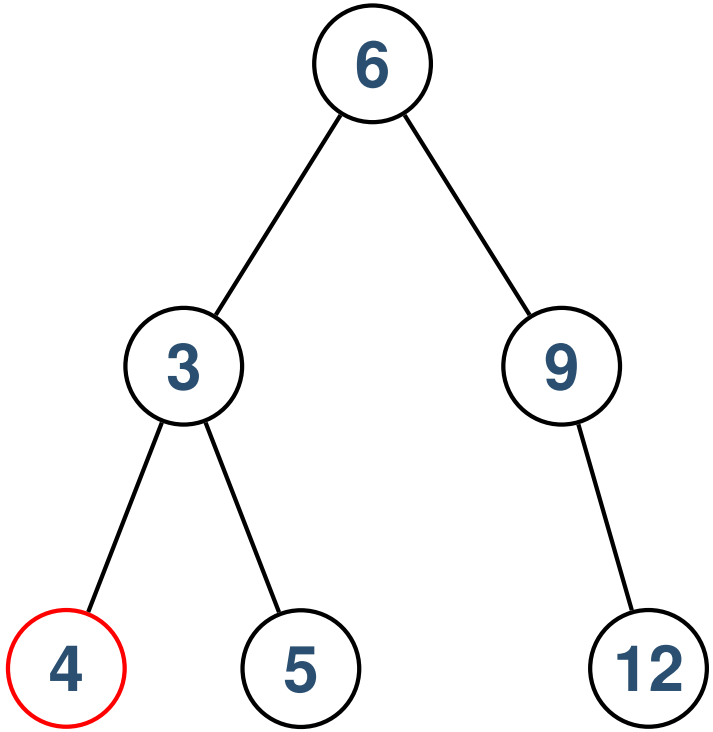
- Ağaçtaki x düğümünden y düğümüne bir ok varsa x düğümüne y düğümünün ebeveyni y düğümüne ise x'in çocuğu denir.
- Bir düğümün çocuk sayısı o düğümün derecesini belirtir.
- Ağaçtaki bir x düğümünden başlayıp çocukları üzerinden giderek bir y düğümüne ulaşılabiliyorsa, y düğümüne x düğümünün soyu, x düğümüne de y düğümünün atası denir.
- Ağaçtaki bir düğümün hiç çocuğu yoksa o düğüme yaprak düğüm denir.
- Bir ağacın derinliği o ağacın kökünden herhangi bir yaprağa ulaşmak için geçtiğimiz elemanların sayısının en fazlasıdır.

İkili Arama Ağacı Tanımı

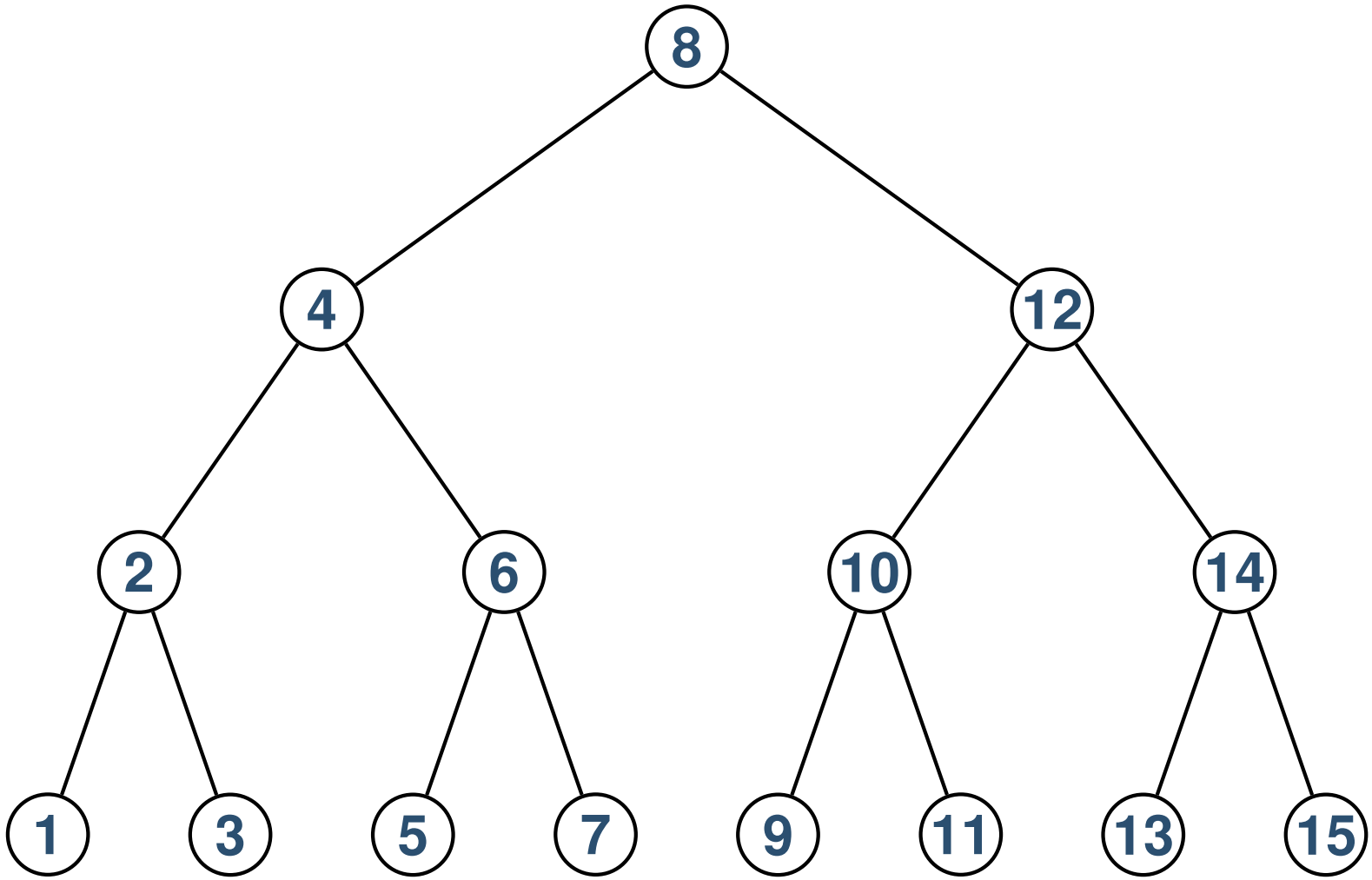
Altı elemandan oluşan bir ikili arama ağacı



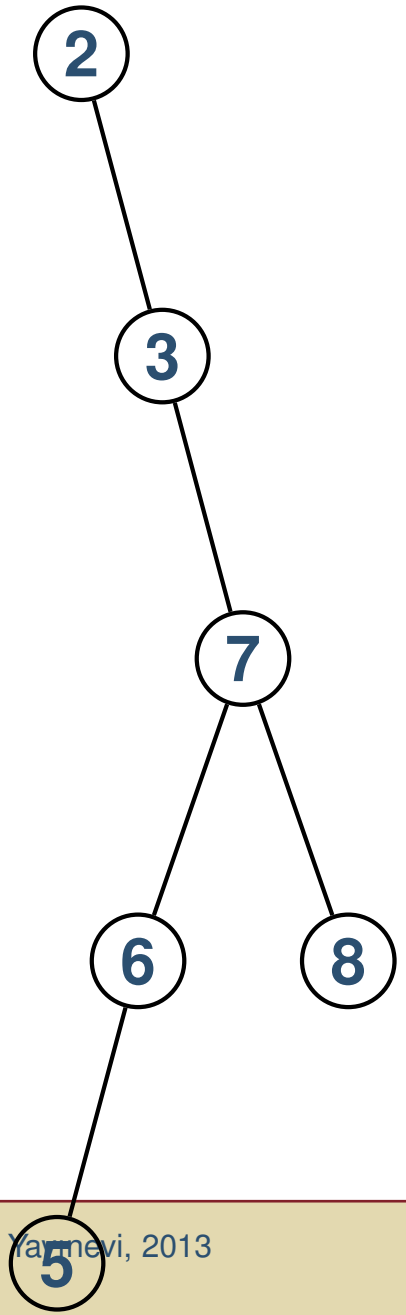
Altı düğümden oluşan bir ağaç.



15 elemandan oluşan 4 derinliğindeki dengeli bir ikili ağaç



Altı elemandan oluşan 4 derinliğindeki bir ikili ağaç yapısı



Tam sayılar içeren düğüm tanımı

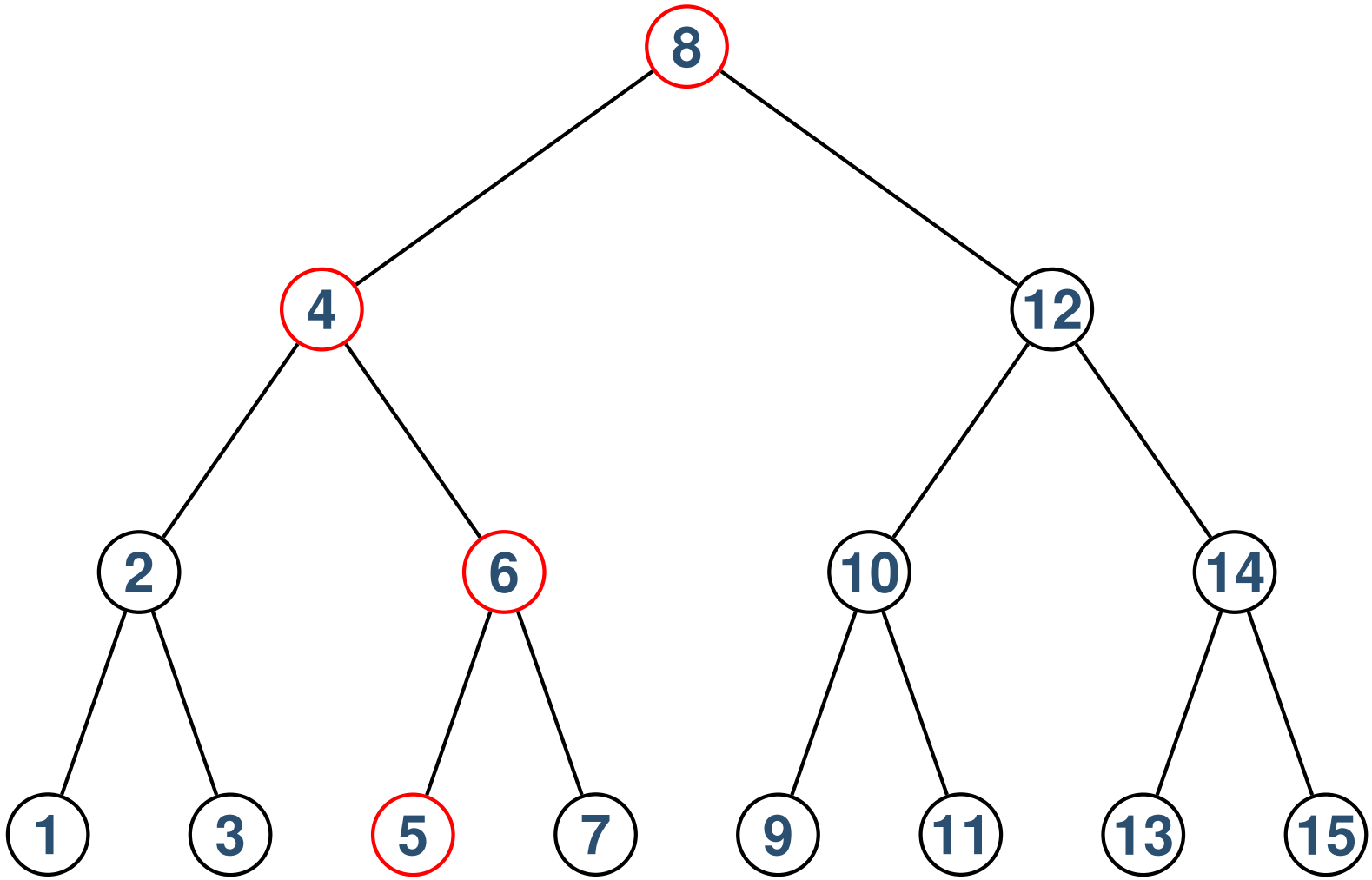

```
public class Dugum{
    int icerik;
    Dugum sol;
    Dugum sag;
    public Dugum(int icerik){
        this.icerik = icerik;
        sol = null;
        sag = null;
    }
}
```

Tam sayılar içeren ikili arama ağacı tanımı


```
public class Agac{
    Dugum kok;
    public Agac(){
        kok = null;
    }
}
```

Temel İkili Arama Ağacı İşlemleri

Örnek bir ikili arama ağacında 5'i arama.



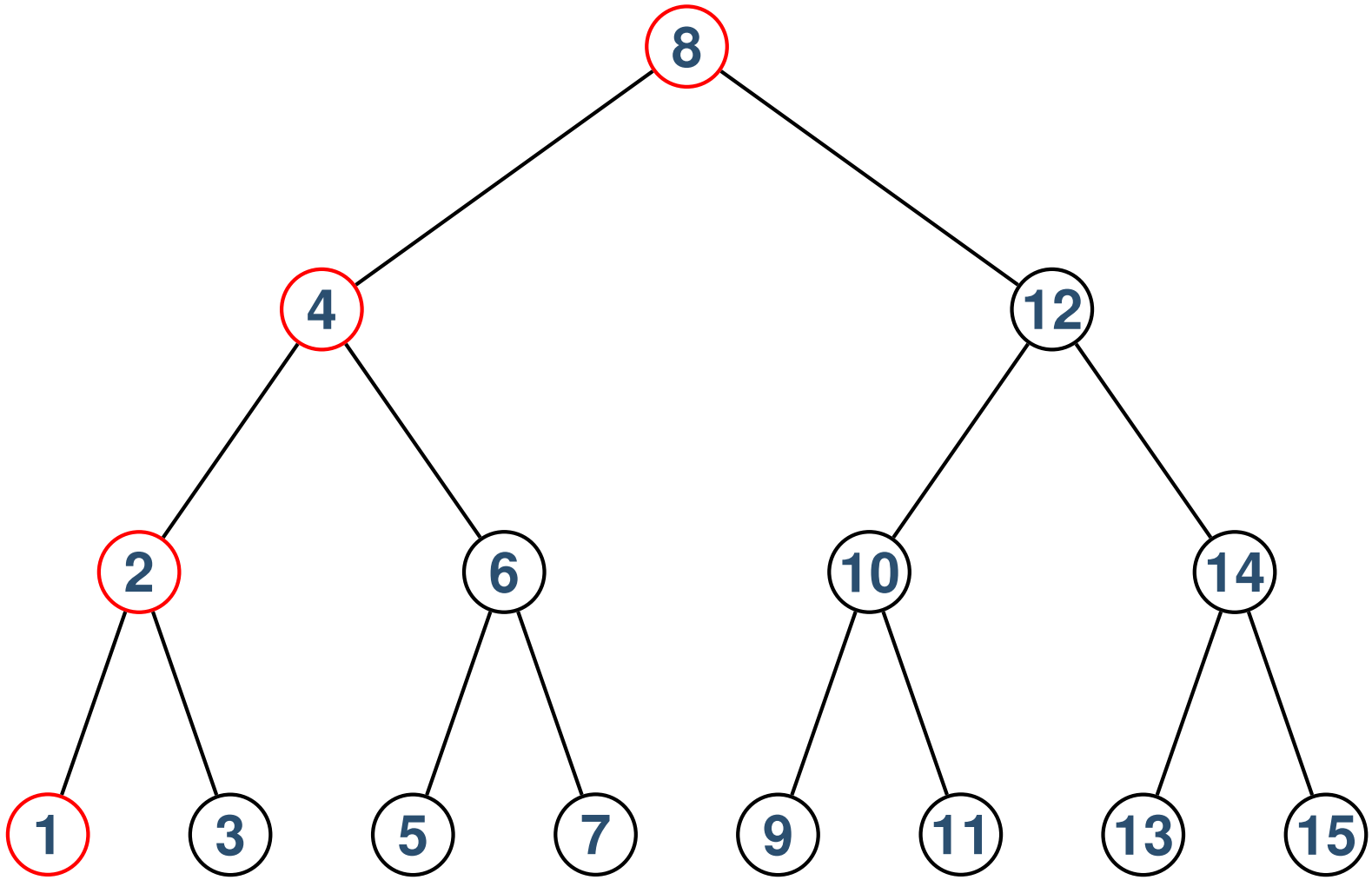
Verilen bir değeri ikili arama ağacında arayan özyinelemeli algoritma


```
1  Dugum agacAra(int eleman){
2     if ( icerik  == eleman)
3        return this;
4     else
5        if ( icerik  > eleman)
6           if (sol  != null)
7              return sol.agacAra(eleman);
8           else
9              return null;
10       else
11          if (sag != null)
12             return sag.agacAra(eleman);
13          else
14             return null;
15 }
```

Verilen bir değeri ikili arama ağacında arayan yinelemesiz algoritma


```
1  Dugum agacAra(int eleman){
2      Dugum d;
3      d = kok;
4      while (d != null){
5          if (d.icerik  == eleman)
6              return d;
7          else
8              if (d.icerik  > eleman)
9                  d = d.sol;
10             else
11                 d = d.sag;
12     }
13     return null;
14 }
```

İkili arama ağacındaki en küçük elemanı arama



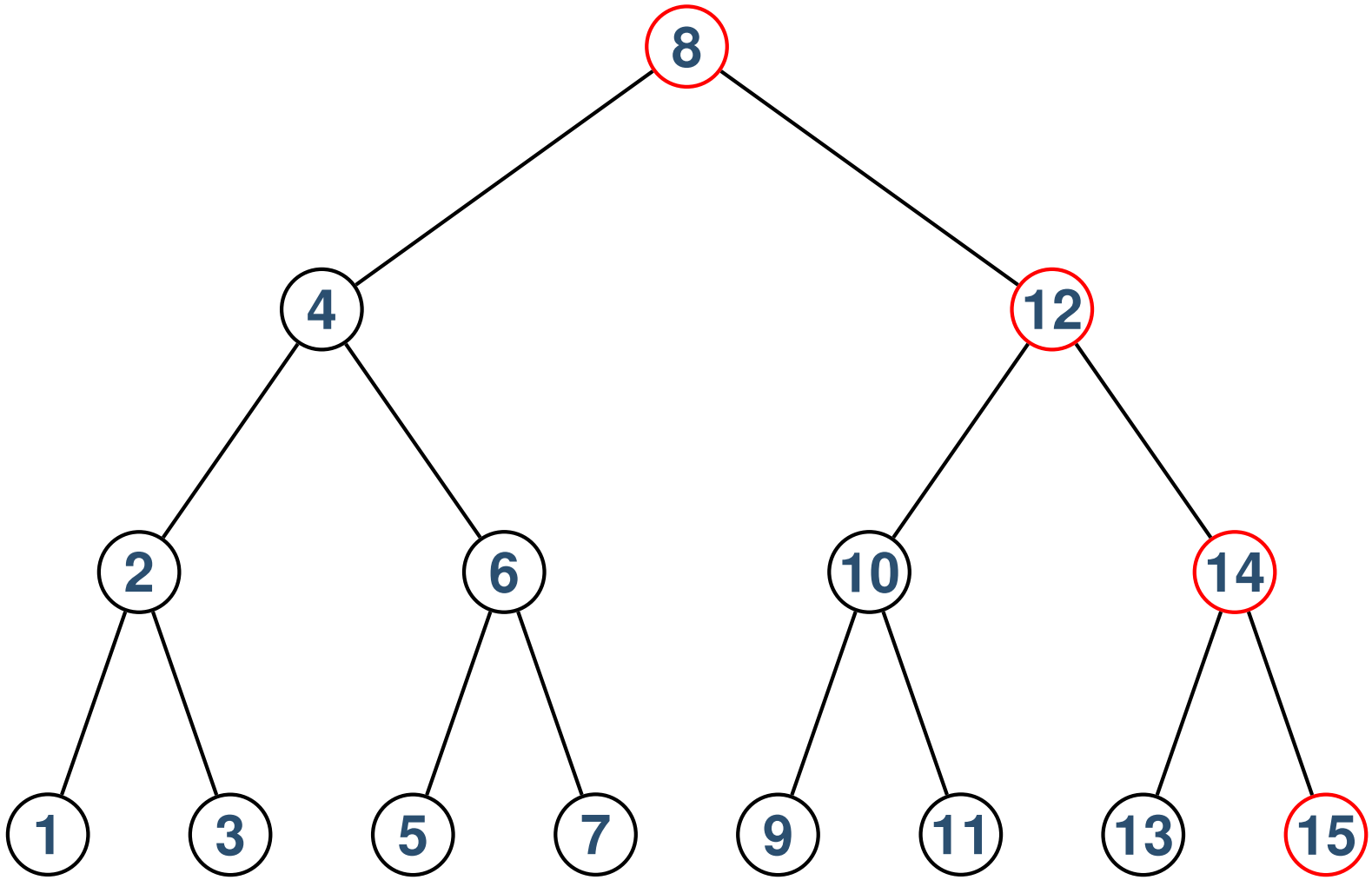
Arama ağacındaki en küçük elemanı bulan yinelemesiz algoritma

```
Dugum asgariAra(){
    Dugum sonuc = this;
    while (sonuc.sol != null)
        sonuc = sonuc.sol;
    return sonuc;
}
```

Arama ağacındaki en küçük elemanı bulan özyinelemeli algoritma


```
1 Dugum asgariAra(){
2     if (sol == null)
3         return this;
4     else
5         return sol.asgariAra();
6 }
```

İkili arama ağacındaki en büyük elemanı arama



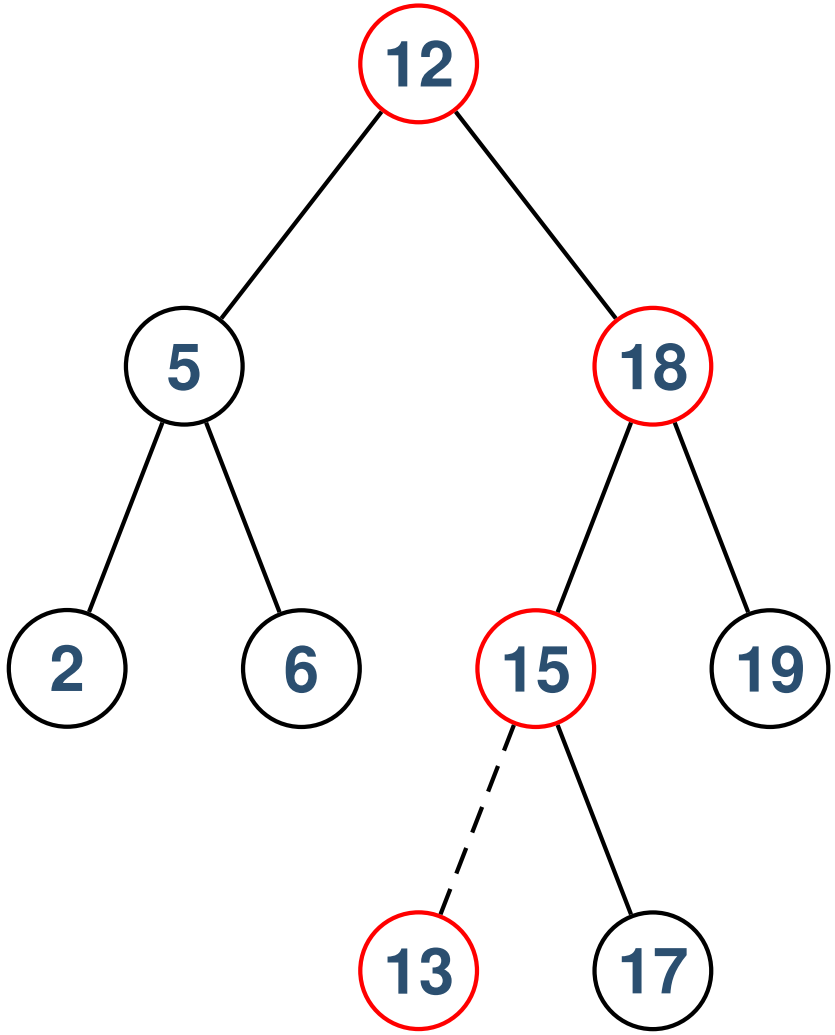
Arama ağacındaki en büyük elemanı bulan yinelemesiz algoritma

```
Dugum azamiAra(){
    Dugum sonuc = this;
    while (sonuc.sag != null)
        sonuc = sonuc.sag;
    return sonuc;
}
```

Arama ağacındaki en büyük elemanı bulan özyinelemeli algoritma


```
Dugum azamiAra(){
   if (sag == null)
      return this;
   else
      return sag.azamiAra();
}
```

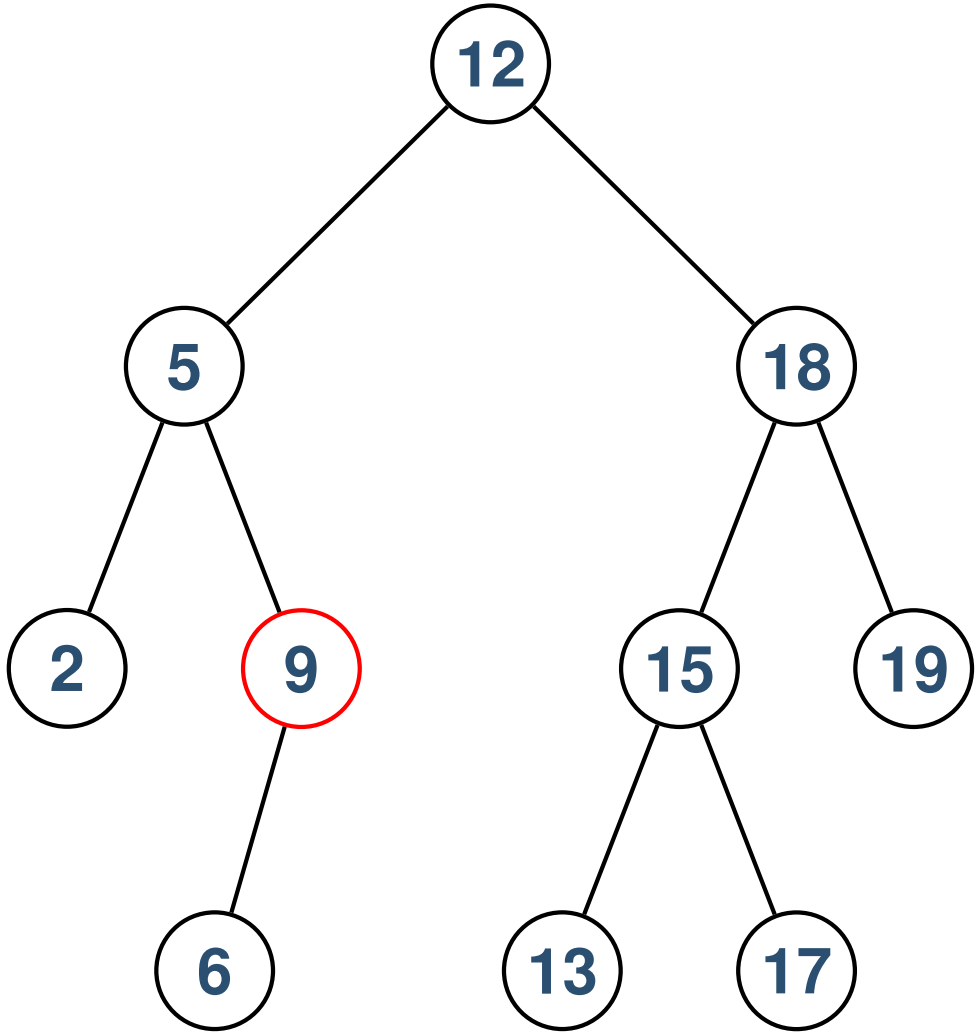
Örnek bir ikili arama ağacına 13 elemanının eklenmesi.



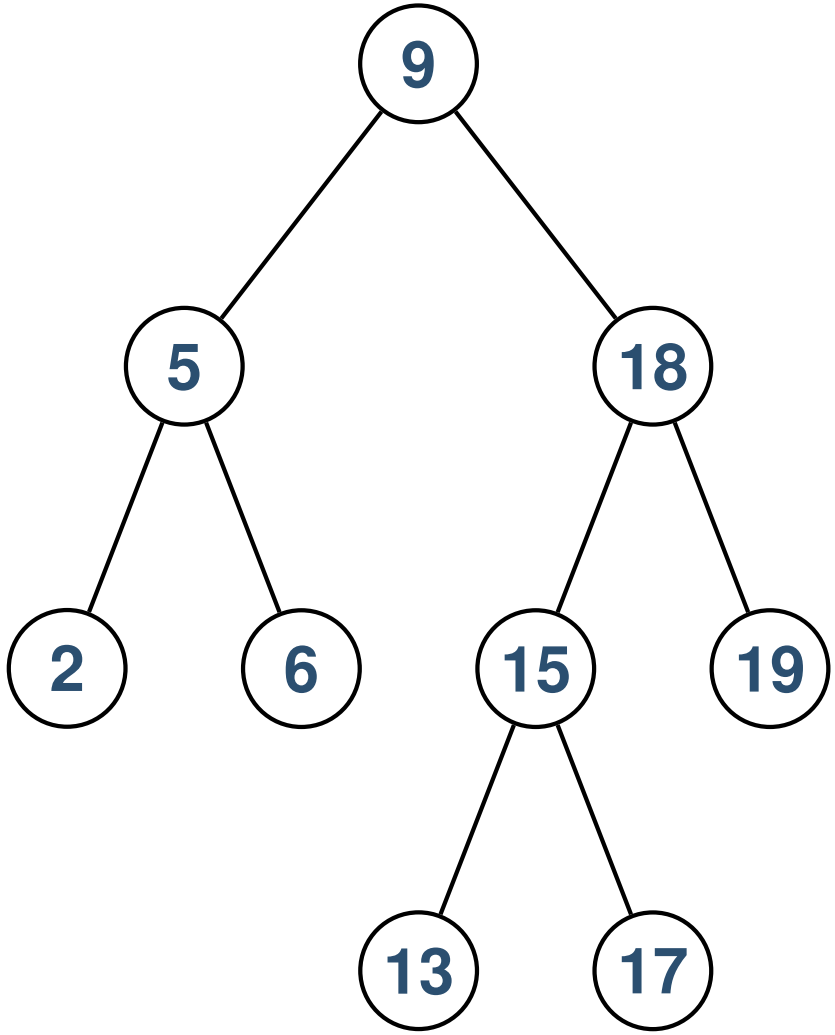
Arama ağacına yeni bir eleman ekleyen algoritma


```
1  void agacaEkle(Dugum yeni){
2      Dugum y = null;
3      Dugum x = kok;
4      while (x != null){
5          y = x;
6          if (yeni.icerik < x.icerik)
7              x = x.sol;
8          else
9              x = x.sag;
10     }
11     if (y == null)
12         kok = yeni;
13     else
14         if (yeni.icerik < y.icerik)
15             y.sol = yeni;
16         else
17             y.sag = yeni;
18 }
```

Örnek bir ikili arama ağacın kök elemanının silinmesi (1)



Örnek bir ikili arama ağacın kök elemanının silinmesi (2)



Arama ağacından bir eleman silen algoritma


```
1  void agacSil(int icerik){
2      Dugum y = null, x = kok;
3      while (x.icerik  !=  icerik ){
4          if (x. icerik  >  icerik )
5              x = x.sol ;
6          else
7              x = x.sag;
8      }
9      while (true){
10         if (x.sol  != null)
11             y = x.sol.azamiAra();
12         if (y == null && x.sag != null)
13             y = x.sag.asgariAra();
14         if (y == null)
15             break;
16         x. icerik  = y. icerik ;
17         x = y;
18     }
19 }
```

İkili Arama Ağacı İşlemleri

- Arama: $\mathcal{O}(\log N)$
- Ekleme: $\mathcal{O}(\log N)$
- Silme: $\mathcal{O}(\log N)$

Gezintiler

Önce gezinti algoritması


```
void onceGezinti(){
    System.out.print( icerik );
    if (sol  != null)
        sol.onceGezinti();
    if (sag != null)
        sag.onceGezinti();
}
```

Ara gezinti algoritması

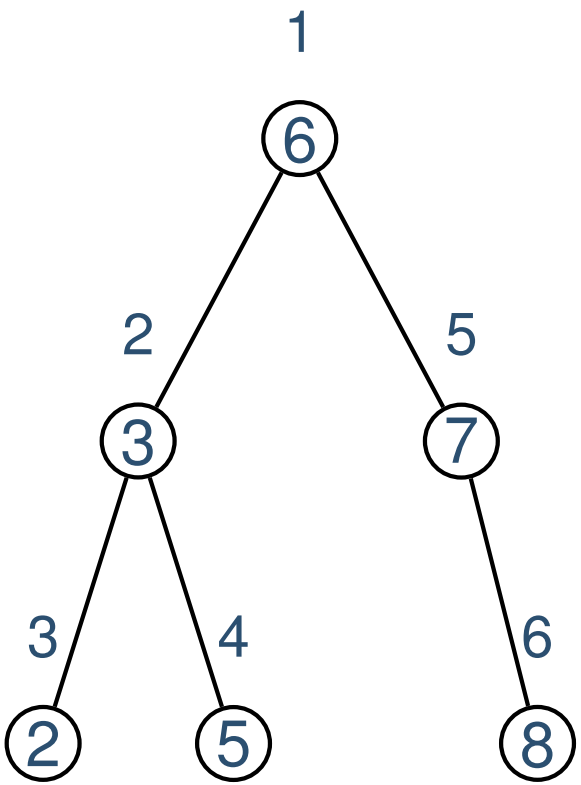

```
void araGezinti(){
    if (sol  != null)
        sol.araGezinti();
    System.out.print( icerik );
    if (sag != null)
        sag.araGezinti();
}
```

Sonra gezinti algoritması

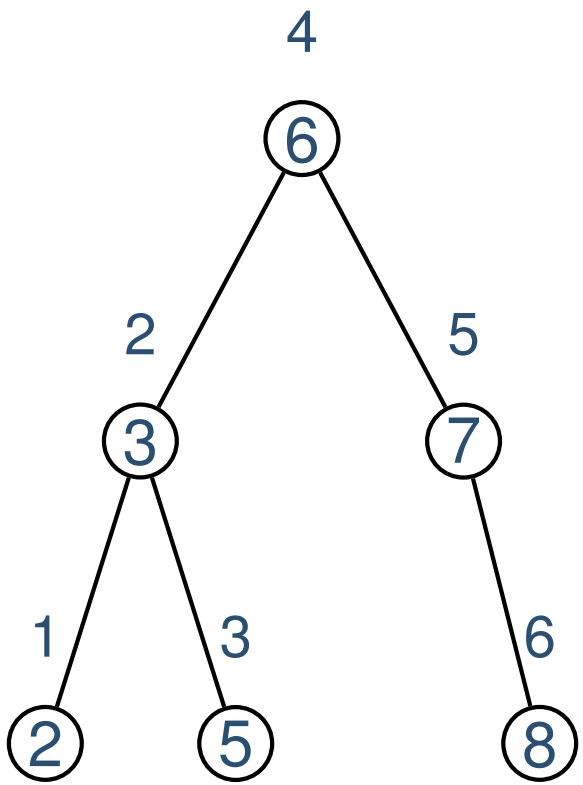

```
void sonraGezinti(){
    if (sol  != null)
        sol.sonraGezinti();
    if (sag != null)
        sag.sonraGezinti();
    System.out.print( icerik );
}
```

İkili arama ağacında değişik gezinti algoritmalarının düğümleri ziyaret ediş sıraları

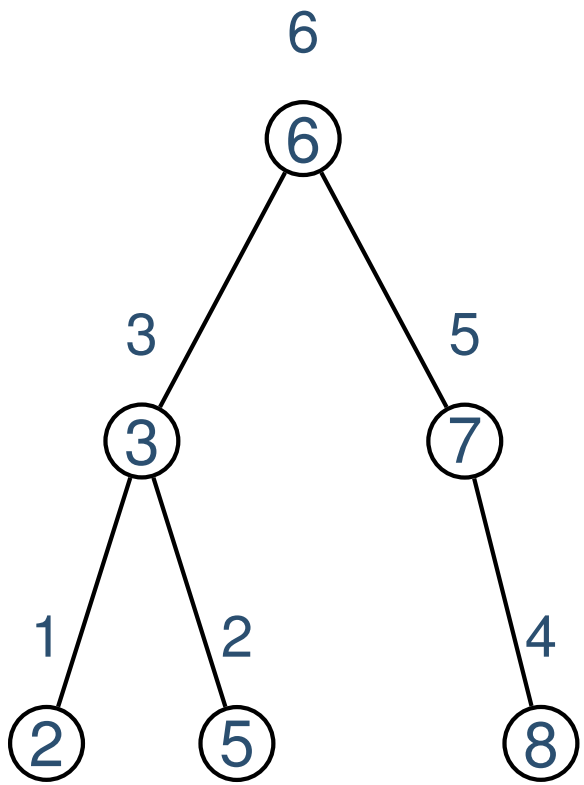
Önce Gezinti



Ara Gezinti



Sonra Gezinti



İçeriği bir ağaç düğümü (alt ağaç) olan eleman yapısı


```
public class Eleman{
    Dugum dugum;
    Eleman ileri;
    public Eleman(Dugum dugum){
        this.dugum = dugum;
        ileri = null;
    }
}
```

Bir ikili arama ağacındaki düğüm sayısını bulan algoritma (Çıkın) (1)


```
int dugumSayisi(){
    Dugum d;
    Eleman e;
    Cikin  c;
    int sayi = 0;
    c = new Cikin();
    d = kok;
    if (d != null){
        e = new Eleman(d);
        c.cikinEkle(e);
    }
```

Bir ikili arama ağacındaki düğüm sayısını bulan algoritma (Çıkın) (2)


```
12      while (!c.cikinBos()){
13          e = c. cikinSil  ();
14          d = e.dugum;
15          sayi++;
16          if (d.sol  != null){
17              e = new Eleman(d.sol);
18              c.cikinEkle(e);
19          }
20          if (d.sag != null){
21              e = new Eleman(d.sag);
22              c.cikinEkle(e);
23          }
24      }
25      return sayi;
26  }
```

Bir ikili arama ağacındaki düğüm sayısını bulan algoritma (Kuyruk) (1)

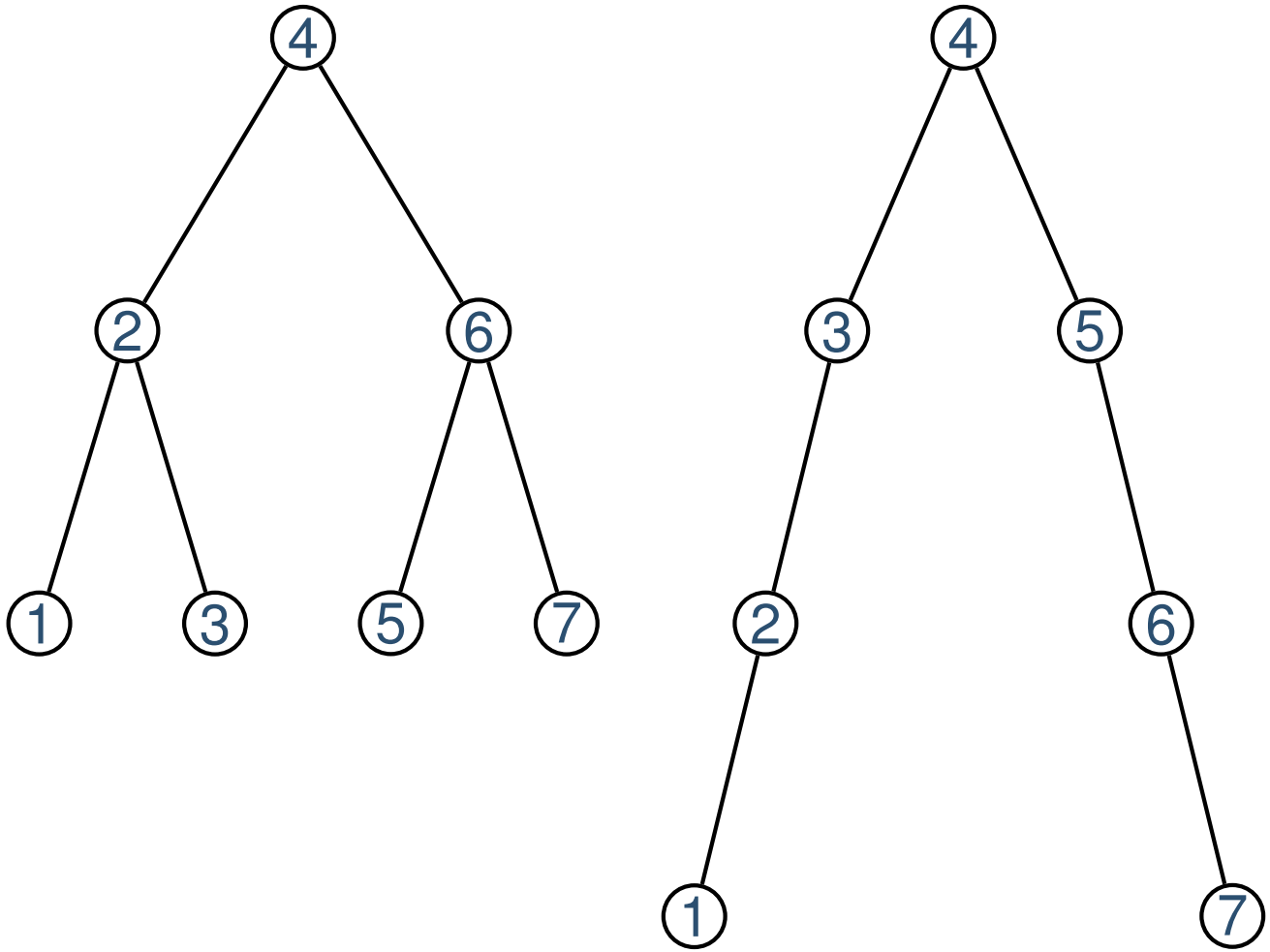

```
int dugumSayisi(){
    Dugum d;
    Eleman e;
    Kuyruk k;
    int sayi = 0;
    k = new Kuyruk();
    d = kok;
    if (d != null){
        e = new Eleman(d);
        k.kuyrugaEkle(e);
    }
```

Bir ikili arama ağacındaki düğüm sayısını bulan algoritma (Kuyruk) (2)

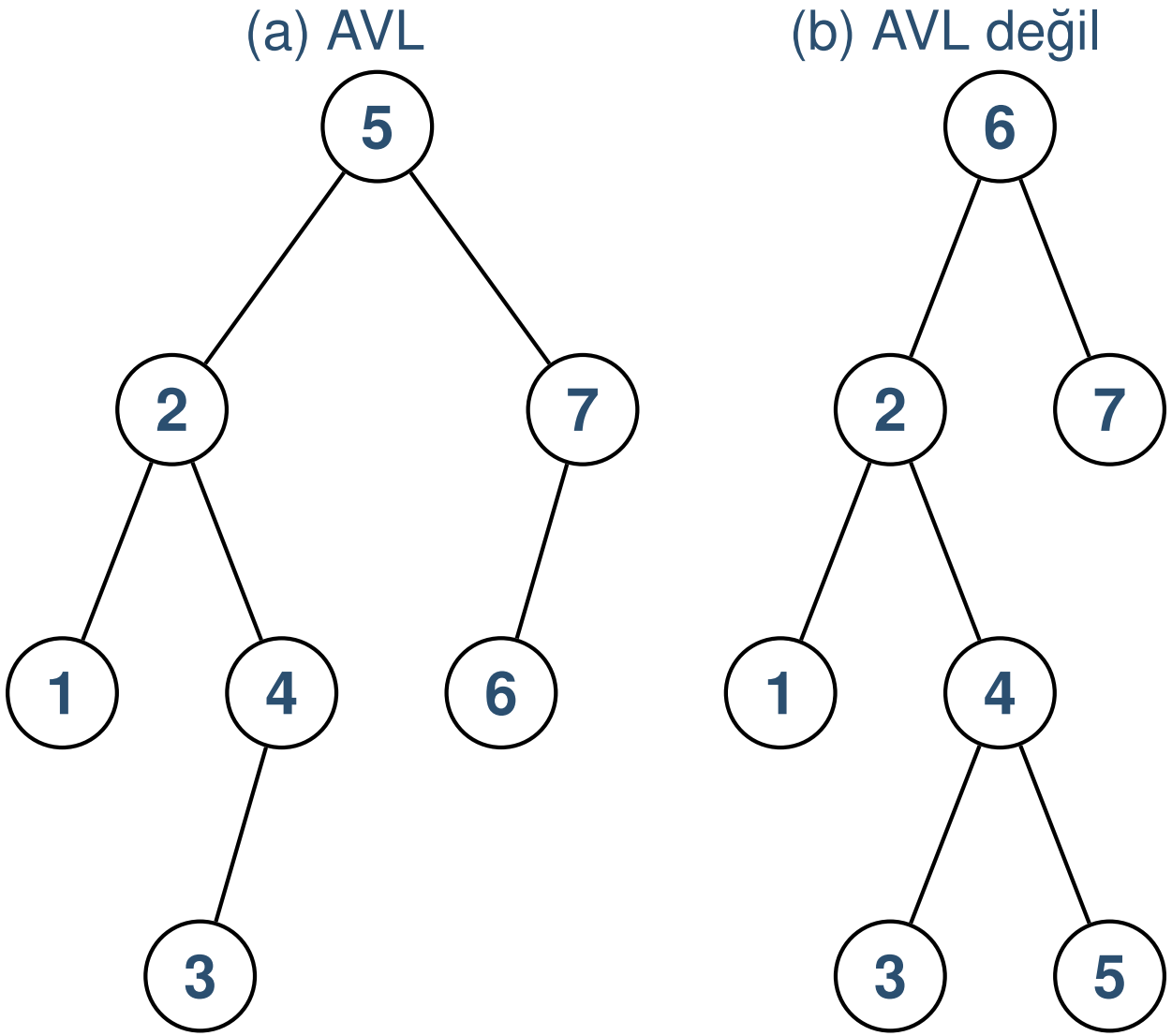

```
12      while (!k.kuyrukBos()){
13          e = k.kuyrukSil ();
14          d = e.dugum;
15          sayi++;
16          if (d.sol  != null){
17              e = new Eleman(d.sol);
18              k.kuyrugaEkle(e);
19          }
20          if (d.sag != null){
21              e = new Eleman(d.sag);
22              k.kuyrugaEkle(e);
23          }
24      }
25      return sayi;
26  }
```

AVL Ağacı

1'den 7'ye kadar olan sayıların farklı sırada ikili arama ağacına eklenmesiyle oluşan iki değişik ağaç



İkili arama ağaçları



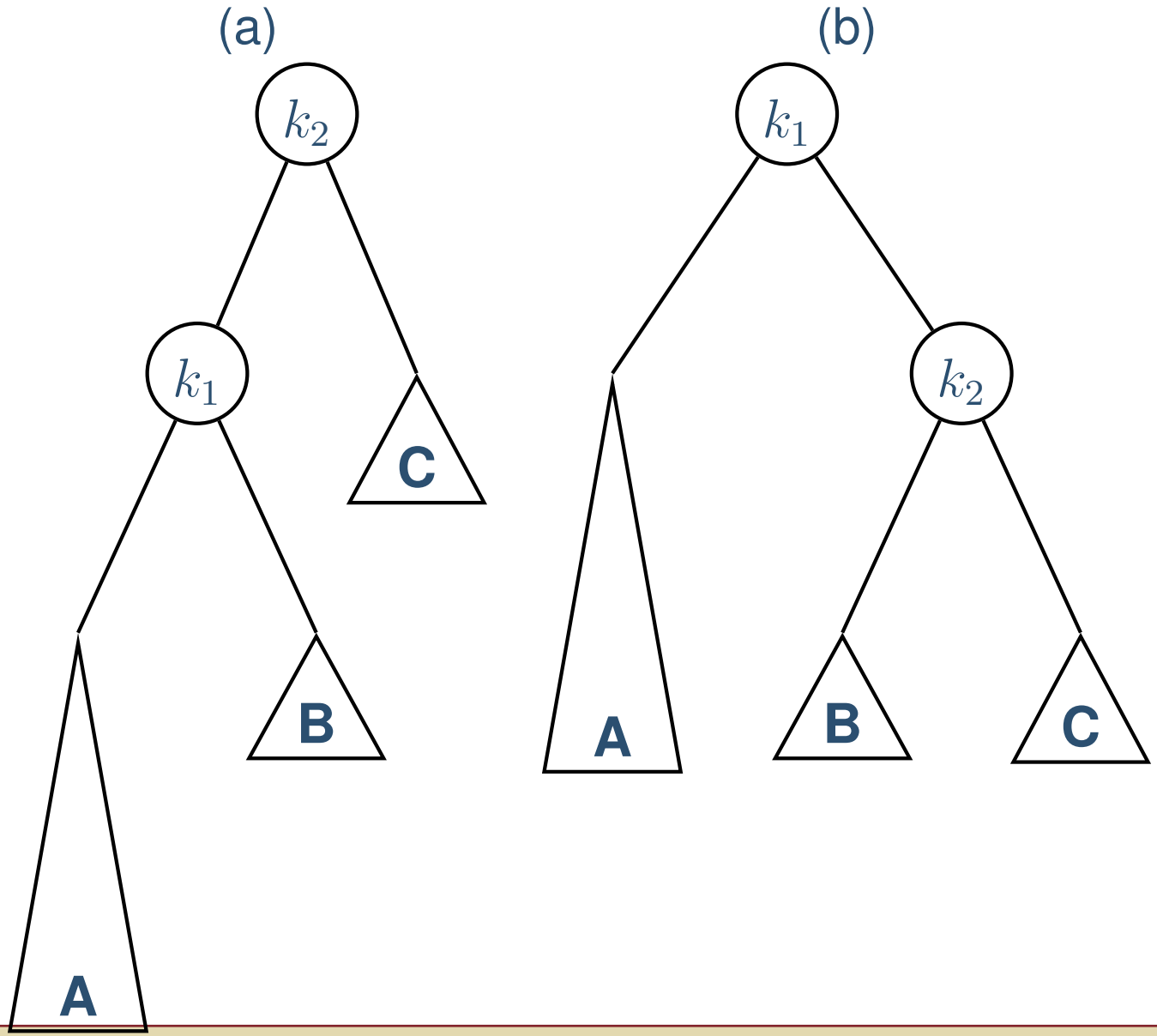
Tam sayılar içeren AVL düğümü tanımı


```
1  public class AvlDugum{
2      int icerik;
3      int boy;
4      AvlDugum sol;
5      AvlDugum sag;
6      public AvlDugum(int icerik){
7          this.icerik = icerik ;
8          sol = null;
9          sag = null;
10         boy = 1;
11     }
12 }
```

AVL ağacı tanımı


```
1  public class AvlAgac{
2      AvlDugum kok;
3      public AvlAgac(){
4          kok = null;
5      }
6  }
7  int boy(AvlDugum d){
8      if (d == null)
9          return 0;
10     else
11         return d.boy;
12 }
```

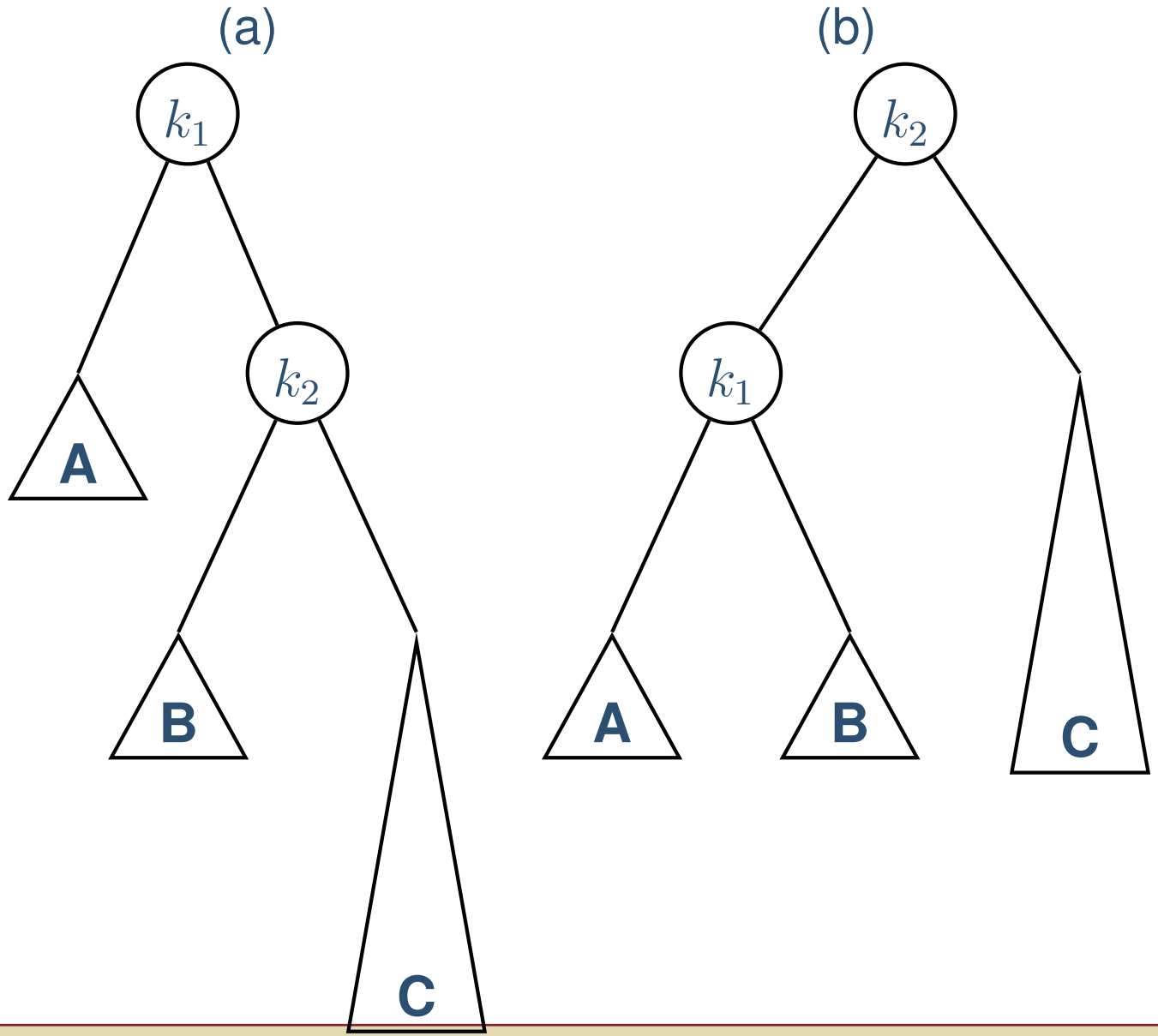
Durum 1'i çözmek için uygulanan tek rotasyon



Durum 1'i çözmek için uygulanan tek rotasyon algoritması


```
AvlDugum solTekRotasyon(AvlDugum k2){
    AvlDugum k1 = k2.sol;
    k2.sol  = k1.sag;
    k1.sag = k2;
    k2.boy = azami(boy(k2.sol), boy(k2.sag)) + 1;
    k1.boy = azami(boy(k1.sol), k1.sag.boy) + 1;
    return k1;
}
```

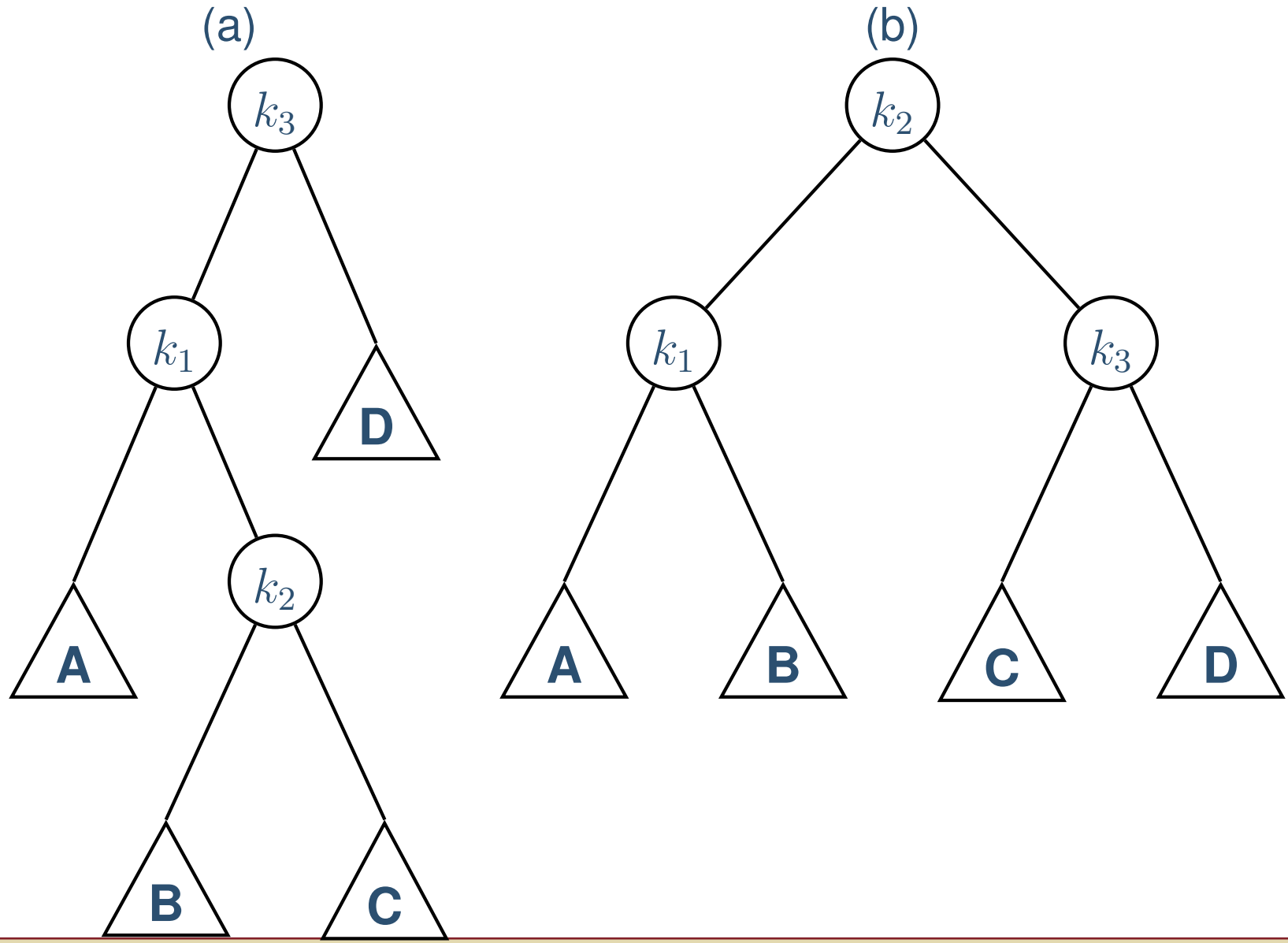
Durum 4'ü çözmek için uygulanan tek rotasyon



Durum 4'ü çözmek için uygulanan tek rotasyon algoritması

```
AvlDugum sagTekRotasyon(AvlDugum k1){
    AvlDugum k2 = k1.sag;
    k1.sag = k2.sol;
    k2.sol  = k1;
    k2.boy = azami(k2.sol.boy, boy(k2.sag)) + 1;
    k1.boy = azami(boy(k1.sol), boy(k1.sag)) + 1;
    return k2;
}
```

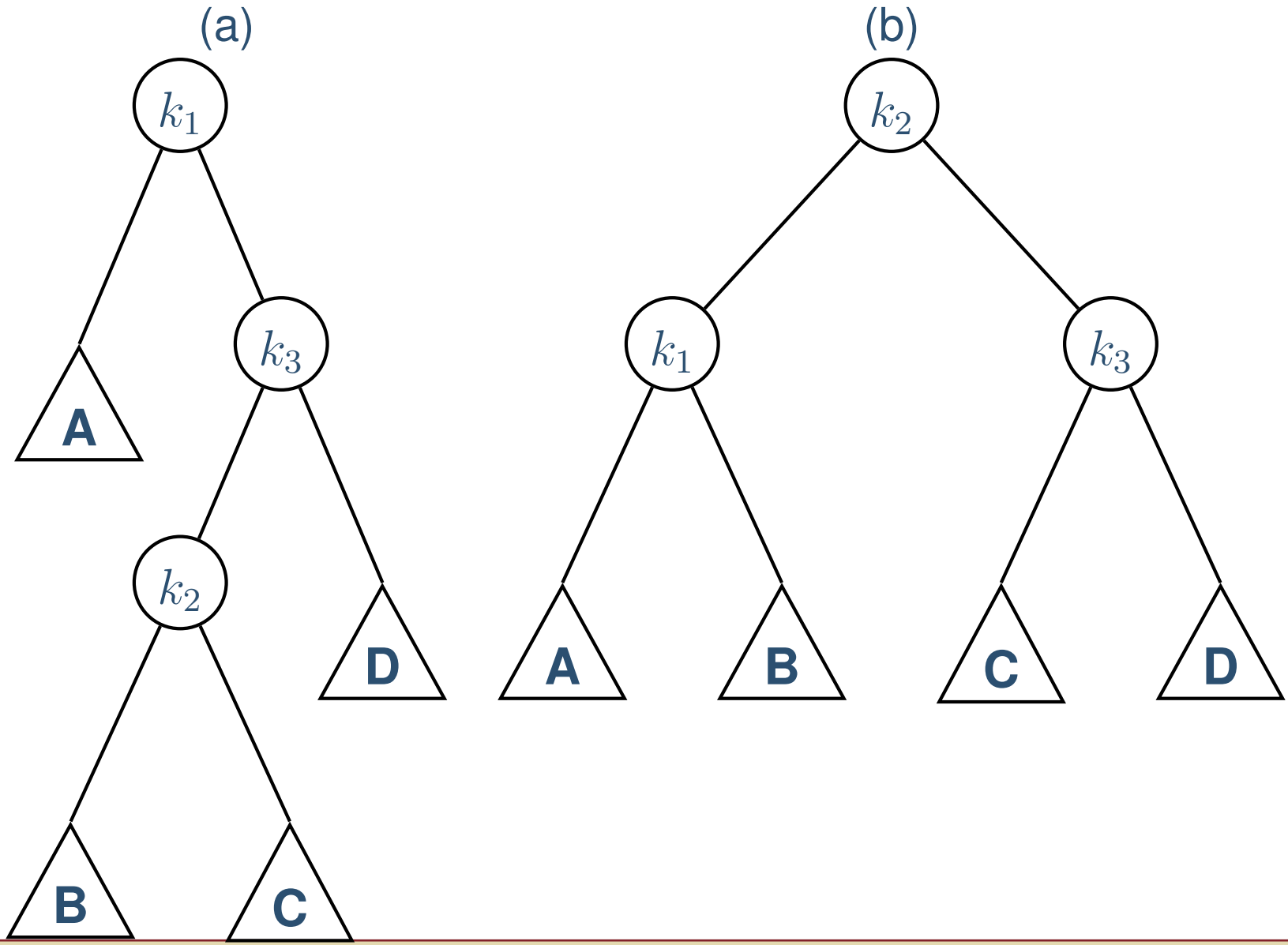
Durum 2'yi çözmek için uygulanan çift rotasyon



Durum 2'yi çözmek için uygulanan çift rotasyon algoritması

```
1 AvlDugum solCiftRotasyon(AvlDugum k3){
2     k3.sol  = sagTekRotasyon(k3.sol);
3     return solTekRotasyon(k3);
4 }
```

Durum 3'ü çözmek için uygulanan çift rotasyon



Durum 3'ü çözmek için uygulanan çift rotasyon algoritması


```
1 AvlDugum sagCiftRotasyon(AvlDugum k1){
2     k1.sag = solTekRotasyon(k1.sag);
3     return sagTekRotasyon(k1);
4 }
```

AVL ağacına yeni bir eleman ekleyen algoritma (Java) (1)

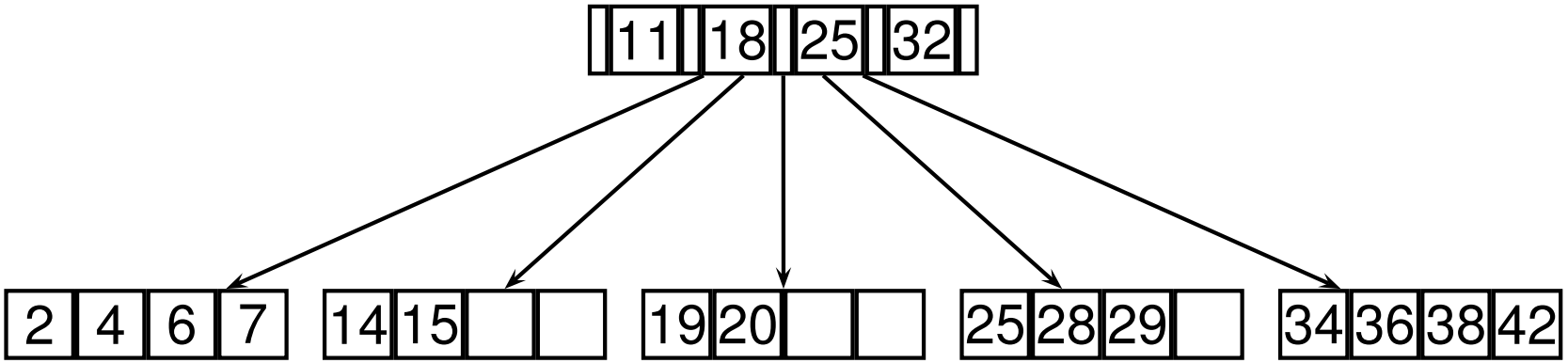

```
1  void agacaEkle(Avldugum yeniEleman){
2      Avldugum y = null, x = kok, t;
3      Eleman e;
4      int yon1 = 0, yon2 = 0;
5      Cikin c = new Cikin(100);
6      while (x != null){
7          y = x;
8          e = new Eleman(y);
9          c.cikinEkle(e);
10         yon1 = yon2;
11         if (yeniEleman.icerik < x. icerik ){
12             x = x.sol;
13             yon2 = SOL;
14         }else{
15             x = x.sag;
16             yon2 = SAG;
17         }
18     }
19     cocukYerlestir(y, yeniEleman);
```

AVL ağacına yeni bir eleman ekleyen algoritma (Java) (2)

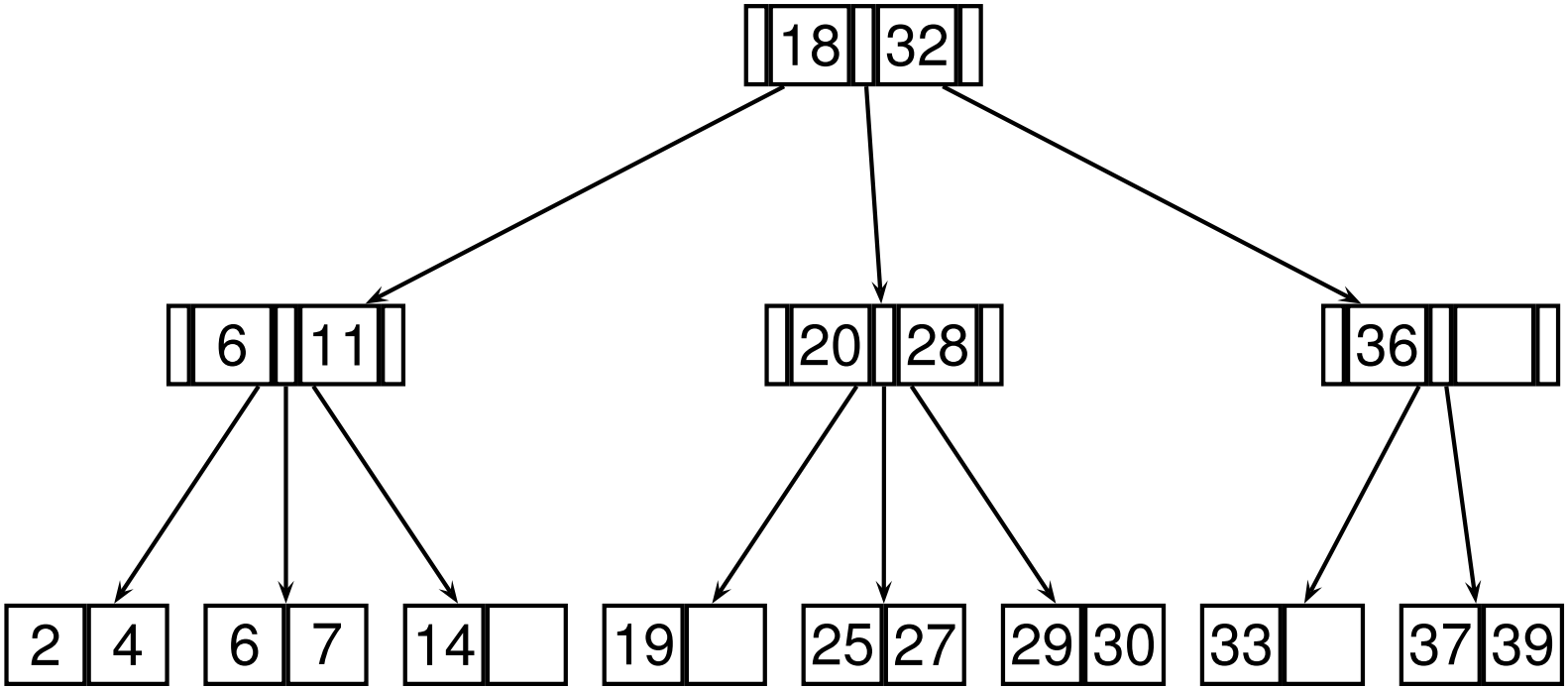

```
20      while (!c.cikinBos()){
21          e = c. cikinSil ();
22          x = e.dugum;
23          x.boy = azami(boy(x.sol), boy(x.sag)) + 1;
24          if (Math.abs(boy(x.sol) - boy(x.sag)) == 2){
25              if (yon1 == SOL && yon2 == SOL)
26                  t = solTekRotasyon(x);
27              if (yon1 == SOL && yon2 == SAG)
28                  t = solCiftRotasyon(x);
29              if (yon1 == SAG && yon2 == SOL)
30                  t = sagCiftRotasyon(x);
31              if (yon1 == SAG && yon2 == SAG)
32                  t = sagTekRotasyon(x);
33              e = c. cikinSil ();
34              y = e.dugum;
35              cocukYerlestir(y, t);
36              break;
37          }
38      }
39  }
```

B+ Ağacı

15 veri içeren d= 2 dereceli örnek bir B+ ağacı



13 veri içeren d= 1 dereceli örnek bir B+ ağacı



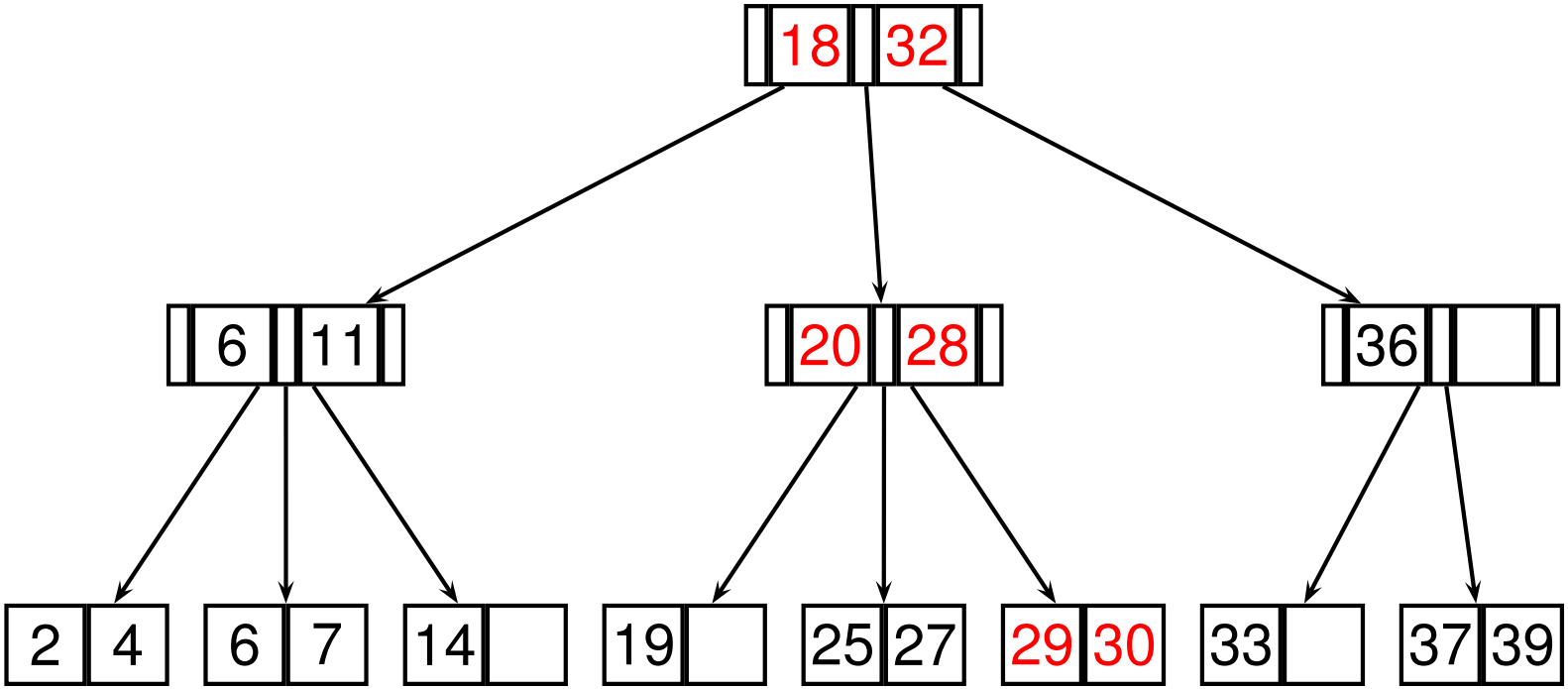
B+ düğümü tanımı


```
1  public class BDugum{
2      int[] K;
3      int m;
4      int d;
5      boolean yaprak;
6      BDugum[] nesil;
7      public BDugum(int d){
8          m = 0;
9          this.d = d;
10         yaprak = true;
11         K = new int[2 * d + 1];
12         nesil  = new BDugum[2 * d + 1];
13     }
14 }
```

B+ ağacı tanımı


```
1 public class BAgac{
2     BDugum kok;
3     public BAgac(){
4         kok = null;
5     }
6 }
```

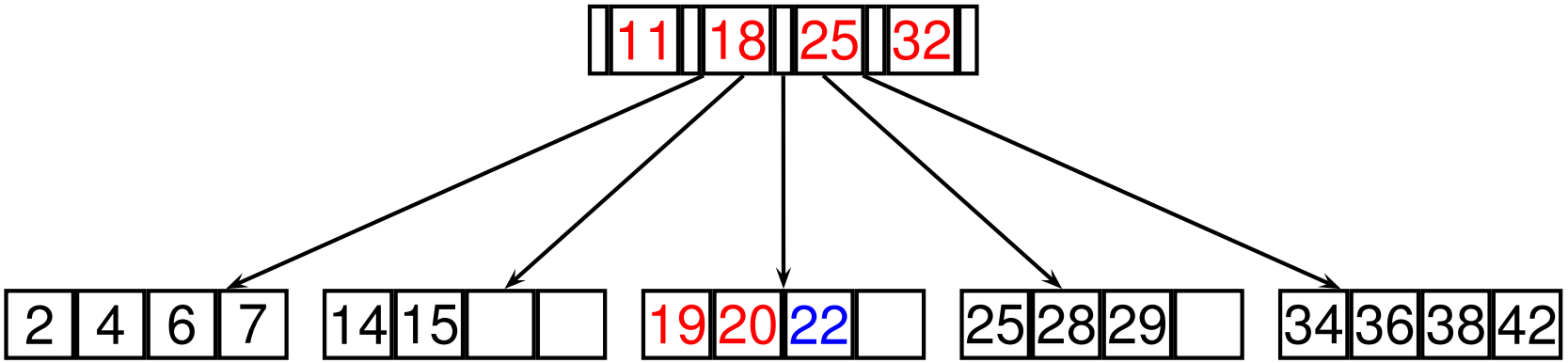
Önceki şekildeki B+ ağacında 30'un aranması



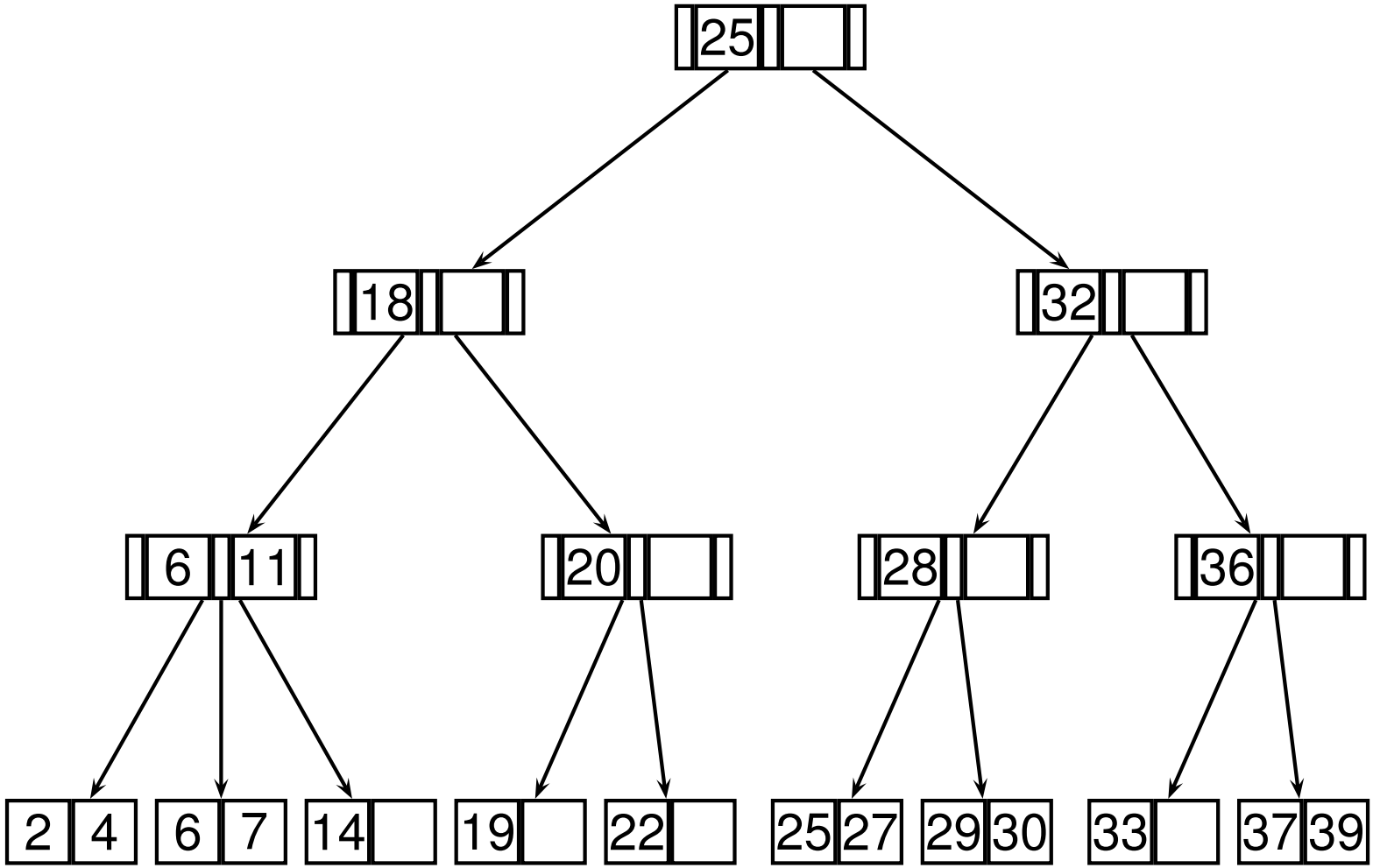
Verilen bir değeri B+ ağacında arayan algoritma


```
1  BDugum agacAra(int eleman){
2      int cocuk;
3      BDugum b;
4      b = kok;
5      while (!b.yaprak){
6          cocuk = b.pozisyon(eleman);
7          b = b.nesil[cocuk];
8      }
9      return b;
10 }
11 int pozisyon(int eleman){
12     int i;
13     if (eleman >= K[m - 1])
14         return m;
15     else
16         for (i = 0; i < m; i++)
17             if (eleman < K[i])
18                 return i;
19 }
```

Önceki şekildeki ağaca 22'nin eklenmesi



Önceki şekildeki ağaca 22'nin eklenmesi



Verilen bir değeri B+ düğümüne ekleyen algoritma (1)


```
1  BDugum dugumeEkle(BAgac a, int eleman){
2      BDugum s, yeni;
3      int cocuk, i;
4      cocuk = pozisyon(eleman);
5      if (!nesil[cocuk].yaprak)
6          s = nesil[cocuk].dugumeEkle(a, eleman);
7      else
8          s = nesil[cocuk].yapragaEkle(eleman);
9      if (s == null)
10         return null;
11     for (i = m; i > cocuk; i --)
12         K[i] = K[i - 1];
13     K[cocuk] = s.K[2 * d];
14     if (m < 2 * d){
15         for (i = m + 1; i > cocuk; i --)
16             nesil[i] = nesil[i - 1];
17         nesil[cocuk] = s;
18         m++;
19         return null;
```

Verilen bir değeri B+ düğümüne ekleyen algoritma (2)


```
20      } else {
21          yeni = new BDugum(d);
22          for (i = 0; i < d; i++)
23              yeni.K[i] = K[d + i + 1];
24          yeni.K[2 * d] = K[d];
25          for (i = 0; i < d; i++)
26              yeni.nesil[i] = nesil[d + i + 1];
27          yeni.m = d;
28          m = d;
29          if (this == kok){
30              a.kok = new BDugum(d);
31              a.kok.m = 1;
32              a.kok.nesil[0] = this;
33              a.kok.nesil[1] = yeni;
34              a.kok.K[0] = this.K[d];
35              return null;
36          } else
37              return yeni;
38      }
39  }
```

Verilen bir değeri B+ yaprağına ekleyen algoritma


```
1  BDugum yapragaEkle(int eleman){
2      int i, cocuk;
3      BDugum yeni;
4      cocuk = pozisyon(eleman);
5      for (i = m; i > cocuk; i --)
6          K[i] = K[i - 1];
7      K[cocuk] = eleman;
8      if (m < 2 * d){
9          m++;
10         return null;
11     } else {
12         yeni = new BDugum(d);
13         for (i = 0; i < d + 1; i++)
14             yeni.K[i] = K[d + i];
15         yeni.K[2 * d] = K[d];
16         yeni.m = d + 1;
17         m = d;
18         return yeni;
19     }
20 }
```

Uygulama: Ağaç Dizini

Veritabanı

Öğrenci Sistemi veritabanı

- Üniversiteye kayıtlı olan lisans, yüksek lisans, doktora vs. tüm öğrencilerin bilgilerini içeren **ogrenci** tablosunu,
- Üniversitede ders veren tüm öğretim görevlilerinin bilgilerini içeren **gorevli** tablosunu,
- Üniversitede o güne kadar açılmış olan tüm derslerin bilgilerini içeren **ders** tablosunu,
- Üniversiteye kayıtlı olan tüm öğrencilerin o güne kadar almış olduğu tüm derslerin notlarını içeren **not** tablosunu,
- . . .

içerecektir.

Sorgu

Numarası 18 olanın ad ve soyadını **ogrenci** tablosundan

```
SELECT Ad, Soyad
FROM Ogrenci
WHERE No = 18
```

sorgusuyla, numarası 23'ten büyük tüm öğrencilerin sayısını ise yine **ogrenci** tablosundan

```
SELECT Count(*)
FROM Ogrenci
WHERE No > 23
```

sorgusuyla öğreniriz.

Öğrenci bilgilerini (no, ad, soyad) içeren düğüm yapısı.


```
1  public class Dugum{
2      String  ad;
3      String  soyad;
4      int icerik;
5      Dugum sol;
6      Dugum sag;
7      public Dugum(int no, String ad, String soyad){
8          this.icerik = no;
9          this.ad = ad;
10         this.soyad = soyad;
11         sol  = null;
12         sag = null;
13     }
14 }
```

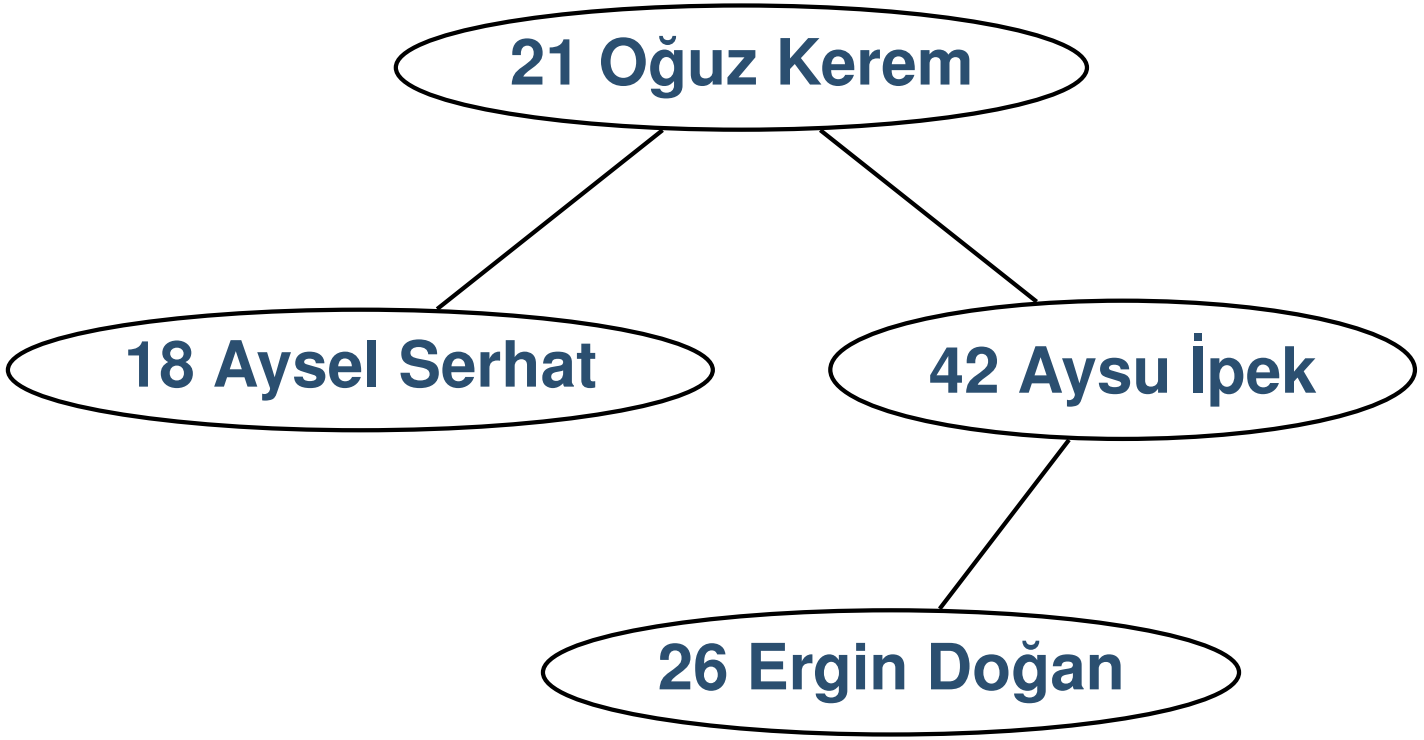
Örnek öğrenci bilgileri dosyası

```
4
21 Oğuz Kerem
18 Aysel Serhat
42 Aysu İpek
26 Ergin Doğan
```

Öğrenci dosyasındaki bilgilerle ikili arama ağacını doldurmak


```
1  Agac dosyaOku(){
2      Scanner dosya;
3      Dugum d;
4      String  ad;
5      String  soyad;
6      int no, yas, i ,  sayi;
7      Agac agac;
8      dosya = new Scanner(new File("ogrenci.txt"));
9      sayi  = dosya.nextInt();
10     agac = new Agac();
11     for (i  = 0;  i  < sayi;  i++){
12         no = dosya.nextInt();
13         ad = dosya.next();
14         soyad = dosya.next();
15         d = new Dugum(no, ad, soyad);
16         agac.agacaEkle(d);
17     }
18     return agac;
19 }
```

Örnek öğrenci dosyasındaki bilgilerle doldurulan ikili arama ağacı



Numarası 18 olan öğrencinin ad ve soyadını bulan yinelemesiz fonksiyon


```
1  void sorgu(Agac a){
2      Dugum d;
3      d = a.kok;
4      while (d != NULL){
5          if (d.icerik  < 18)
6              d = d.sag;
7          else
8              if (d.icerik  > 18)
9                  d = d.sol;
10             else{
11                 System.out.print(d.ad);
12                 System.out.print(d.soyad);
13                 break;
14             }
15     }
16 }
```

Numarası 23'ten büyük kaç öğrenci olduğunu bulan fonksiyonlar


```
1  int sorgu2(Dugum d){
2      int sayi = 0;
3      if (d. icerik  > 23){
4          sayi  = 1;
5          if (d.sol  != null)
6              sayi  += sorgu2(d.sol);
7      }
8      if (d.sag != null)
9          sayi  += sorgu2(d.sag);
10     return sayi;
11 }
12 int sorgu(Agac a){
13     if (a.kok != null)
14         return sorgu2(a.kok);
15     else
16         return 0;
17 }
```