YZM 2116
Veri Yapıları

Yrd. Doç. Dr. Deniz KILINÇ
Celal Bayar Üniversitesi
Hasan Ferdi Turgutlu Teknoloji Fakültesi
Yazılım Mühendisliği

BÖLÜM - 8

Bu bölümde,

- Problem Tanımı
- Arama Ağaçları
- İkili Arama Ağacı
- İkili Arama Ağacı Gerçekleştirim
- Arama
- Ekleme
- Min ve Maks Değer Bulma
- Silme
- Uygulamalar

konusuna değinilecektir.

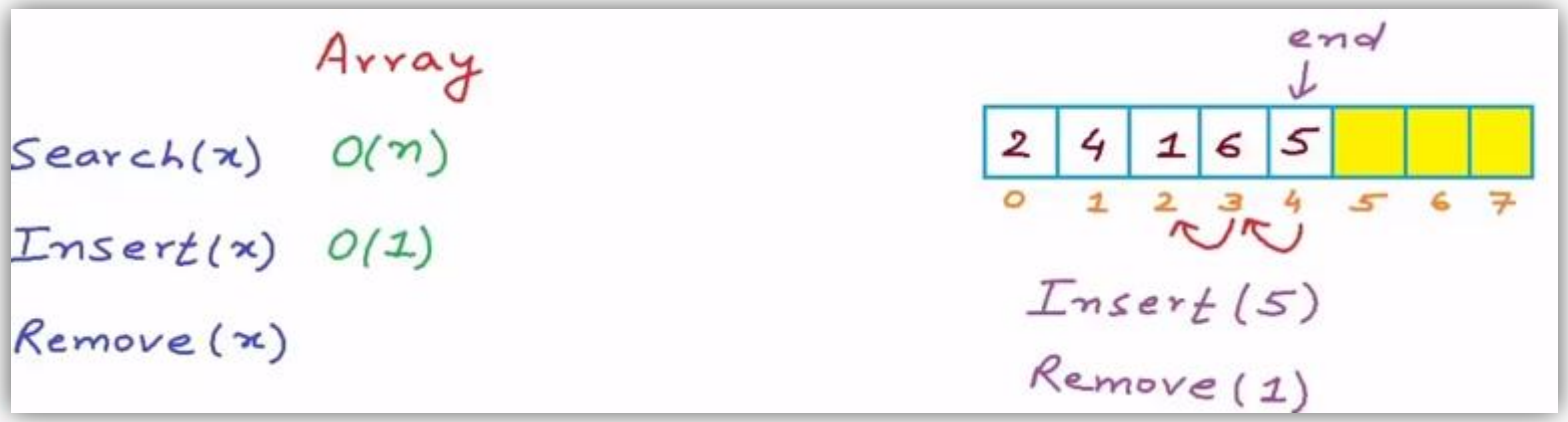
Soru

- Elimizde bir grup veri varsa (mesela **1 kilo**) ve bu veriler üzerinde
  - Arama
  - Silme
  - Değiştirme

gibi işlemler yapılmak isteniyorsa hangi veri yapısını seçmeliyiz?

Dizi? Bağlı Liste?

Cevap 1: Dizi



- DİKKAT
  - **Insert (x)** için dizinin dolması durumunda karmaşıklığın artacağını ve O(n) olacağını dikkate alınız.
  - **Remove (x)** için en kötü durumda (n-1) eleman kaydırılacağı için O(n) olur.

Cevap 2: Bağlı Liste

YENİ SORU?

- Her iki veri yapısında da arama performansını nasıl arttırırız?

CEVAP:

- Elemanları sırala ve İkili Arama (Binary Search) kullan.

İkili Arama Cevabı Üzerine…

	Array (unsorted)	Linked List	Array (sorted)
Search(x)	O(n)	O(n)	O(log n)
Insert(x)	O(1)	O(1)	O(n)
Remove(x)	O(n)	O(n)	O(n)

YENİ SORU?

- Peki **Ekleme (insert)** ve **Silme (remove)** işlemleri için performans iyileştirme yapılabilir mi?

CEVAP:

- **İkili arama ağacıyla** mümkün olabilir.
- **Örneğin:** İkili Arama Ağacı (Binary Search Tree) ile…

Arama Ağaçları

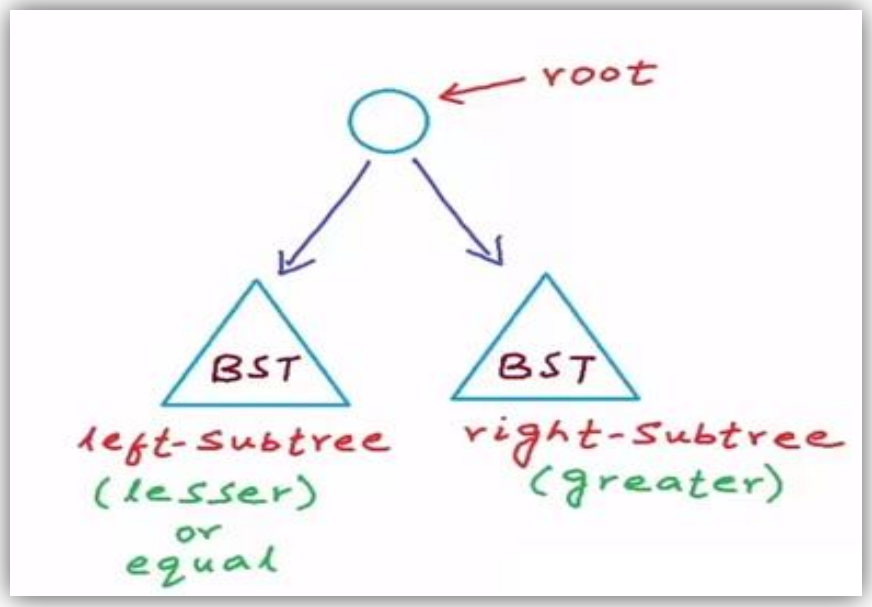
- Arama, gezinme, ekleme ve silme gibi işlemleri destekleyen ağaçlar, **arama ağacı** olarak *adlandırılır*.
- Bu bağlamda arama ağacı, belirli sayıda düzenli/sıralı elemana/değere **sıralı erişimi** *sağlar*.
- Bir arama ağacı, *verilerin düğümlere* ***belli bir düzende*** *eklenmesiyle* **elde edilir**.
- Arama ağaçlarının **önemli avantajı**, *veri girişinin disipline edilmesi sayesinde*, aranan elemana ulaşmak için **tüm ağacın** *dolaşılmasına (traverse)* **gerek olmamasıdır**.

Arama Ağaçları (devam…)

- Bilinen arama ağaçları aşağıdaki gibidir:
  1. İkili Arama Ağacı (Binary search tree (BST))
  2. AVL Ağacı
  3. Splay Ağacı
  4. 2-3-4 Ağacı
  5. Red-Black Ağacı
  6. B Ağacı ve B+ Ağacı

İkili Arama Ağacı (Binary Search Tree)

- Özyinelemeli (**recursive**) olarak BST, tüm elemanların kök referans olmak şartı ile:
  - i. **Kökten küçükse <u>kökün solunda</u>**
  - ii. **Kökten büyükse <u>kökün sağında</u>** yer almasıdır.

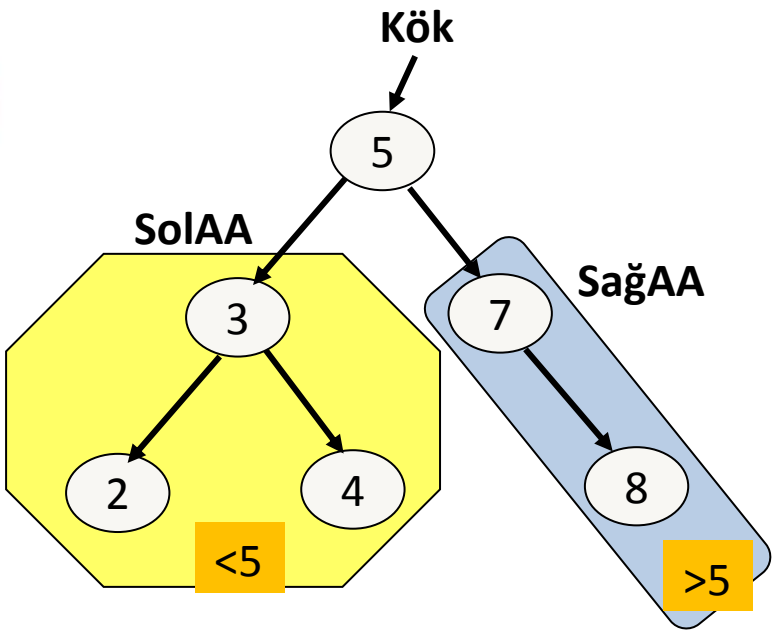


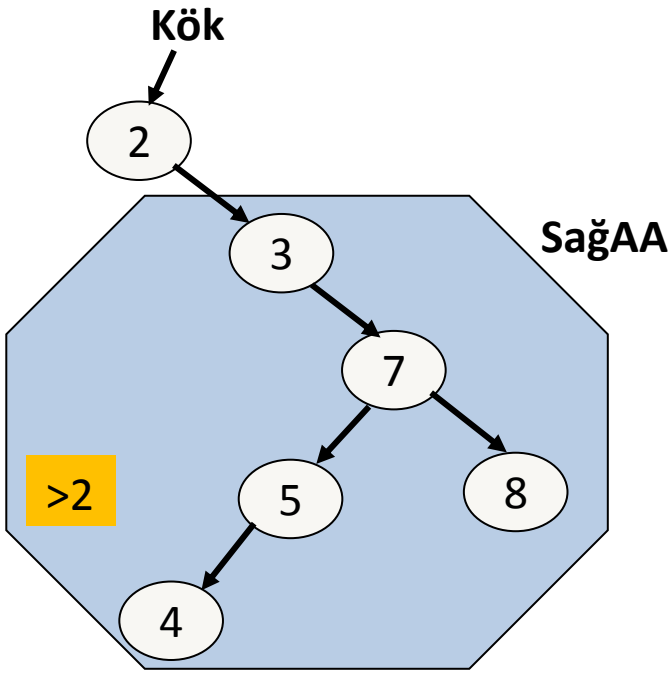
İkili Arama Ağacı (BST) (devam…)

x: ikili arama ağacında herhangi bir düğüm olsun.

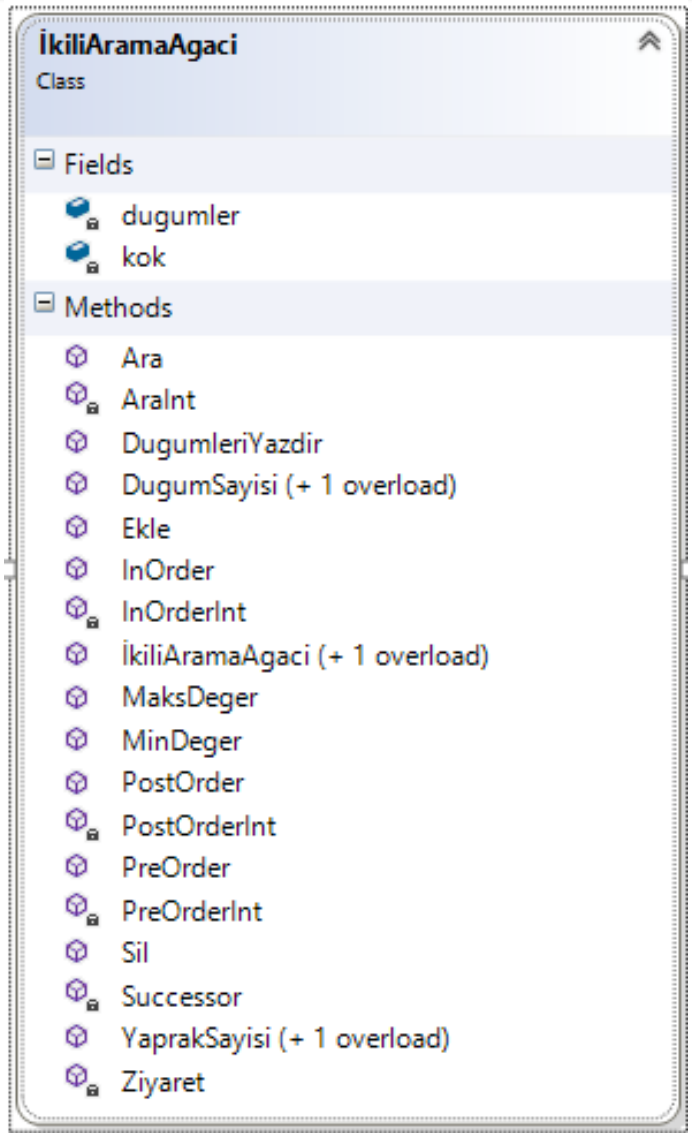
Eğer, y düğümü x'in sol alt ağacında ise **key[y] ≤ key[x]**

Eğer, y düğümü x'in sağ alt ağacında ise **key[y] ≥ key[x]**





İkili Arama Ağacı Gerçekleştirim



İkili Arama Ağacı Üzerinde Bazı İşlemler

- Dolaşma/Gezinme (Traversal)
- Ara
- MinDeger
- MaksDeger
- SonraGelenEnKucuk (Successor)
- Ekle
- Sil
- DugumSayisi
- YaprakSayisi
- Vb.

İkili Arama Ağacı - Gezinme

- İkili ağaç ile aynı mantıkta çalışır:
  - **Önce-kök/ziyaret (Preorder - NLR):**
    - ✓ **Kök/ziyaret, Sol, Sağ**
  - **Ortada-kök/ziyaret (Inorder - LNR):**
    - ✓ **Sol, Kök/ziyaret, Sağ**
    - ✓ **Not:** Inorder gezinme, düğüm değerlerinin **sıralı elde edilmesini** sağlar.
  - **Sonra-kök/ziyaret (Postorder - LRN):**
    - ✓ **Sol, Sağ, Kök/ziyaret**

İkili Arama Ağacı – Arama

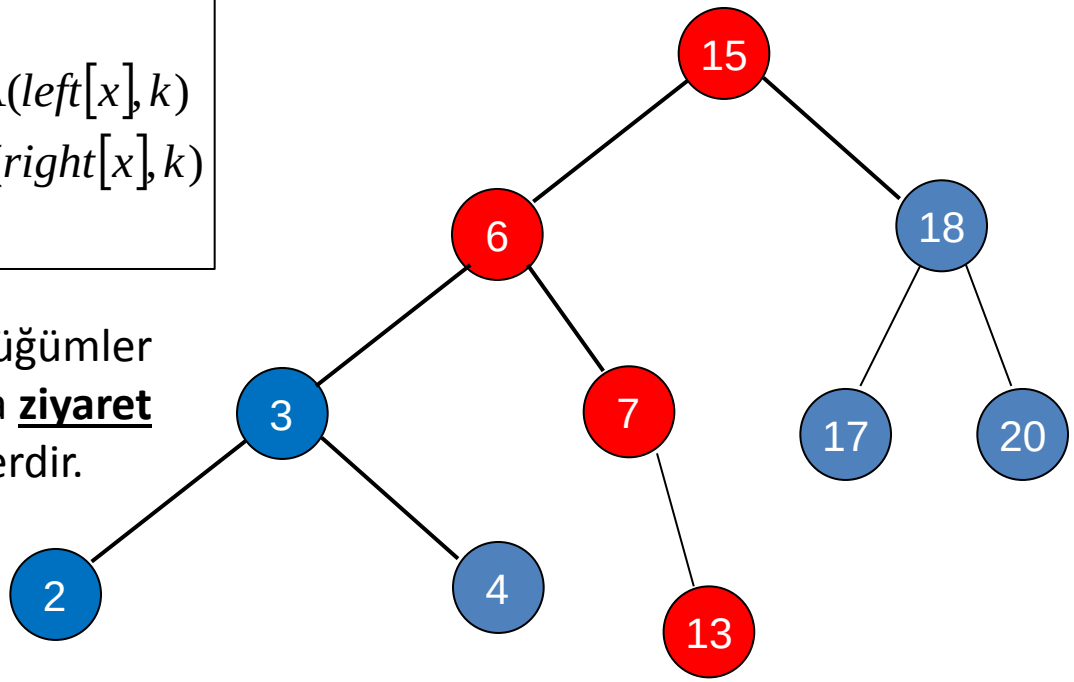
- Bir **k anahtar değerini aramak** için, *kök düğümden başlanarak* aşağı doğru bir yol izlenir.
- Bir sonraki ziyaret edilecek düğüm, k anahtar değerinin, geçerli düğümün anahtar değeriyle *karşılaştırılması sonucuna bağlıdır*.
  - Aranan değer, kök düğümden küçükse sol alt ağaç üzerinden aramaya devam edilir.
  - Aranan değer, kök düğümden büyükse sağ alt ağaç üzerinden aramaya devam edilir.
- Eğer yaprağa ulaşıldıysa **anahtar bulunamamıştır** ve **null değer** geri döndürülür.

İkili Arama Ağacı – Arama (devam…)

```
ARA(x, k)
1  if x = NILL or key = key[x]
2     then return x
3  if  k < key[x]
4     then return ARA(left[x], k)
5     else return ARA(right[x], k)
```

- Kırmızı renkli düğümler arama sırasında **ziyaret edilen** düğümlerdir.

Aranan sayı, k = 13



İkili Arama Ağacı – Arama (devam…)

```
public İkiliAramaAgacDugumu Ara(int anahtar)
{
    return AraInt(kok, anahtar);
}
3 references
private İkiliAramaAgacDugumu AraInt(İkiliAramaAgacDugumu dugum,
                                    int anahtar)
{
    if (dugum == null)
        return null;
    else if ((int)dugum.veri == anahtar)
        return dugum;
    else if ((int)dugum.veri > anahtar)
        return (AraInt(dugum.sol, anahtar));
    else
        return (AraInt(dugum.sag, anahtar));
}
```

İkili Arama Ağacı – MinDeger

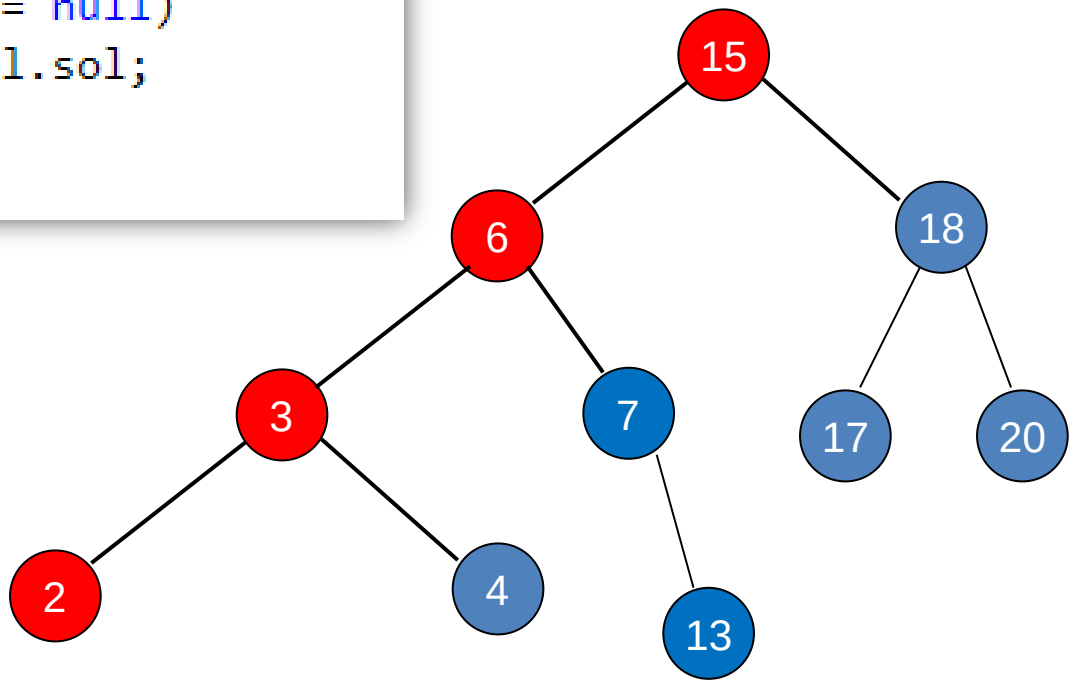
- Ağaçtaki en küçük elemanı içeren düğümü bulur ve geri döndürür.
  - En küçük elemanı içeren düğüm **en soldaki düğümde** bulunur.
  - Kökten başlayarak **devamlı sola gidilerek** bulunur.

```
MINDEGER (x)
1  while left[x] ≠ NILL
2      do x ← left[x]
3  return x
```

İkili Arama Ağacı – MinDeger (devam…)

```
public İkiliAramaAgacDugumu MinDeger()
{
    İkiliAramaAgacDugumu tempSol = kok;
    while (tempSol.sol != null)
        tempSol = tempSol.sol;
    return tempSol;
}
```

MinDeger = 2



İkili Arama Ağacı – MaksDeger

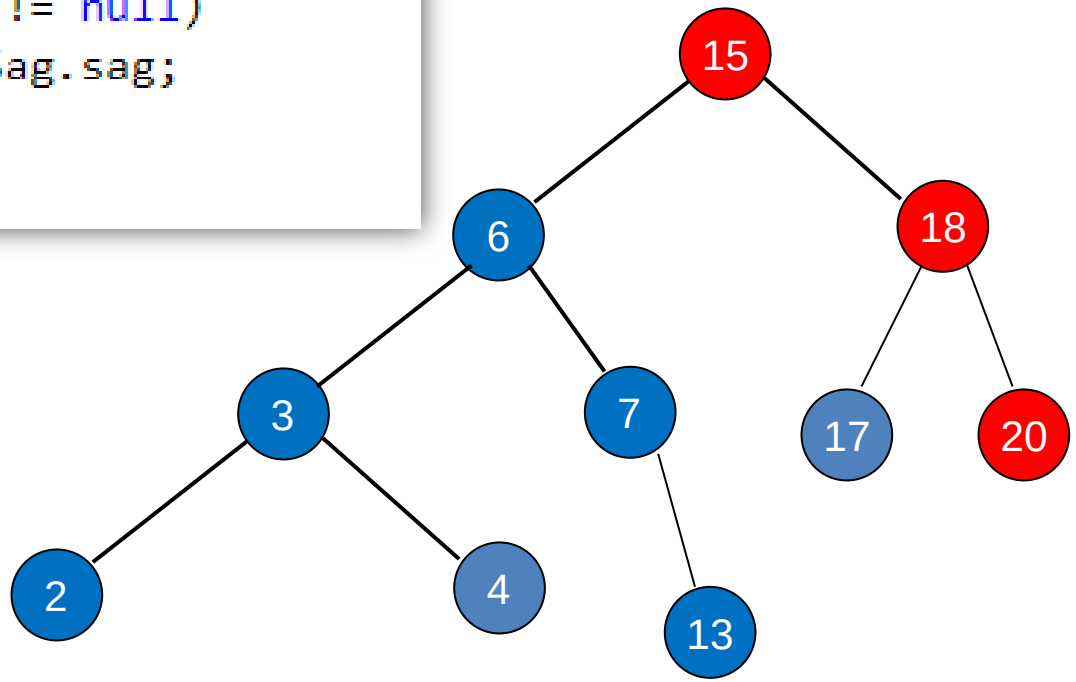
- Ağaçtaki en büyük elemanı içeren düğümü bulur ve geri döndürür.
  - En büyük elemanı içeren düğüm **en sağdaki düğümde** bulunur.
  - Kökten başlayarak **devamlı sağa gidilerek** bulunur.

```
MAKSDEGER (x)
1  while right[x] ≠ NILL
2      do x ← right[x]
3  return x
```

İkili Arama Ağacı – MaksDeger (devam…)

```
public İkiliAramaAgacDugumu MaksDeger()
{
    İkiliAramaAgacDugumu tempSag = kok;
    while (tempSag.sag != null)
        tempSag = tempSag.sag;
    return tempSag;
}
```

MaksDeger = 20



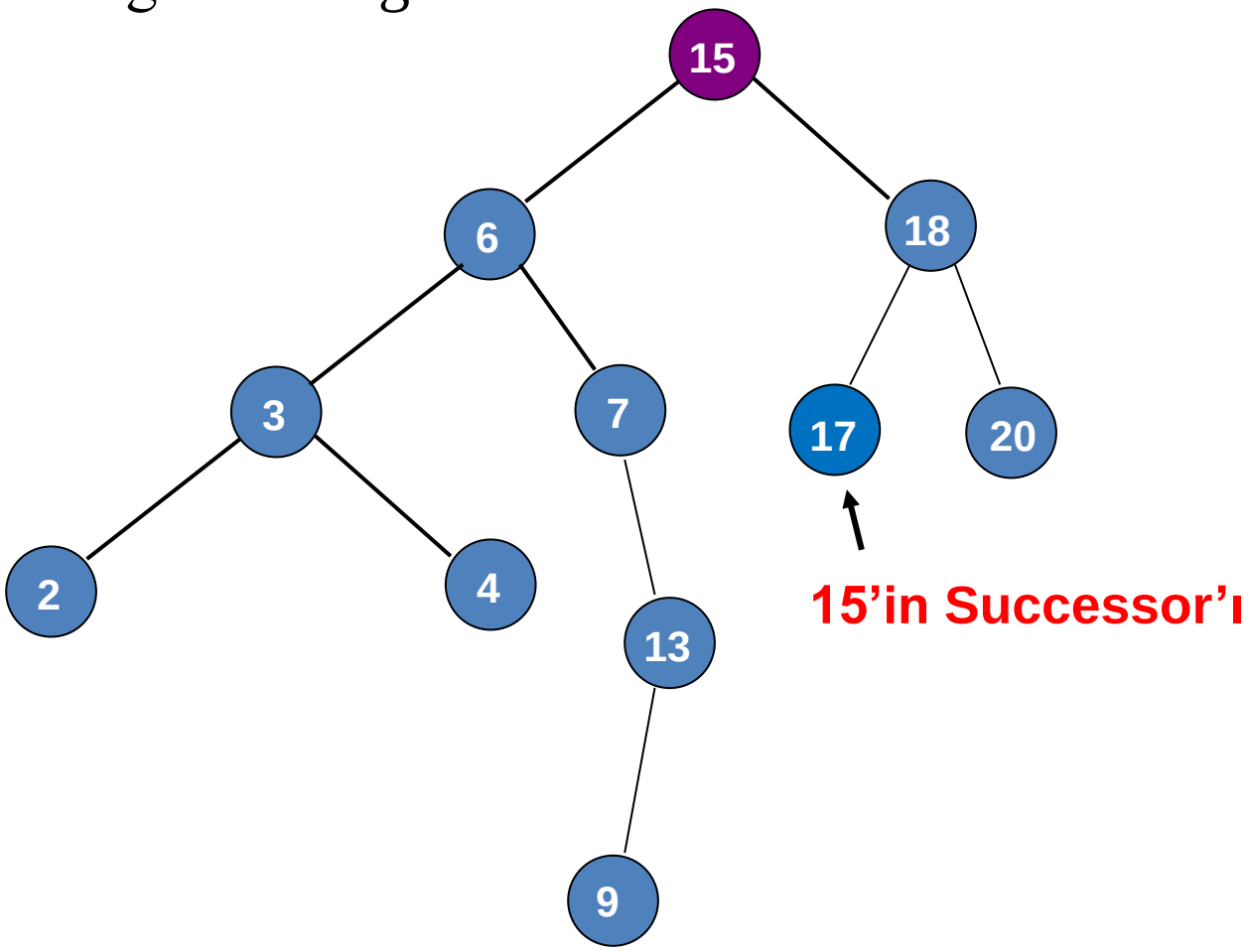
İkili Arama Ağacı – SonraGelenEnKucuk

- Bir düğümün kendinden sonra gelen (kendinden büyük) en küçük değeri yani **successor'u** geriye döndürülür.
- Kendinden büyük bu değerler, düğümlerin sağ alt-ağaçlarında olduğu için düğümün sağ-alt ağacının olup olmamasına göre algoritma değişiklik gösterir.

```
SONRA - GELEN - EN - KUCUK(x)
1  if right[x] ≠ NILL
2     then return MINDEGER(right[x])
3  y ← p[x]
4  while y ≠ NILL and x = right[y]
5        do x ← y
6           y ← p[y]
7  return y
```

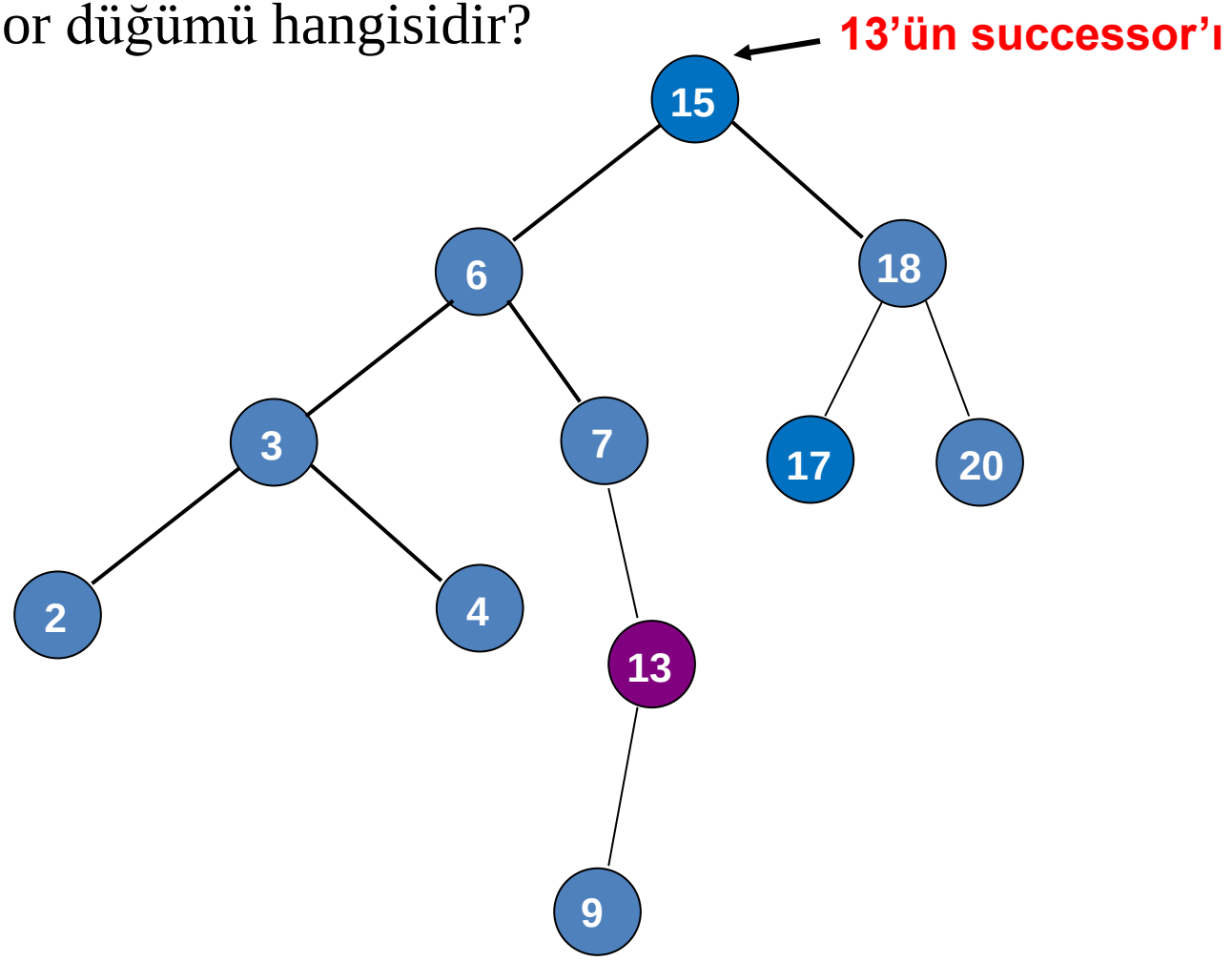
İkili Arama Ağacı – SonraGelenEnKucuk (devam…)

- 15'in successor düğümü hangisidir?



İkili Arama Ağacı – SonraGelenEnKucuk (devam…)

- 13'ün successor düğümü hangisidir?

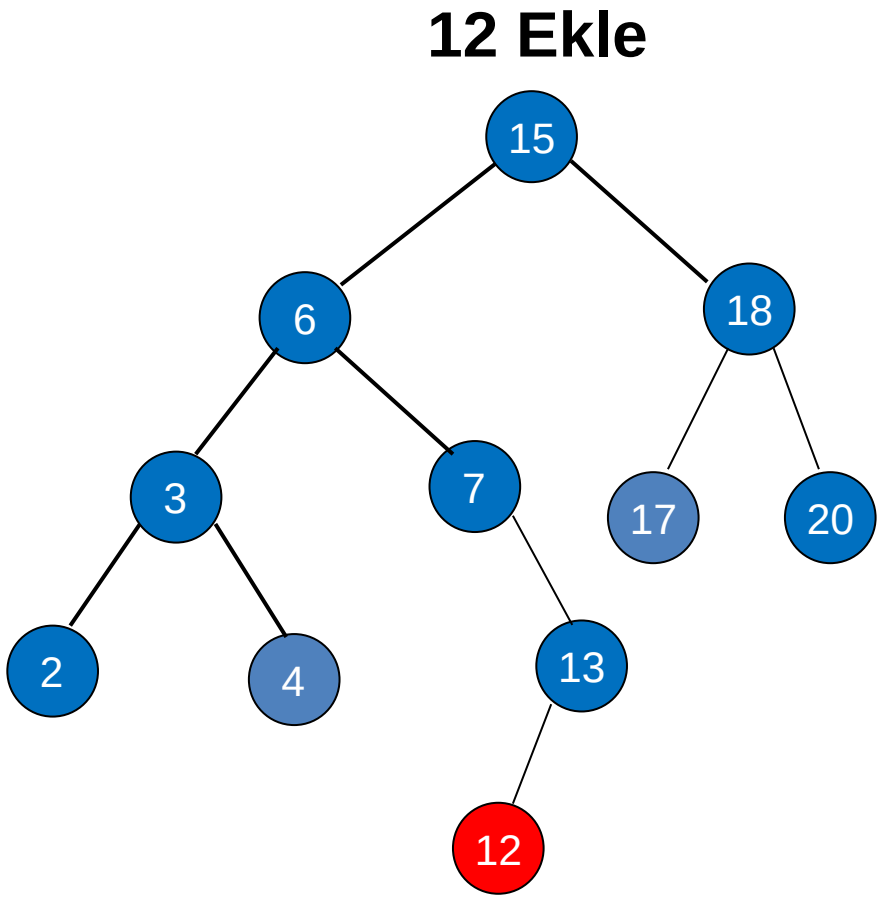


İkili Arama Ağacı – Ekleme

- Eklenecek değeri içeren **"z"** isimli yeni bir düğüm oluştur.
- Kökten başlayarak ağaç üzerinde *eklenecek sayıyı arıyormuş gibi* aşağıya doğru ilerle.
- **Yeni düğüm**, **aramanın bittiği düğümün çocuğu olmalıdır**.

İkili Arama Ağacı – Ekleme (devam…)

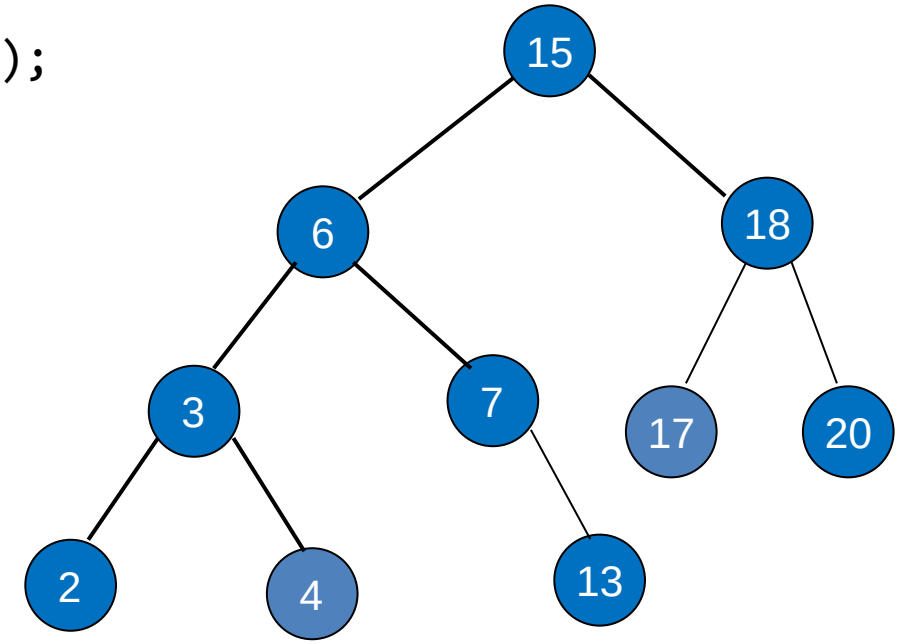
```
EKLE(T, z)
1   y ← NILL
2   x ← root[T]
3   while x ≠ NILL
4         do y ← x
5              if key[z] < key[x]
6                   then x ← left[x]
7                   else x ← right[x]
8   p[z] ← y
9   if y = NILL
10       then root[T] ← z
11       else if key[z] < key[y]
12                 then left[y] ← z
13                 else right[y] ← z
```



İkili Arama Ağacı – Ekleme (devam…)

- **Soru:** Aşağıdaki ağacı ekleme fonksiyonu kullanarak nasıl oluştururuz?

```
a = new İkiliAramaAgaci();
a.Ekle(15);
a.Ekle(6);
a.Ekle(3);
a.Ekle(2);
a.Ekle(4);
a.Ekle(7);
a.Ekle(13);
a.Ekle(18);
a.Ekle(17);
a.Ekle(20);
```



Ekleme sırası size tanıdık geldi mi?

İkili Arama Ağacı – Ekleme

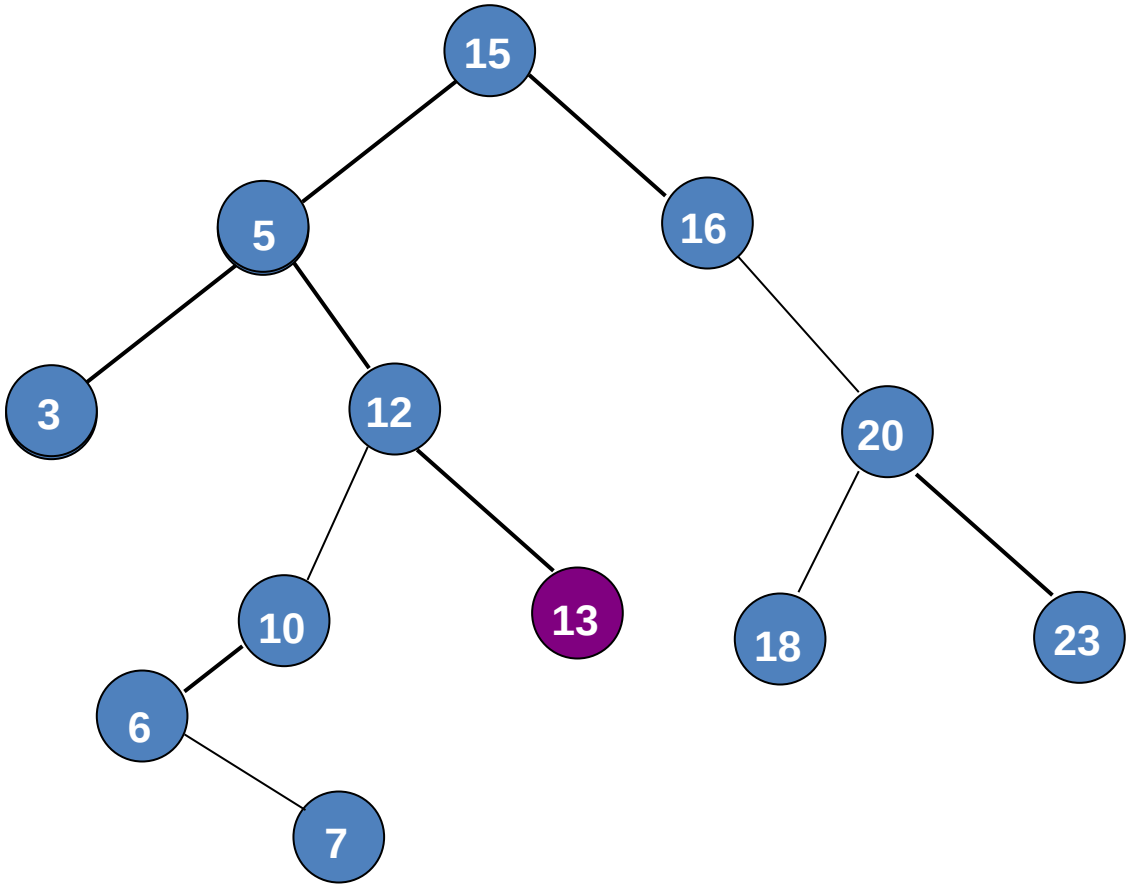
```
public void Ekle(int deger)
{
    //Yeni eklenecek düğümün parent'ı
    İkiliAramaAgacDugumu tempParent = new İkiliAramaAgacDugumu();
    //Kökten başla ve ilerle
    İkiliAramaAgacDugumu tempSearch = kok;

    while (tempSearch != null)
    {
        tempParent = tempSearch;
        //Deger zaten var, çık.
        if (deger == (int)tempSearch.veri)
            return;
        else if (deger < (int)tempSearch.veri)
            tempSearch = tempSearch.sol;
        else
            tempSearch = tempSearch.sag;
    }
    İkiliAramaAgacDugumu eklenecek = new İkiliAramaAgacDugumu(deger);
    //Ağaç boş, köke ekle
    if (kok == null)
        kok = eklenecek;
    else if (deger < (int)tempParent.veri)
        tempParent.sol = eklenecek;
    else
        tempParent.sag = eklenecek;
}
```

İkili Arama Ağacı – Silme

- Silme işlemi diğer işlemlere göre daha karmaşıktır.
- Silme işleminde 3 durum bulunmaktadır:
  1. Silinecek düğümün hiç çocuğu yoksa (**yaprak düğüm**)
  2. Silinecek düğümün 1 çocuğu varsa
  3. Silinecek düğümün 2 çocuğu varsa

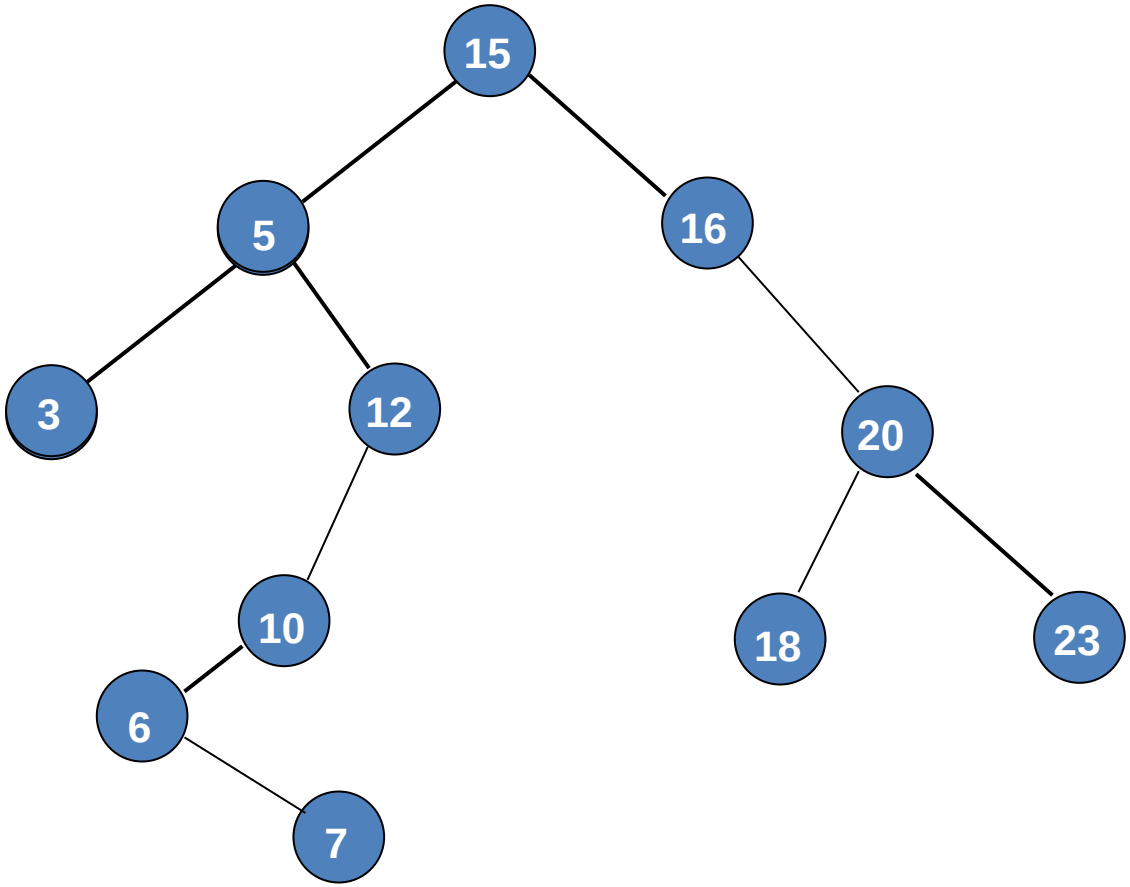
Durum 1 – Yaprak Düğümü Silme

- **Soru:** 13 değerine sahip düğüm nasıl sileriz?



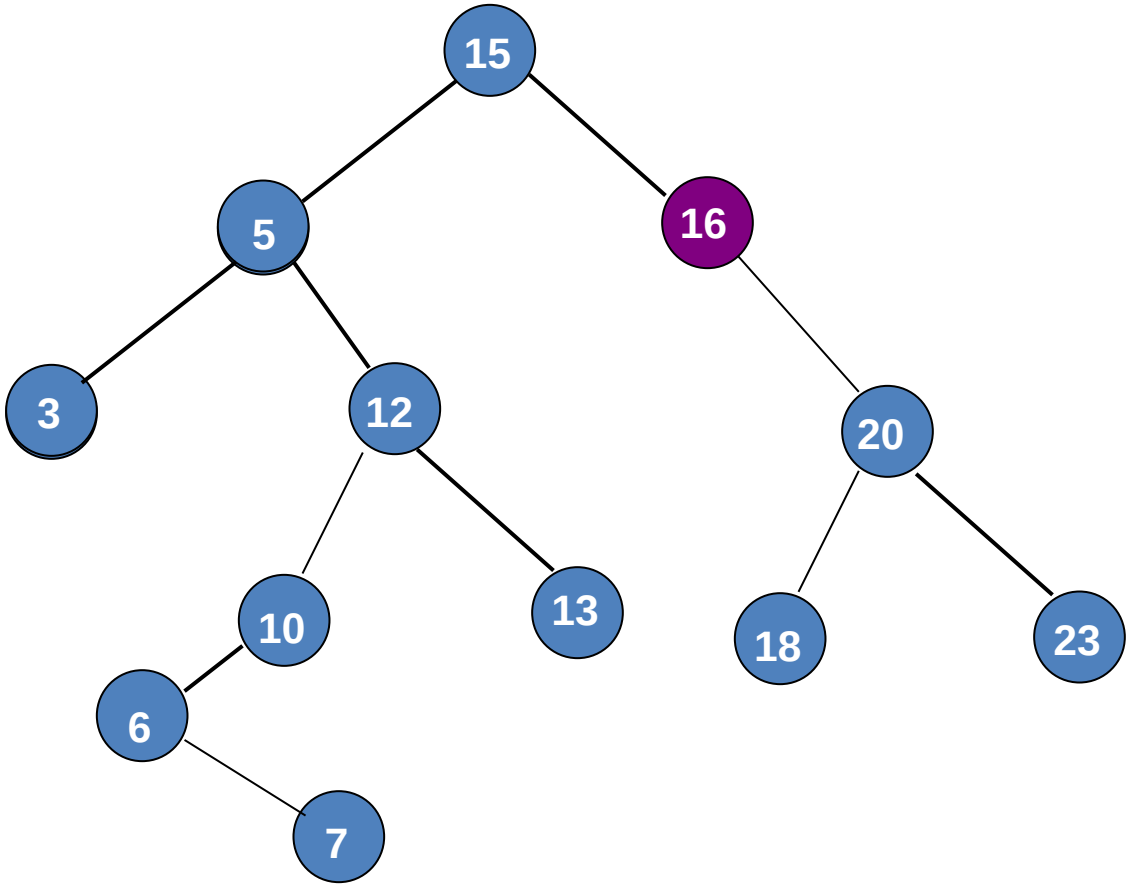
Durum 1 – Yaprak Düğümü Silme

- **Cevap:** Düğüm bulunur, kaldırılır ve bağlantısı güncellenir.



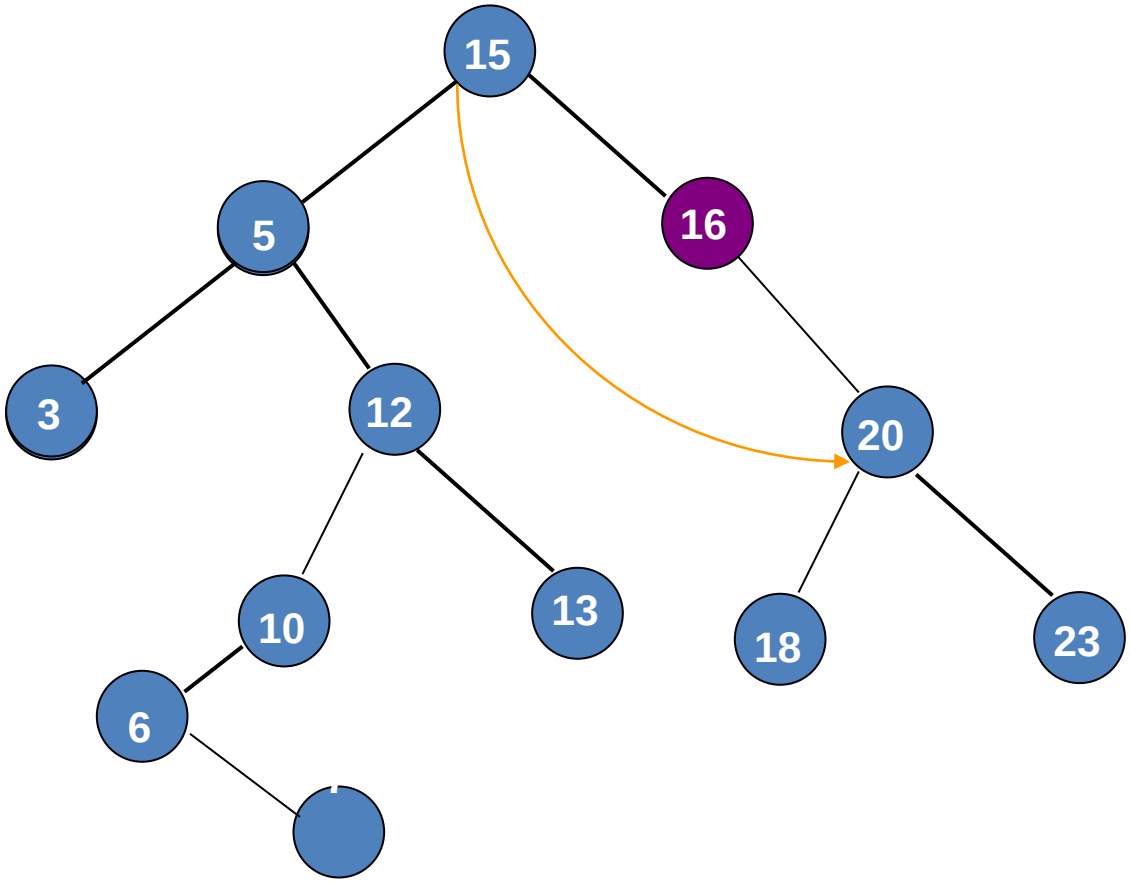
Durum 2 – 1 Çocuklu Düğüm Silme

- **Soru:** 16 değerine sahip düğüm nasıl sileriz?



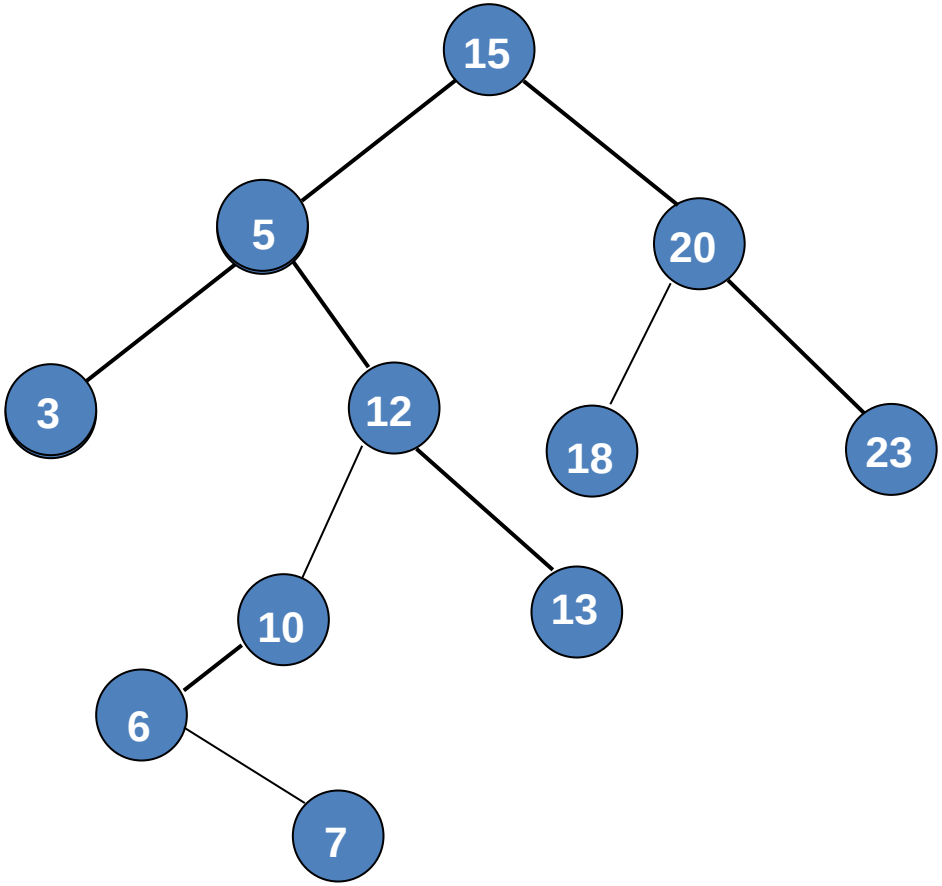
Durum 2 – 1 Çocuklu Düğüm Silme

- **Cevap:** Silinecek düğüm bulunur. Düğümün ailesinin solunda mı yoksa sağında mı olduğu saklanır. Silinecek düğümün çocuğu ile düğümün ailesi arasında bağ kurulur.

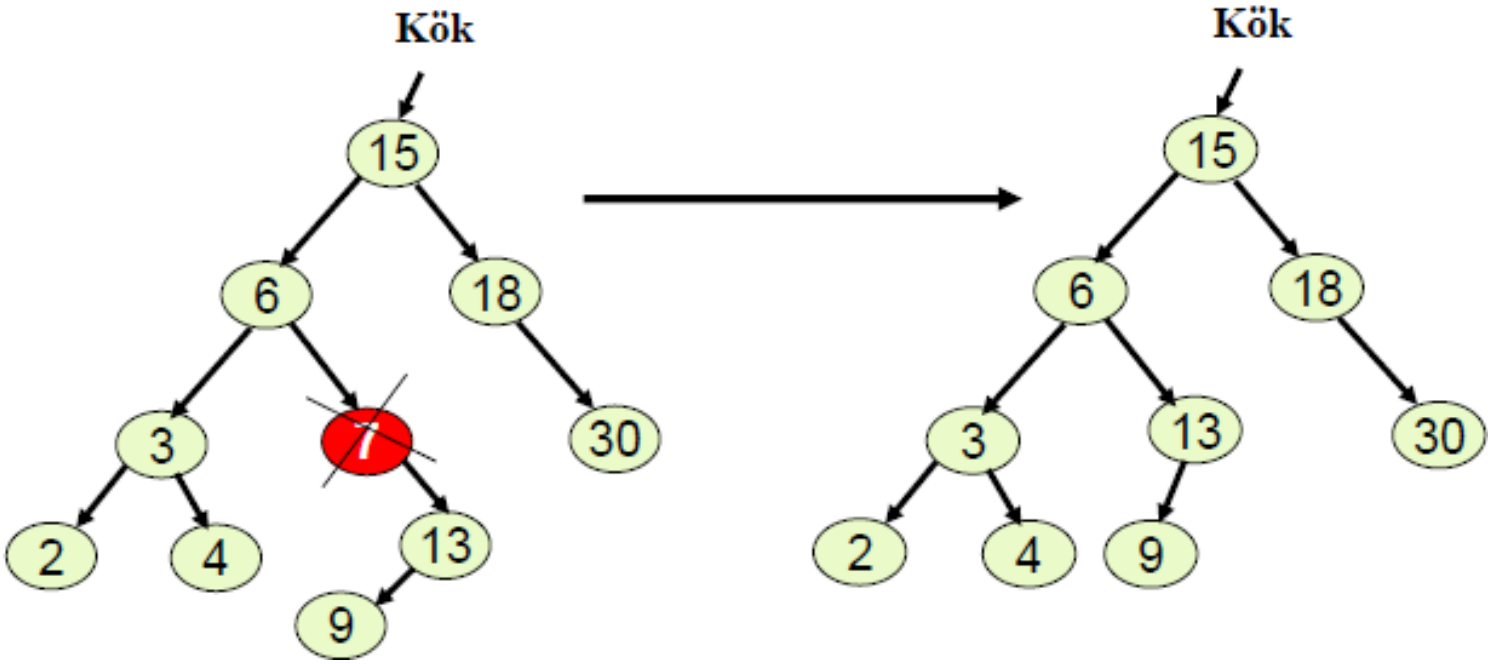


Durum 2 – 1 Çocuklu Düğüm Silme

- **Cevap:** Silinecek düğüm bulunur. Düğümün ailesinin solunda mı yoksa sağında mı olduğu saklanır. Silinecek düğümün çocuğu ile düğümün ailesi arasında bağ kurulur.

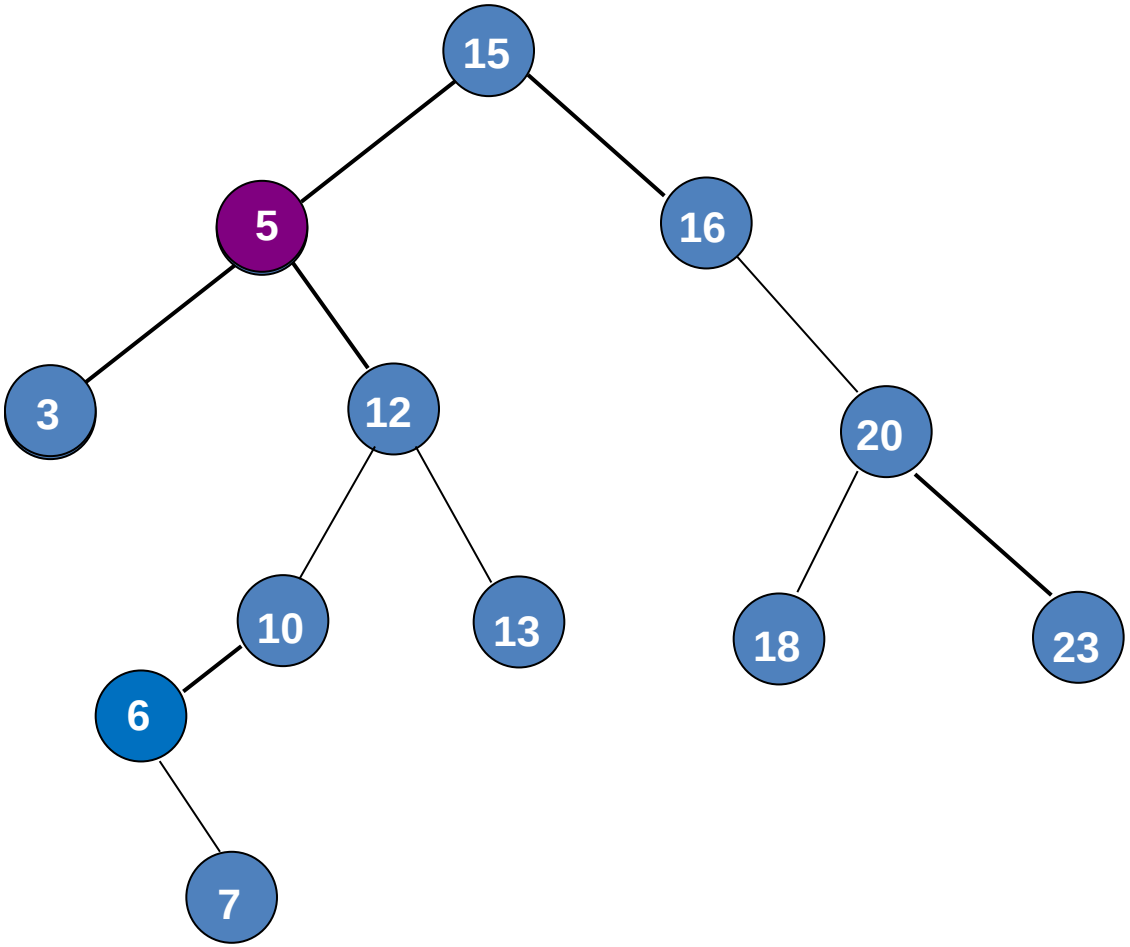


Durum 2 – 1 Çocuklu Düğüm Silme



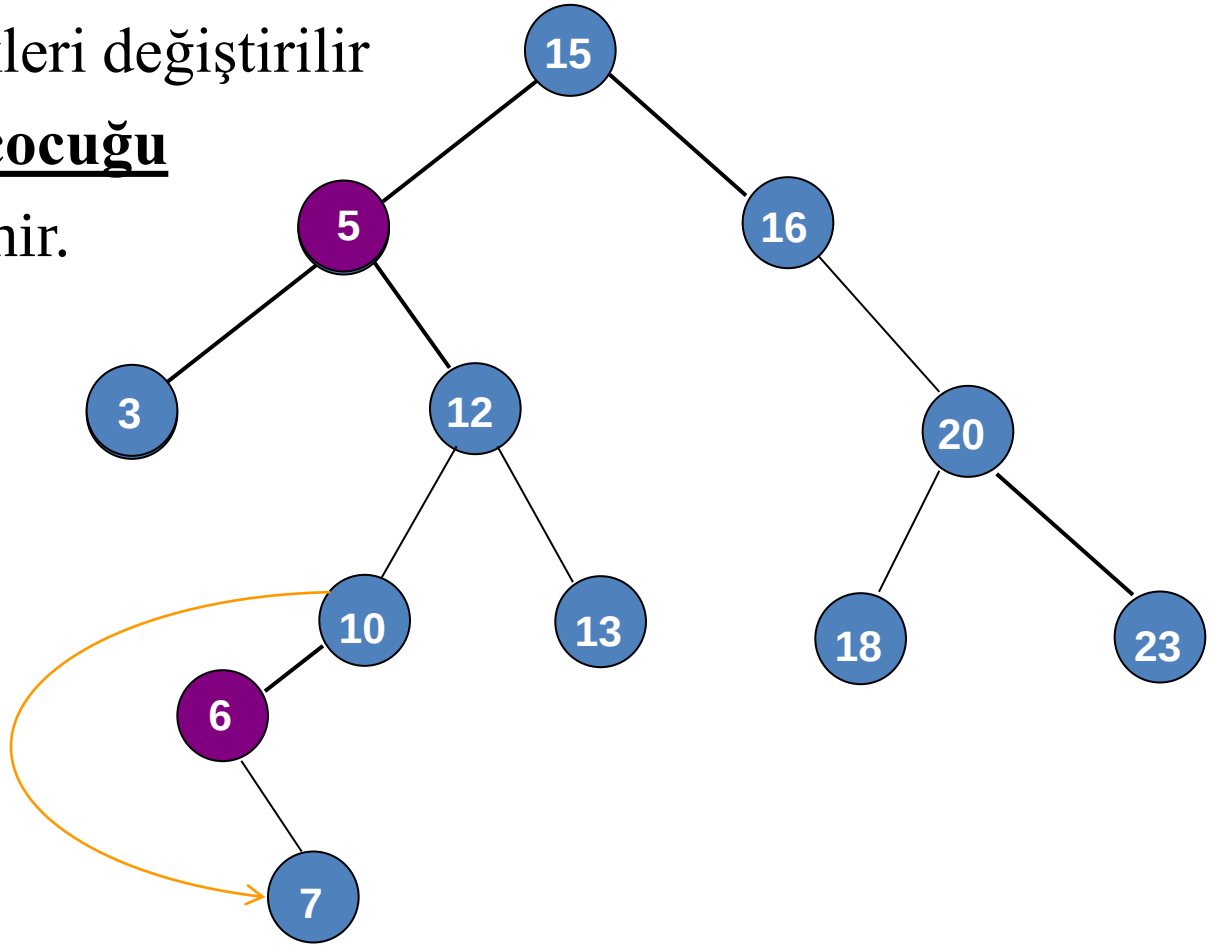
Durum 3 – 2 Çocuklu Düğüm Silme

- **Soru:** 5 değerine sahip düğümü nasıl sileriz?

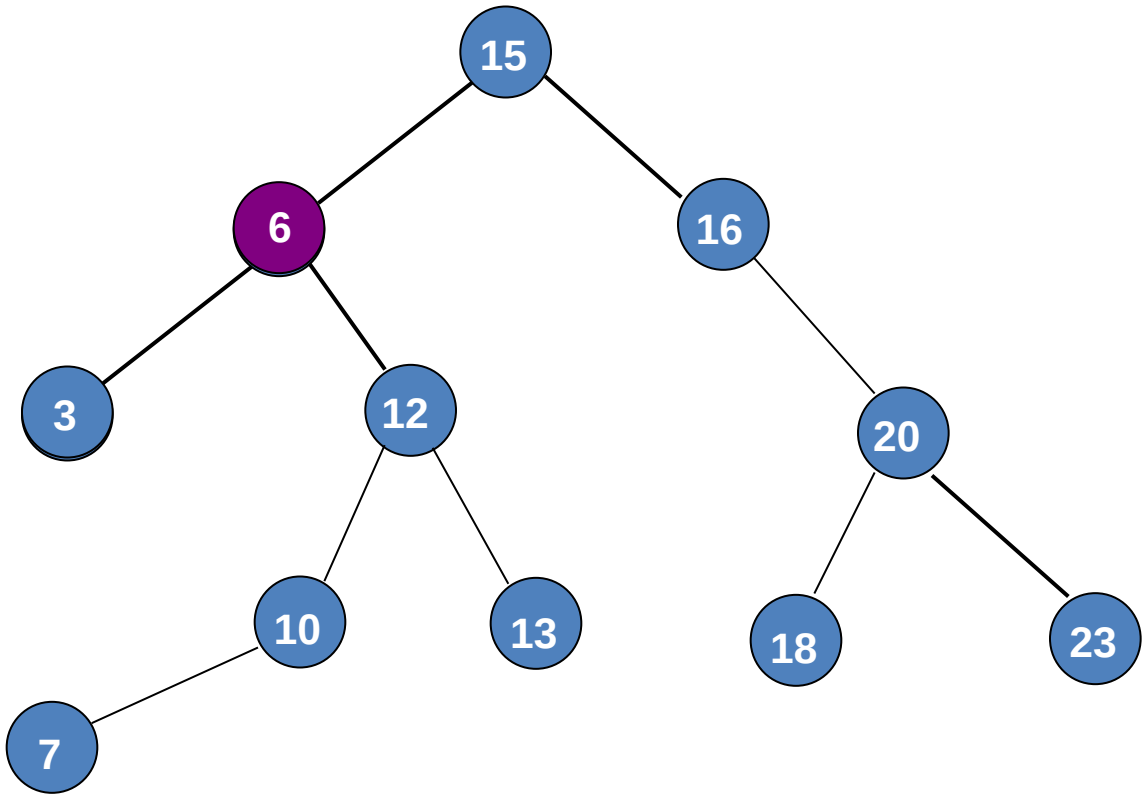


Durum 3 – 2 Çocuklu Düğüm Silme

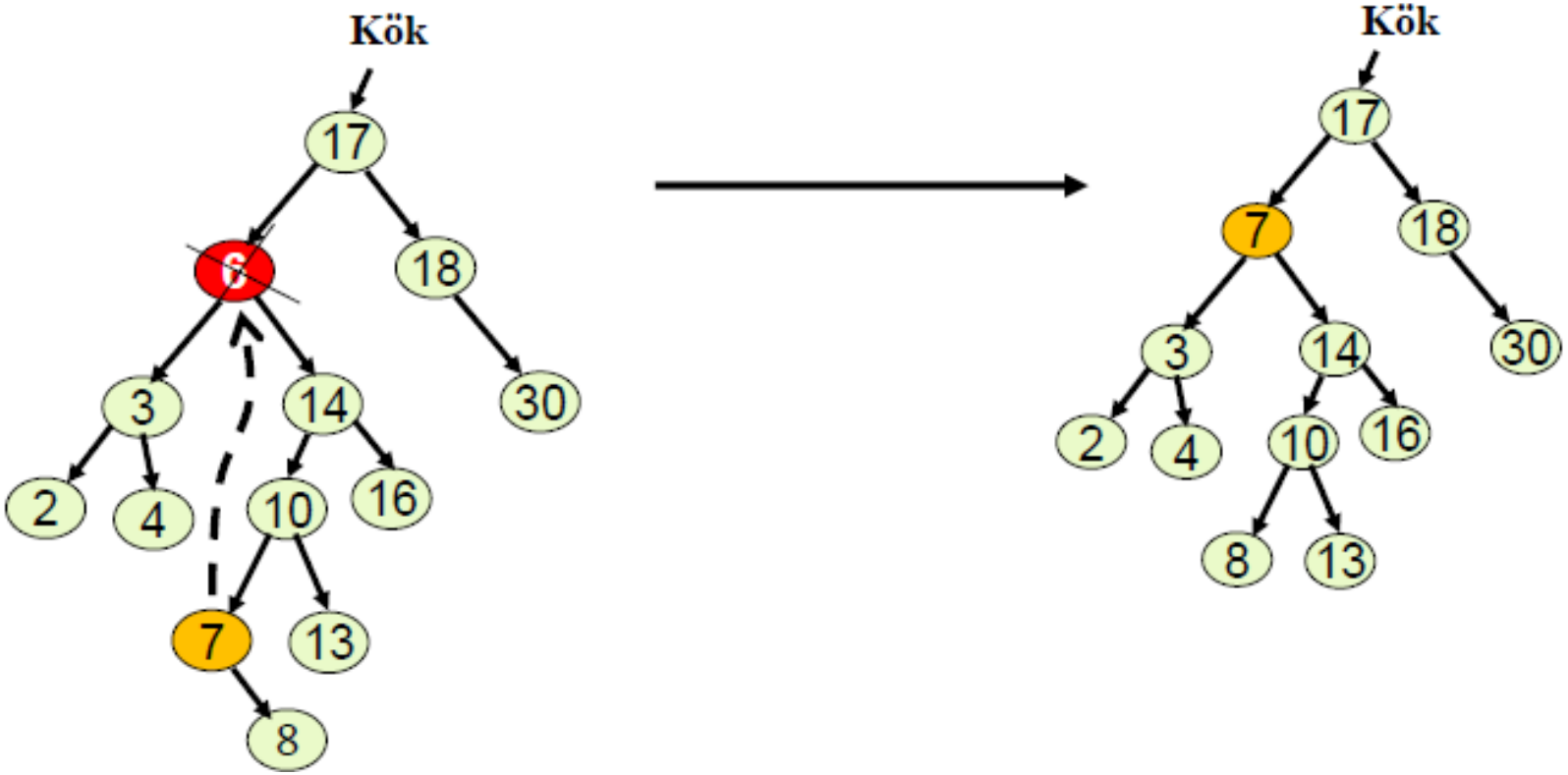
- **Cevap:** 5'in successor'u bulunur (sağ alt ağaçtaki en küçük eleman 6).
- 5 ve 6'nın içerikleri değiştirilir
- 6 elemanının **1 çocuğu varmış gibi** silinir.



Durum 3 – 2 Çocuklu Düğüm Silme



Durum 3 – 2 Çocuklu Düğüm Silme



Silme

1. ADIM – DÜĞÜMÜ BUL

```
İkiliAramaAgacDugumu current = kok;
İkiliAramaAgacDugumu parent = kok;
bool issol = true;
//DÜĞÜMÜ BUL
while ((int)current.veri != deger)
{
    parent = current;
    if (deger < (int)current.veri)
    {
        issol = true;
        current = current.sol;
    }
    else
    {
        issol = false;
        current = current.sag;
    }
    if (current == null)
        return false;
}
```

- **current:** Silmek için bulunan düğüm
- parent: current düğümün parent'ı

Silme (devam…)

DURUM 1 – EĞER YAPRAK DÜĞÜMSE

```
//DURUM 1: YAPRAK DÜĞÜM
if (current.sol == null && current.sag == null)
{
    if (current == kok)
        kok = null;
    else if (issol)
        parent.sol = null;
    else
        parent.sag = null;
}
```

Silme (devam…)

DURUM 2 – EĞER TEK ÇOCUKLU DÜĞÜMSE

```
//DURUM 2: TEK ÇOCUKLU DÜĞÜM
else if (current.sag == null)
{
    if (current == kok)
        kok = current.sol;
    else if (issol)
        parent.sol = current.sol;
    else
        parent.sag = current.sol;
}
else if (current.sol == null)
{
    if (current == kok)
        kok = current.sag;
    else if (issol)
        parent.sol = current.sag;
    else
        parent.sag = current.sag;
}
```

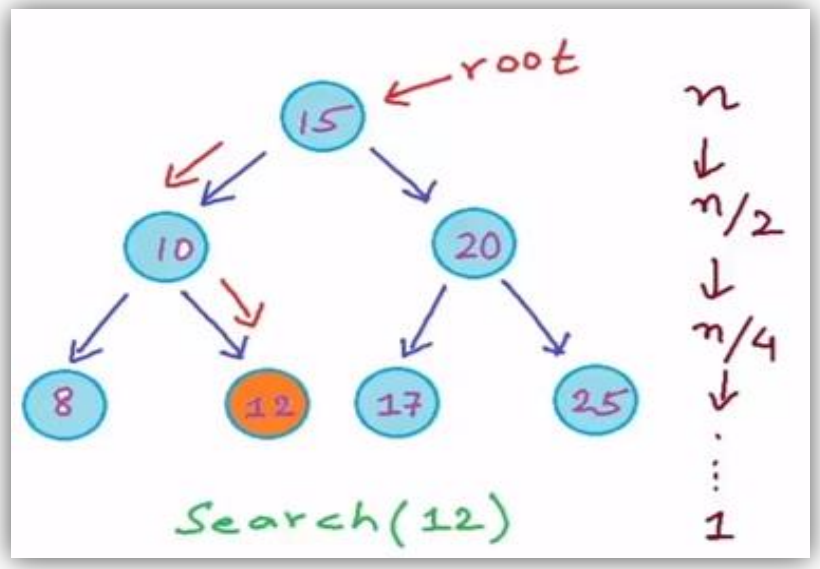
Silme (devam…)

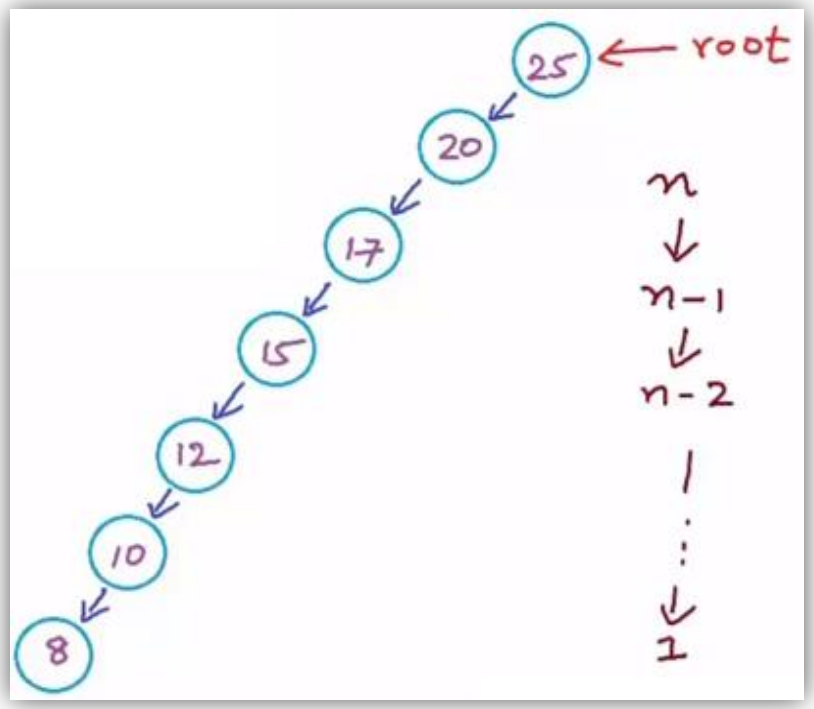
DURUM 3 – EĞER İKİ ÇOCUKLU DÜĞÜMSE

```
//DURUM 3: İKİ ÇOCUKLU DÜĞÜM
else
{
    İkiliAramaAgacDugumu successor = Successor(current);
    if (current == kok)
        kok = successor;
    else if (issol)
        parent.sol = successor;
    else
        parent.sag = successor;
    successor.sol = current.sol;
}
```

İkili Arama Ağacı Karmaşıklık Analizi

- **Soru:** İkili arama ağacının, arama işlemi için karmaşıklık analiz sonucu nedir?





İkili Arama Ağacı Karmaşıklık Analizi

Cevap:

- İkili arama ağaç işlemlerinin karmaşıklığı **O(h)'dır.**
- Fakat h ağacın derinliğine **bağlıdır**.
- Ağaç dengeli olmazsa O(n)'dir.
- Ağaç dengeli olursa O(logn)'dir.
- Nasıl dengeli yaparız?

1. AVL-ağaçları
2. Splay ağaçları
3. Red-Black ağaçları
4. B ağaçları, B+ agaçları

İkili Arama Ağacı Uygulamaları

- İkili arama ağacı *harita*, *sözlük* gibi birçok uygulamada kullanılır. Anahtar-değer çifti şeklinde kullanılacak sistemler için uygundur.
  - **Şehir Bilgi Sistemi:** Posta kodu verilip , şehir ismi döndürülür. (posta kodu/ Şehir ismi)
  - **Telefon Rehberi:** İsim verilir telefon numarası veya adres döndürülür (İsim, Adres/Telefon).
  - **Sözlük:** Kelime verilir anlamı döndürülür (Kelime, anlam).

İYİ ÇALIŞMALAR…

Yararlanılan Kaynaklar

- **Ders Kitabı:**
  - Data Structures through JAVA, V.V.Muniswamy
- **Yardımcı Okumalar:**
  - Data Structures and Algorithms in Java, Narashima Karumanchi
  - Data Structures, Algorithms and Applications in Java, Sartaj Sahni
  - Algorithms, Robert Sedgewick