YZM 2116- VERİ YAPILARI

DERS#6: AĞAÇ VERİ YAPISI

İÇERİK

Bu bölümde,

- Ağaç (Tree) Veri Yapısı Giriş
- Ağaç VY Temel Kavramlar
- İkili Ağaç (Binary Tree)
- İkili Ağaç Özellikleri
- İkili Ağaç Gerçekleştirim
- İkili Ağaç Üzerinde Gezinme (Traverse)
- Preorder, Inorder, Postorder
- İkili Ağaç Gerçekleştirimine Dair Sorular

konusuna değinilecektir.

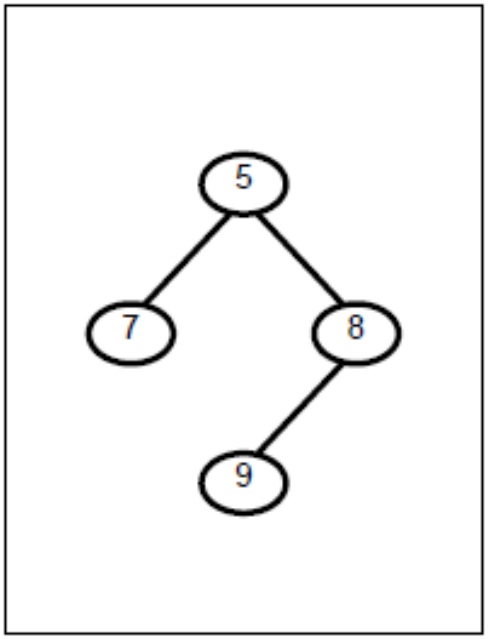
Ağaç VY Giriş

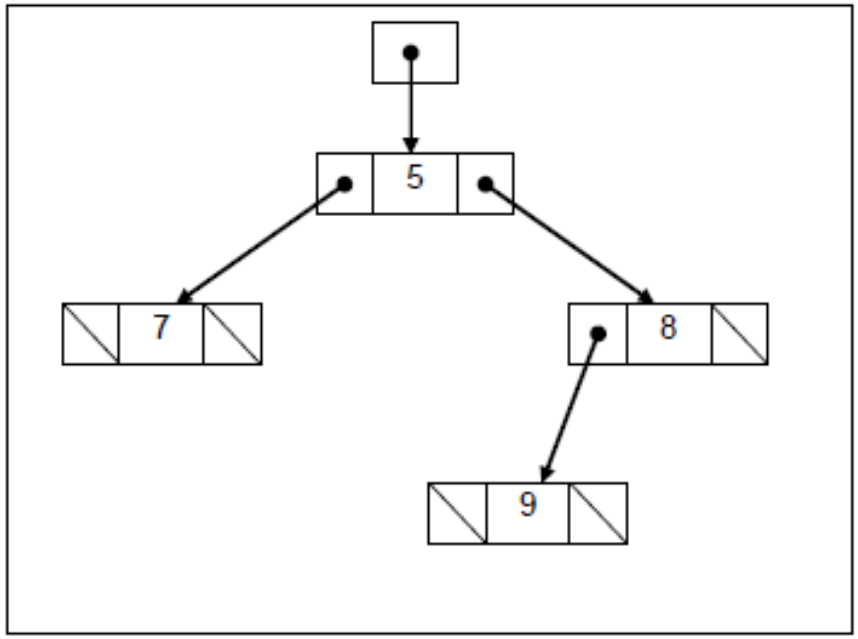
- Verilerin birbirine **sanki** bir ağaç yapısı oluşturuyormuş gibi sanal olarak bağlanmasıyla elde edilen ***hiyerarşik yapıya sahip*** veri yapısıdır.
- Aynı aile soyağacında olduğu gibi hiyerarşik bir yapısı vardır ve orada geçen birçok **kavram** buradaki *ağaç veri yapısında da* **tanımlıdır**.
- Örneğin çocuk, kardeş düğüm, aile, ata gibi birçok kavram ağaç veri yapısında da kullanılır.
- Her biri değişik bir uygulamaya doğal çözüm olan **ikili ağaç**, **kodlama ağacı**, **sözlük ağacı**, **kümeleme ağacı** gibi çeşitli ağaç şekilleri vardır; üstelik **uygulamaya yönelik özel** ağaç şekilleri de **çıkarılabilir**.

Ağaç VY Giriş (devam…)

- Bağlı listeler, yığıtlar ve kuyruklar **doğrusal (linear) veri yapılarıdır**.
- Ağaçlar ise **doğrusal olmayan** belirli niteliklere sahip *iki boyutlu* veri yapılarıdır.
- Ağaçlar **hiyerarşik ilişkileri** göstermek için kullanılır.
- Her ağaç düğümlerden (**node**) ve kenarlardan (**edge**) oluşur.
- Her düğüm bir *nesneyi* gösterir.
- Her **kenar** (bağlantı) **iki node arasındaki** ilişkiyi gösterir.
- Arama işlemi **bağlı listelere** göre **çok hızlı** yapılır.

Ağaç VY Giriş (devam…)



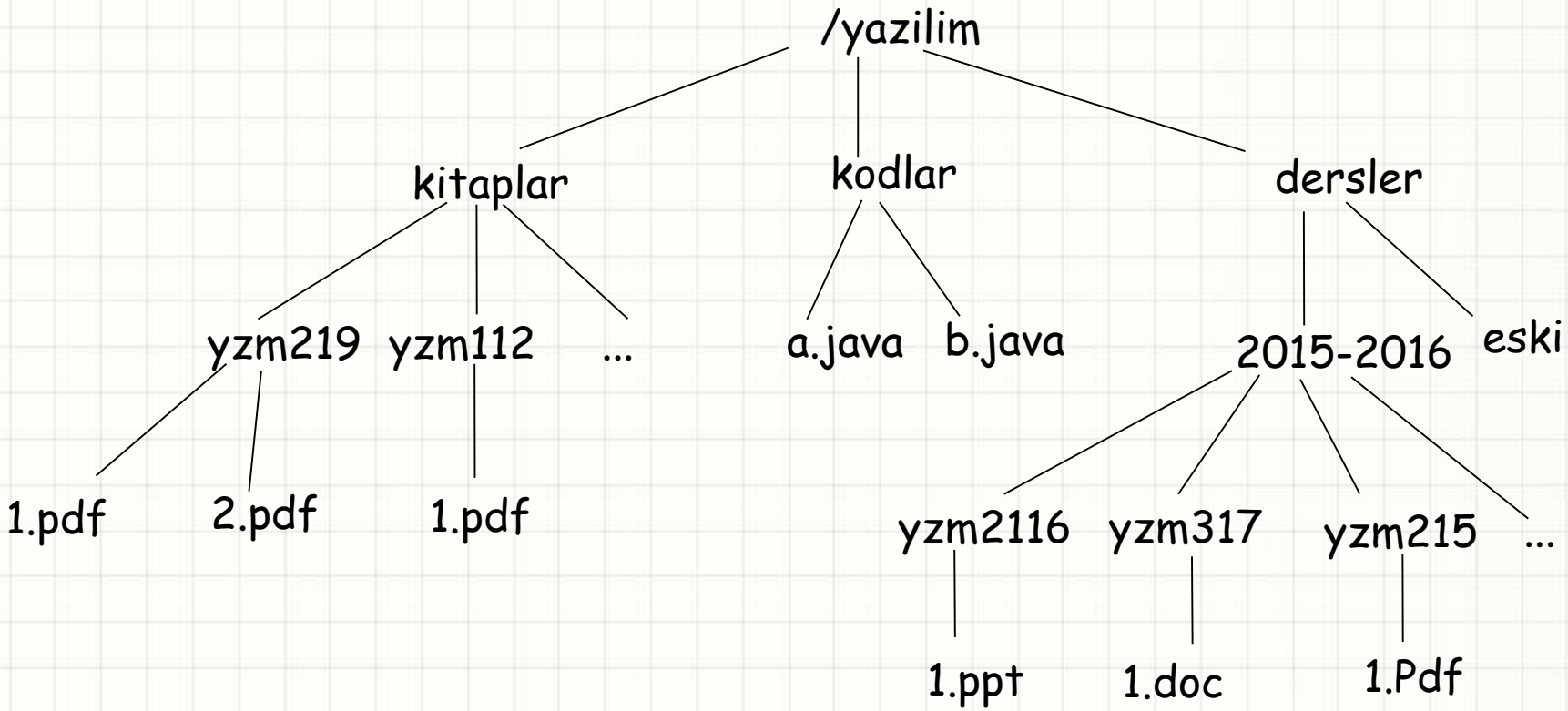


- Ağacın düğümlerindeki bilgiler **sayılardan oluşmuştur**. Her düğümdeki ***sol ve sağ bağlar*** yardımı ile **diğer düğümlere ulaşılır**. Sol ve sağ bağlar **NULL** olabilir.
- Düğüm yapıları değişik türlerde bilgiler içeren veya birden fazla bilgi içeren ***ağaçlar da*** olabilir.

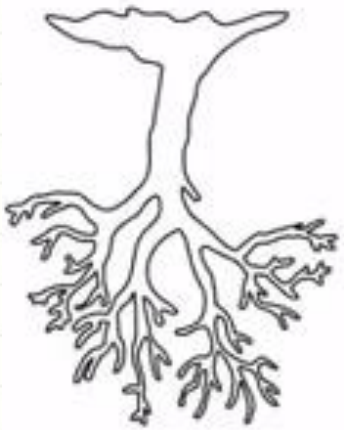
Ağaç VY Kullanılan Yazılım Uygulamaları

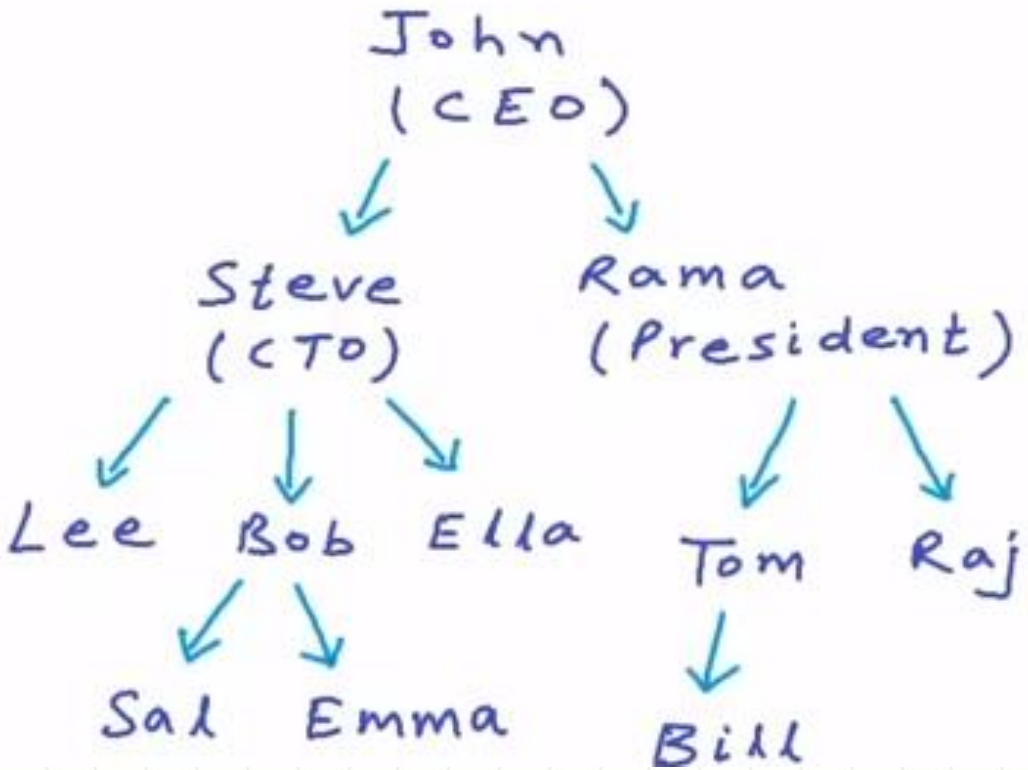
- Ağaç veri yapısı / modeli aşağıdaki ***yazılım uygulamalarında*** kullanılırlar:
  - İşletim sisteminin **dosya sistemini** modellemekte
  - **Oyunlar** için **farklı hamleleri** ele almakta
  - **Ağ routing** (yönlendirme) algoritmalarında
  - Trie adı verilen VY ile **sözlük oluşturmakta** ve dinamik yazım kontrolü gibi alanlarda
  - Huffman sıkıştırma kodlamasında ve
  - Derleyicilerde matematiksel ifadeleri modellemede kullanılırlar.

Örnek Ağaç Yapısı – Dosya Sistemi

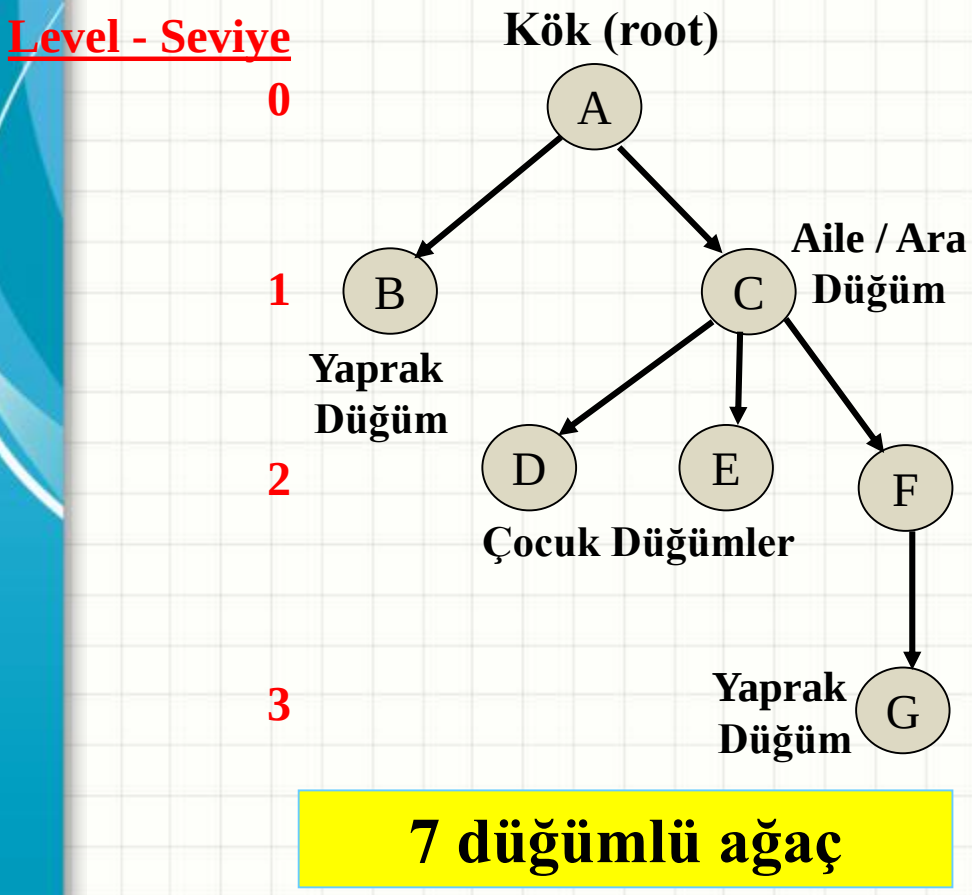


Örnek Ağaç Yapısı – Şirket Organizasyonu





Ağaç VY Temel Kavramlar



- A düğümü **kök** olmak üzere **7 düğümden (n)** oluşmaktadır.
- Toplam ***6 kenar*** **(n-1)** vardır.
- *Sol alt ağaç*, **B kökü** ile başlamakta ve *sağ alt ağaç* da **C kökü** ile başlamaktadır.
- A'dan solda B'ye giden ve sağda C'ye giden ***iki dal (branch)*** çıkmaktadır.

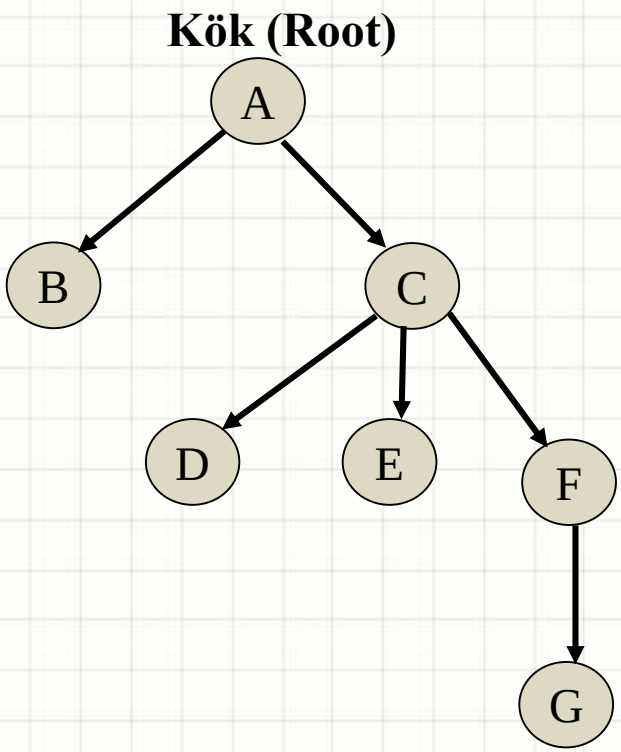
Ağaç VY Temel Kavramlar (devam…)

- **Düğüm (Node):** Ağacın her elemanına verilen isim.
- **Kök (Root):** Ağacın başlangıç düğümüdür.
- **Çocuk (Child):** Bir düğüme doğrudan bağlı olan düğümlere o çocukları denilir.
- **Kardeş Düğüm (Sibling):** Aynı düğüme bağlı düğümlere kardeş düğüm veya kısaca kardeş denir.
- **Aile (Parent):** Düğümlerin doğrudan bağlı oldukları düğüm aile olarak adlandırılır; diğer bir deyişle aile, kardeşlerin bağlı olduğu düğümdür.
- **Ata (Ancestor):** Aile düğümünün daha üstünde kalan düğümlere denir.
- **Torun (Dedscendant):** Bir düğümün çocuğuna bağlı olan düğümlere denir.

Ağaç VY Temel Kavramlar (devam…)

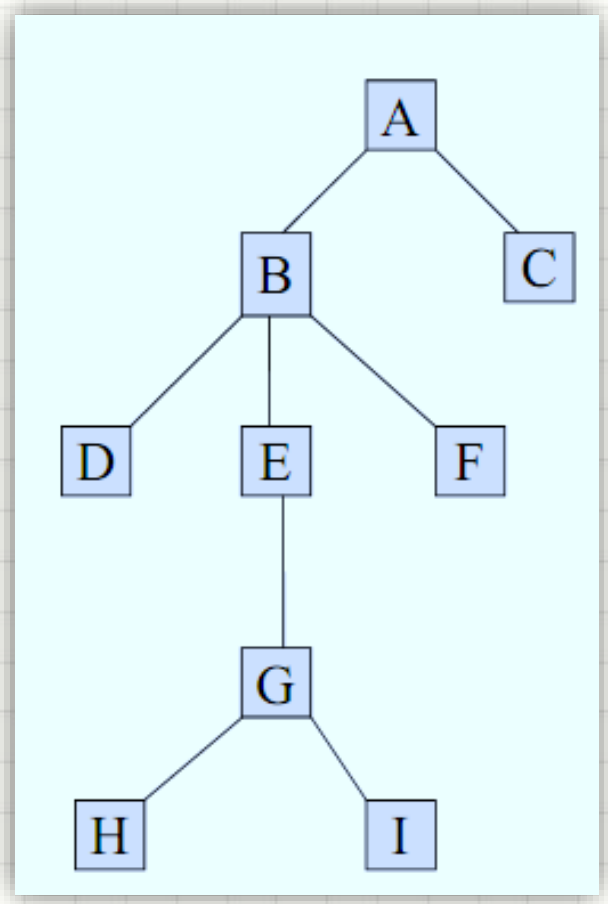
- **Yaprak (Leaf):** Ağacın en altında bulunan ve çocukları olmayan düğümlerdir.
- **Derece (Degree):** Bir düğümden alt hiyerarşiye yapılan bağlantıların sayısıdır; yani **çocuk** veya alt ağaç sayısıdır.
- **Seviye (Level):** Hiyerarşik sıradır (rank). Kök düğüm seviye = 0.
- **Derinlik (Depth):** Bir düğümün köke olan uzaklığı derinliktir. Kök düğüm derinlik = 0.
- **Yükseklik (Height):** Bir düğümün kendi silsilesinden en uzak yaprak düğüme olan uzaklığıdır. Yaprak düğümlerin yükseliği = 0. (Kök yüksekliği = Ağaç Yüksekliği)
- **Yol (Path):** Bir düğümün aşağıya doğru (çocukları üzerinden) bir başka düğüme gidebilmek için üzerinden geçilmesi gereken düğümlerin listesidir.

Ağaç VY Temel Kavramlar (devam…)



Tanım	Kök	B	D
Çocuk/Derece	2	0	0
Kardeş	1	2	3
Seviye	0	1	2
Aile	yok	Kök	C
Ata	yok	yok	Kök
Yol	A	A, B	A,C,D
Derinlik	0	1	2
Yükseklik	3	0	0

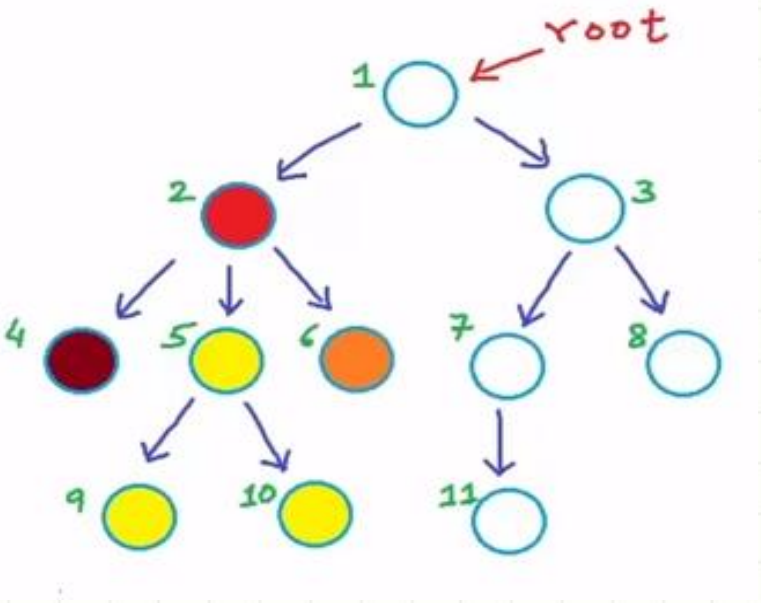
Ağaç VY Temel Kavramlar (devam…)



Düğüm sayısı: 9
Yükseklik: 4
Kök düğüm: A
Yapraklar: C, D, F, H, I
Seviye sayısı: 4
H'nin ataları: E, B, A
B'nin torunları: G, H, I
E'nin kardeşleri: D, F

Ağaçlara Özyinelemeli Yaklaşım

- Rekürsif mantıkla bir ağaç, bir kök (root) ve alt-ağaçlar şeklinde tanımlanır.



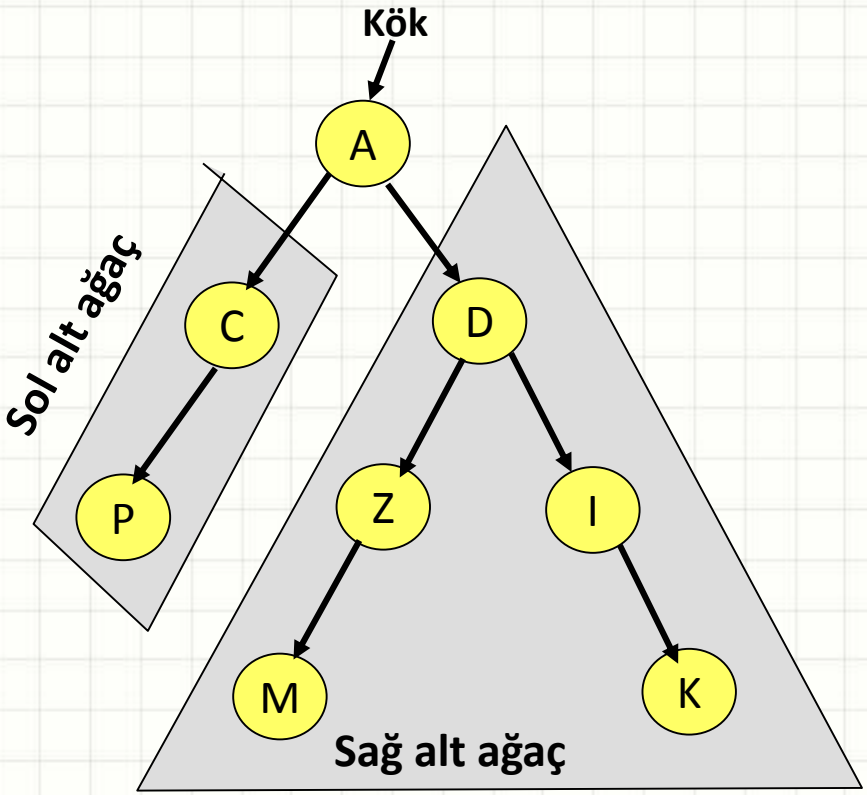
- 1 düğümü → 2 ve 3 alt-ağaçlarından oluşur
- 2 düğümü → 4,6 ve 5 (5-9-10) olmak üzere üç alt-ağaçtan oluşur.

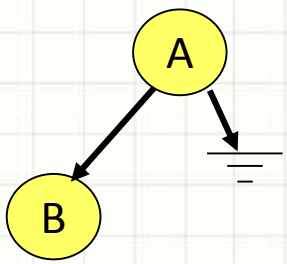
İkili Ağaç (Binary Tree)

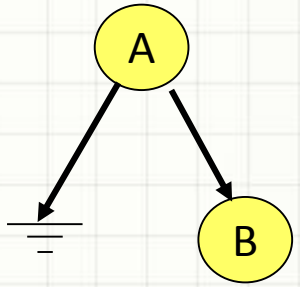
İkili Ağaç (Binary Tree)

- Her düğümün **en fazla** iki çocuk düğüme sahip olduğu ağaç yapısına BT denir.
- BT, bilgisayar bilimlerinde **en çok kullanılan** ağaç veri yapılarından olup, **sıralı olması durumunda** *arama*, *ekleme* ve *silme* işlemlerini **çabuklaştırırlar**.
- Bir BT
  - **Boş** tek bir düğümden oluşabileceği gibi,
  - **Sol alt ağaç** ve **sağ alt ağaç** olmak üzere ***köke ait iki adet*** ikili ağaçtan da oluşabilir.
- BT için gerek şart ilgili BT düğümünün **en fazla iki çocuğa (alt ağaca)** sahip olabileceğidir.

İkili Ağaç (Binary Tree) (devam…)



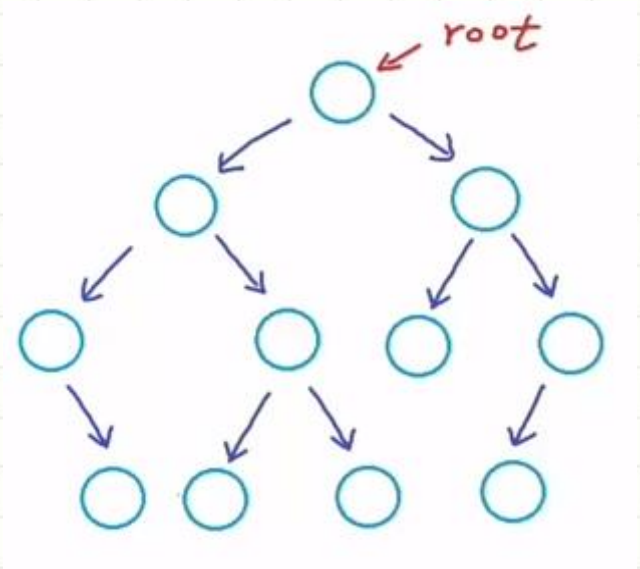




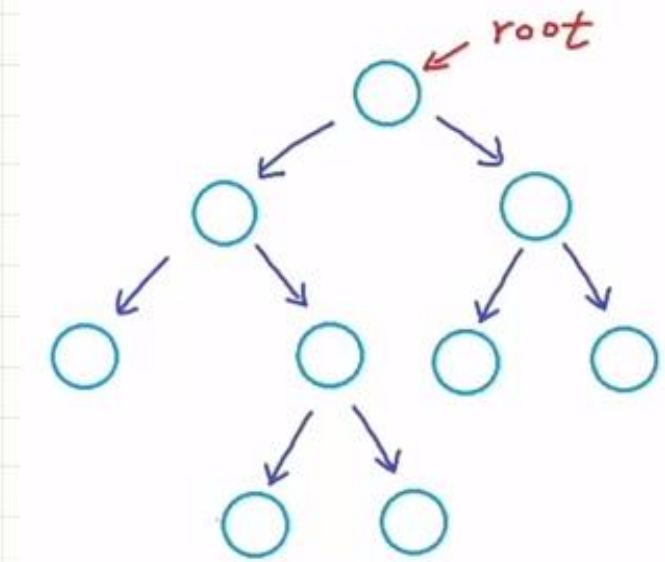
İki farklı ikili ağaç

Katı İkili Ağaç (Strict Binary Tree)

- Yaprak düğümler haricindeki tüm düğümler ***sıfır veya iki çocuğa sahip*** ise **katı ikili ağaç** olarak adlandırılır.



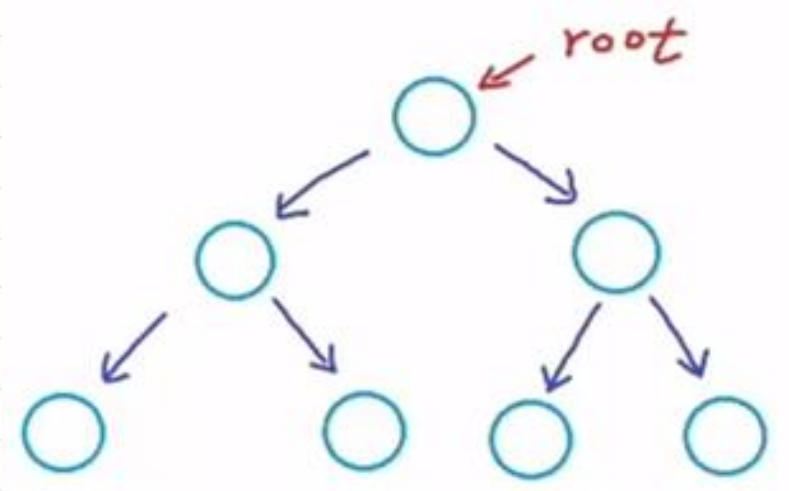
Katı Olmayan İkili Ağaç



Katı İkili Ağaç

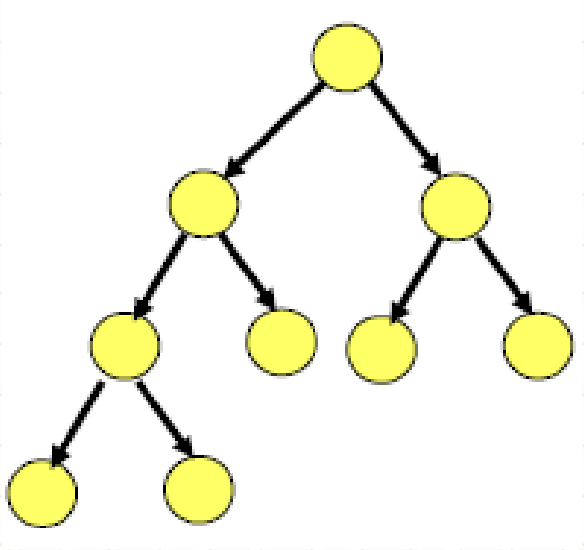
Tam İkili Ağaç (Full Binary Tree)

- Her bir düğümün (i)**net olarak iki çocuk düğüme** sahip olduğu ve (ii)**yaprak düğümlerin aynı seviyede** olduğu iki ağaçtır.
- Her düğüm eşit şekilde sağ ve sol alt-ağaçlara sahiptir.



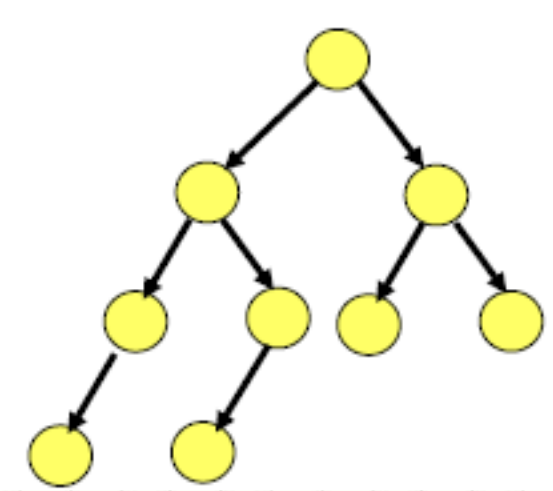
Eksiksiz İkili Ağaç (Complete Binary Tree)

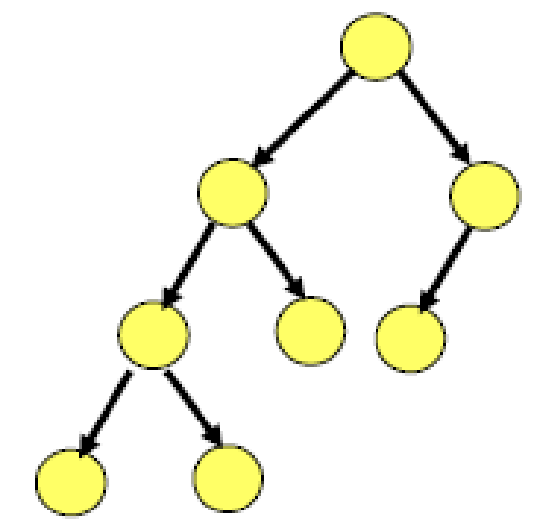
- **Son seviye dışındaki** tüm seviyelerin tam (full) olduğu ikili ağaç türüdür.
- Düğümleri **sol taraftan** (düğüme göre) doldurulur.
- ***Yeni bir derinliğe soldan sağa doğru ekleme başlanır.***



Eksiksiz İkili Ağaç (Complete Binary Tree)

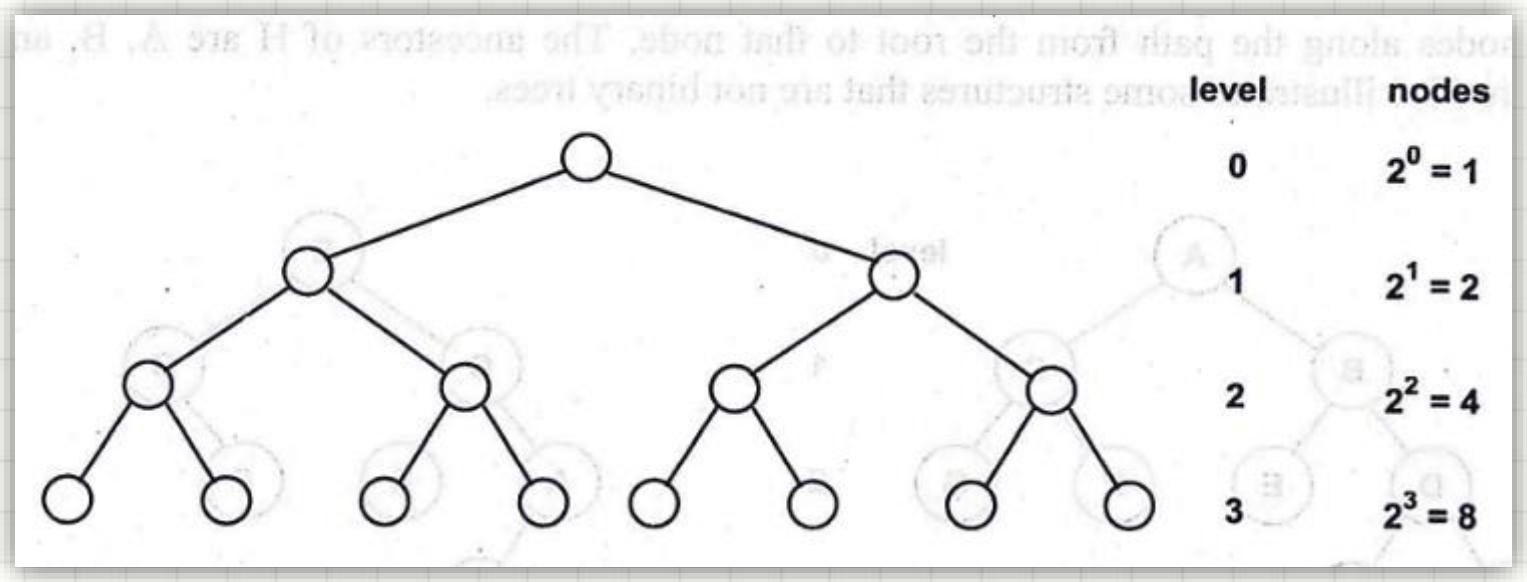
- Aşağıdaki örnekler, eksiksiz ikili ağaç kriterlerine uyuyor mu?





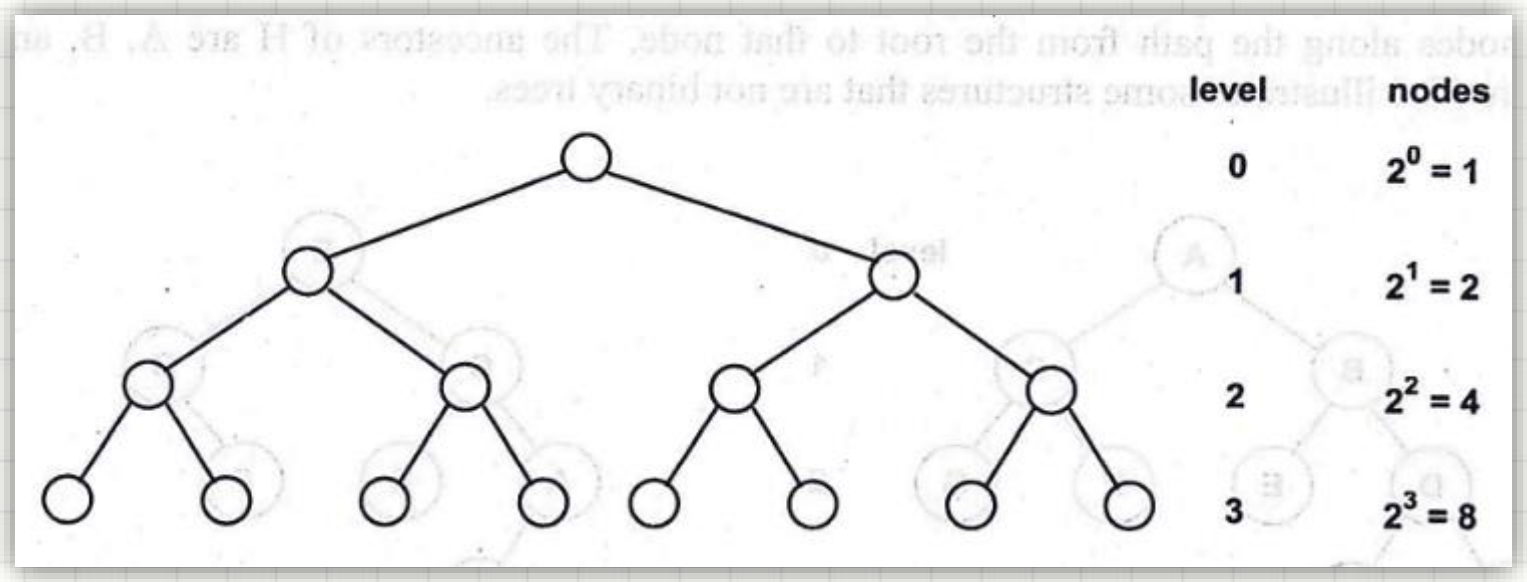
İkili Ağaç Özellikleri – Özellik 1

- Her k seviyesinde, ***maksimum düğüm sayısı*** 2^k dır.
- Kök düğümde **k=0** olduğu için **düğüm sayısı 1** olurken,
- k=2 için maksimum düğüm sayısı 4 olur.



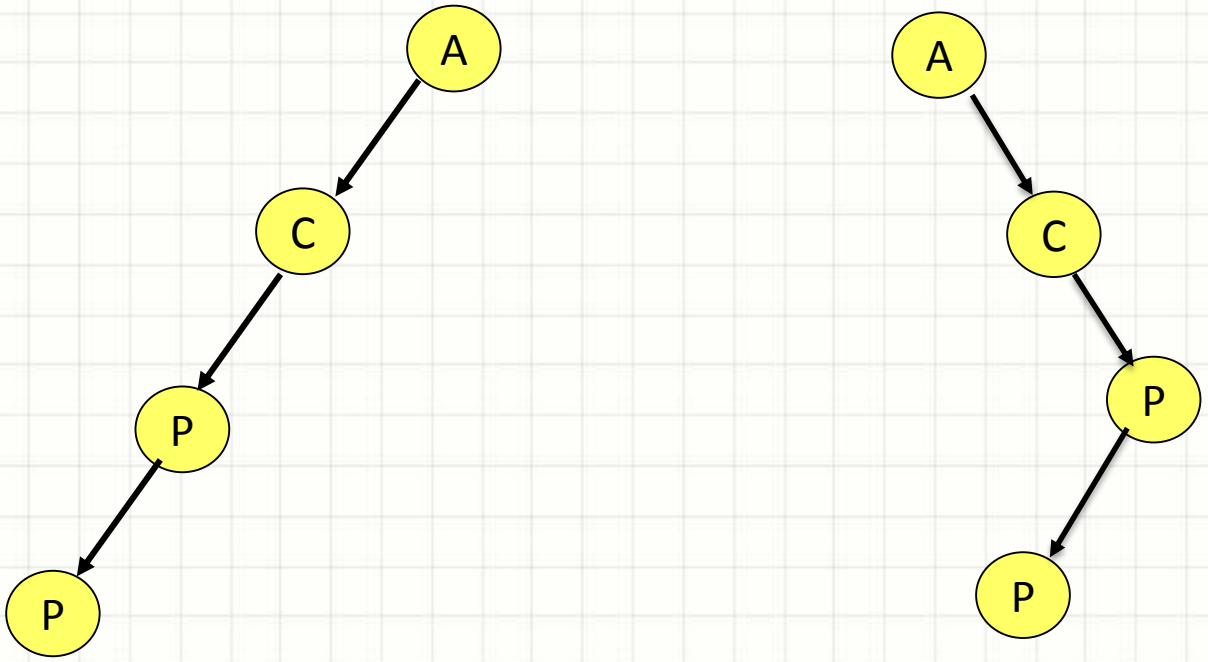
İkili Ağaç Özellikleri – Özellik 2

- **h yüksekliğindeki** ikili ağacın ***maksimum düğüm sayısı*** = $2^{h+1} - 1$ olarak hesaplanır.
- Aşağıdaki ağaç için **h = 3** olduğundan **maksimum düğüm sayısı** 16 – 1 = **15** bulunur.



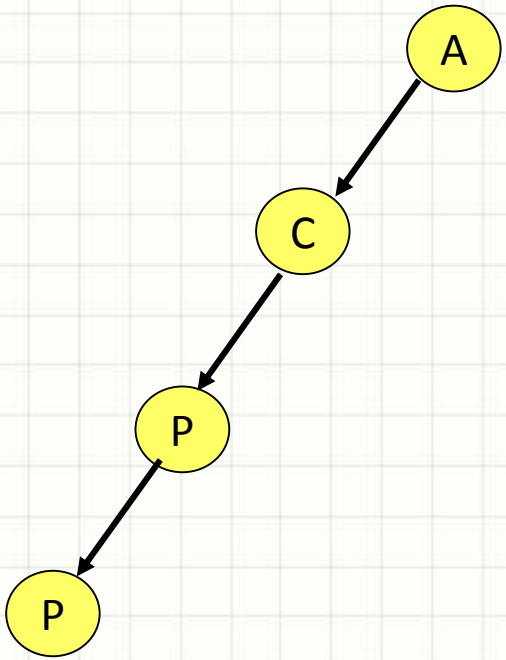
İkili Ağaç Özellikleri – Özellik 3

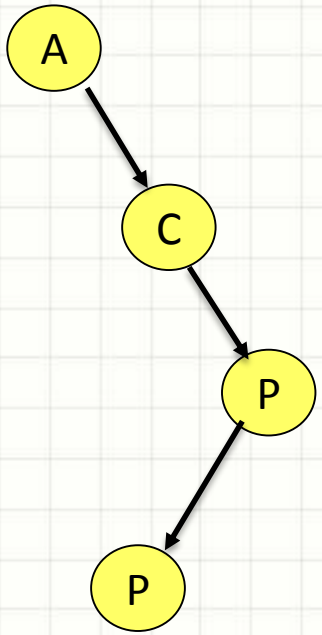
- **h yüksekliğindeki** ikili ağacın
  - ***minimum düğüm sayısı*** = ***h*** + ***1*** olarak hesaplanır.
- **Kök** düğümün **h = 0** da olduğuna dikkat ediniz.



İkili Ağaç Özellikleri – Özellik 4

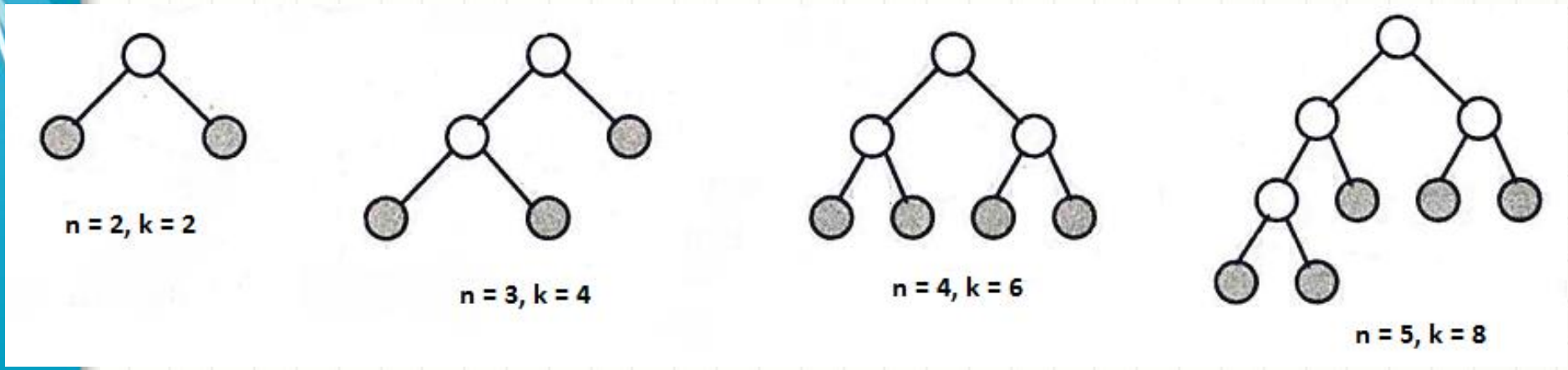
- **En az iki düğüme sahip, n elemanlı** bir ikili ağaçta, **kenar sayısı = (n - 1)** olarak hesaplanır.
- Aşağıdaki örneklerde 4'er düğüm ve 3'er kenar mevcuttur.





İkili Ağaç Özellikleri – Özellik 5

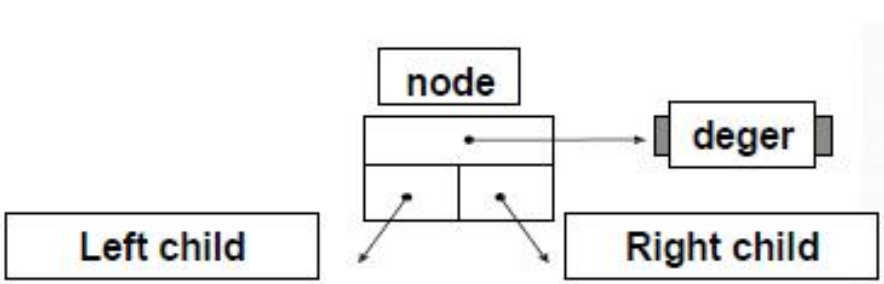
- n yaprak düğüme sahip, **eksiksiz bir ikili ağaçta** (complete), **kenar sayısı (k) = 2(n-1)**'dir.

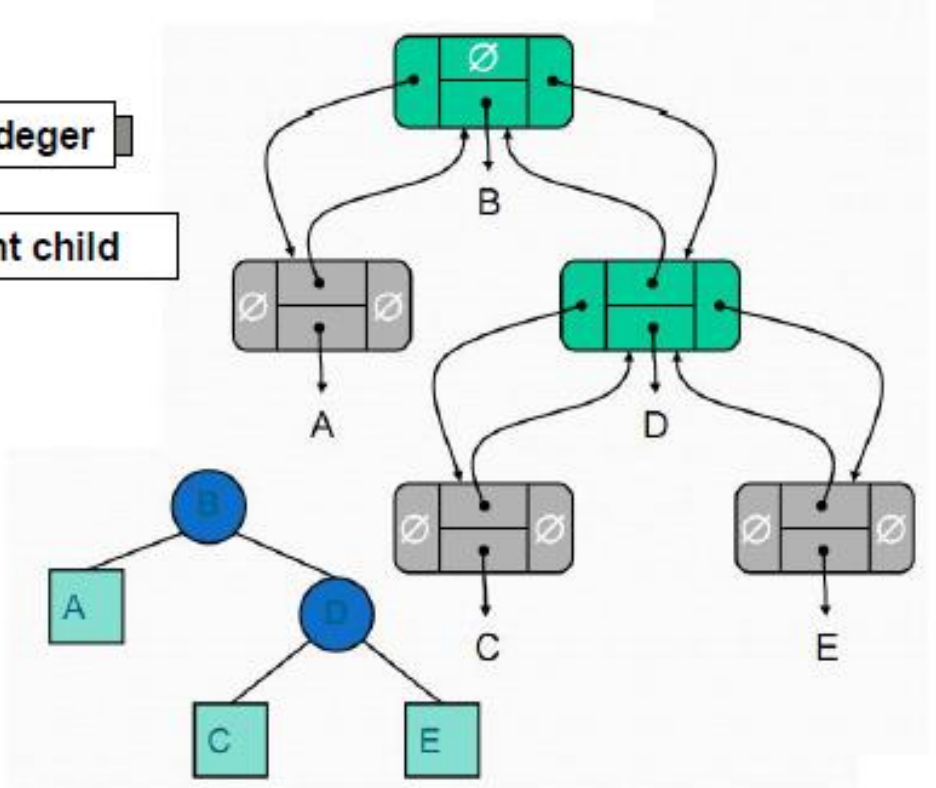


İkili Ağaç Gerçekleştirimi

- İki türlü gerçekleştirimi mümkündür:
  - Dizi kullanarak
  - Bağlı liste kullanarak

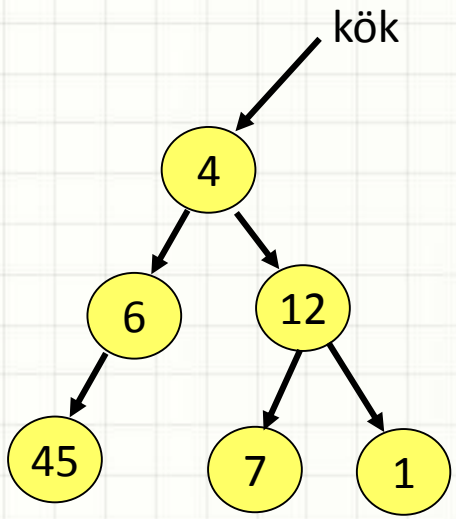
Bağlı Liste İkili Ağaç Gerçekleştirimi





Bağlı Liste İkili Ağaç Gerçekleştirimi (devam…)

```
public class İkiliAgacDugumu
{
    public object veri;
    public İkiliAgacDugumu sol;
    public İkiliAgacDugumu sag;
    16 references
    public İkiliAgacDugumu(object veri)
    {
        this.veri = veri;
        sol = null;
        sag = null;
    }
}
```

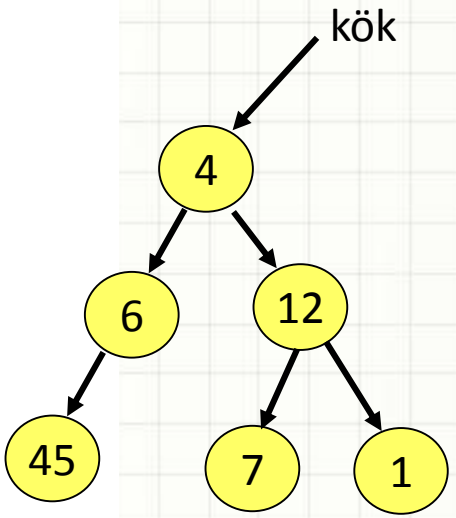


Bağlı Liste İkili Ağaç Gerçekleştirimi (devam…)

```
//kök düğümü oluştur
İkiliAgacDugumu kok = new İkiliAgacDugumu(4);

//***köke sağ çocuk ekle
kok.sag = new İkiliAgacDugumu(12);
//kökün sağ çocuğuna, sağ çocuk ekle(1)
kok.sag.sag = new İkiliAgacDugumu(1);
//kökün sağ çocuğuna, sol çocuk ekle(7)
kok.sag.sol = new İkiliAgacDugumu(7);

//***köke sol çocuk ekle (6),
kok.sol = new İkiliAgacDugumu(6);
//sonra eklediğin sol çocuğa bir sol çocuk ekle (45)
kok.sol.sol = new İkiliAgacDugumu(45);
```



- **Soru:** Kökten itibaren düğümleri elimizde olan ikili ağaç üzerinde nasıl dolaşırız?

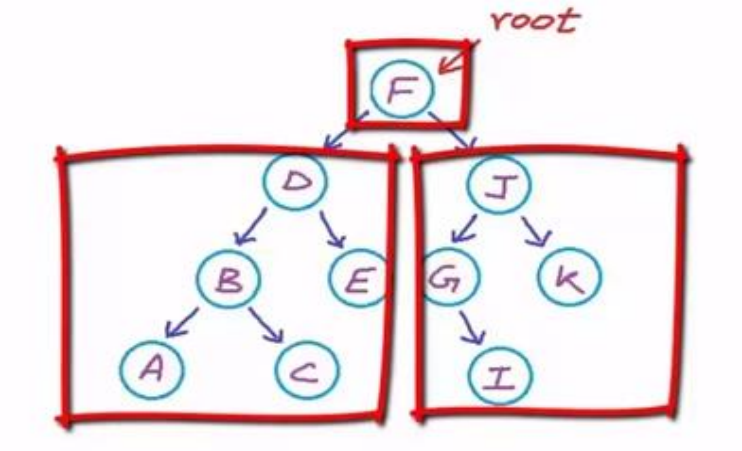
İkili Ağaç Üzerinde Gezinme (Traverse)

- Bir ağacı bir düğüme **sadece bir defa uğrayacak şekilde** gezmeye **gezinme/geçiş** denir.
- Gezinme, ağacın sakladığı bilgi türüne göre bir bilgiye ulaşma, listeleme ve başka amaçlarla gerçekleştirilir.
- **Doğrusal veri yapılarında** baştan sona doğru dolaşmak **kolaydır**.
- Ağaçlar ise düğümleri ***doğrusal olmayan*** veri yapılarıdır. Bu nedenle farklı algoritmalar uygulanır.
- ***Örneğin;*** düğümlerin matematiksel operatörler ve oparandlar olduğu bir ikili ağacın gezinme işlemi, infix-postfix gibi bir notasyonu elde etmemizi sağlar.

İkili Ağaç Üzerinde Gezinme (devam…)

- İkili ağacın üzerinde gezinilmesi sırasında bağımsız olarak 3 grup ikili ağaç parçasının
  1. Kök
  2. Sol alt ağaç
  3. Sağ alt ağaç

değişik sıralarda gezilmesiyle ile gerçekleşir.



İkili Ağaç Üzerinde Gezinme (devam…)

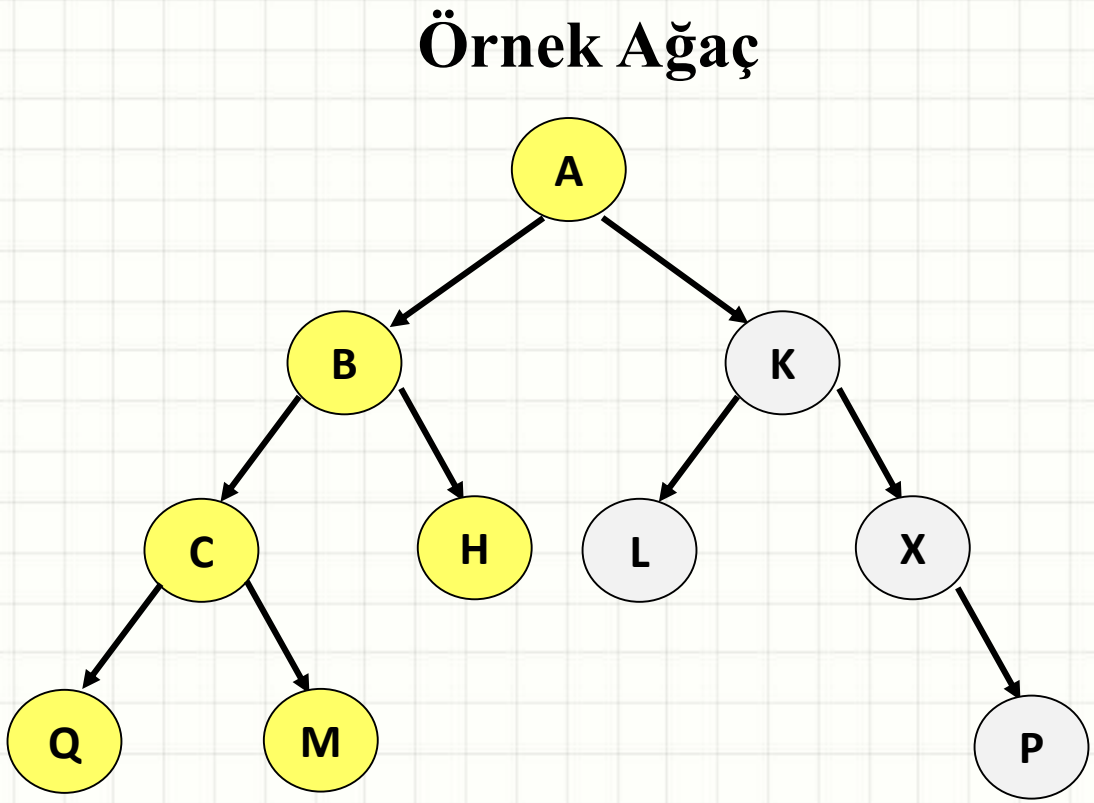
- İkili ağaç üzerinde dolaşmak için **3 temel yol vardır**. Bunlar:
  - **Önce-kök/ziyaret (Preorder - NLR):**
    - ✓ **Kök/ziyaret, Sol, Sağ**

    Önce kök, sonra sol alt ağaç ve ardından sağ alt ağaç
  - **Ortada-kök/ziyaret (Inorder - LNR):**
    - ✓ **Sol, Kök/ziyaret, Sağ**

    Önce sol alt ağaç, kök ve sağ alt ağaç
  - **Sonra-kök/ziyaret (Postorder - LRN):**
    - ✓ **Sol, Sağ, Kök/ziyaret**

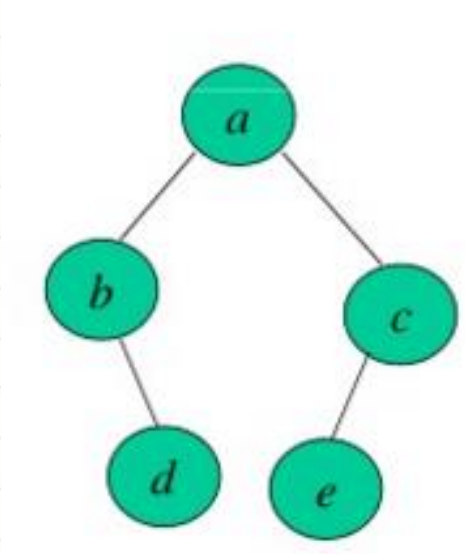
    Önce sol alt ağaç, sağ alt ağaç ve kök.

İkili Ağaç Üzerinde Gezinme (devam…)



Preorder ile Gezinme

1. Düğümü ziyaret et
2. Sol alt ağaçta gezin
3. Sağ alt ağaçta gezin

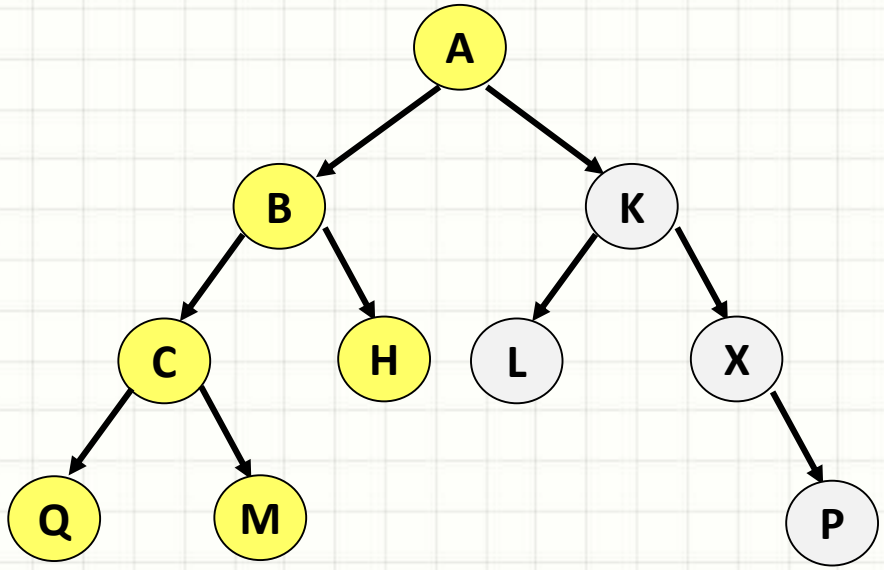


Preorder Gezinme:
a, b, d, c, e

Preorder ile Gezinme (devam…)

```
private void Ziyaret(İkiliAgacDugumu dugum)
{
    dugumler += dugum.veri + " ";
}
1 reference
public void PreOrder()
{
    PreOrderInt(kok);
}
3 references
private void PreOrderInt(İkiliAgacDugumu dugum)
{
    if (dugum == null)
        return;
    Ziyaret(dugum);
    PreOrderInt(dugum.sol);
    PreOrderInt(dugum.sag);
}
```

Preorder ile Gezinme (devam…)

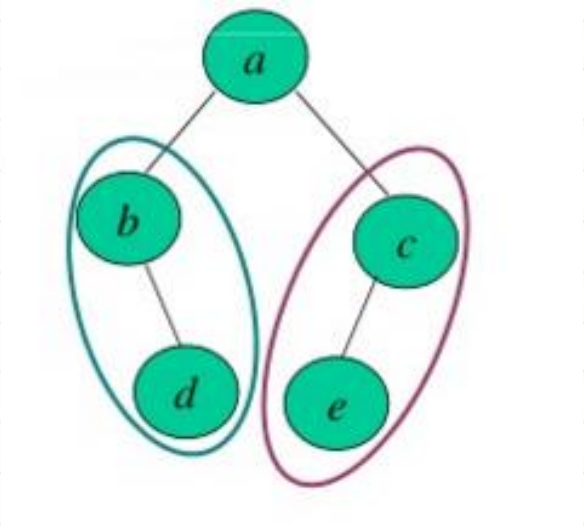



```
private void PreOrderInt(İkiliAgacDugumu dugum)
{
    if (dugum == null)
        return;
    Ziyaret(dugum);
    PreOrderInt(dugum.sol);
    PreOrderInt(dugum.sag);
}
```

Preorder Gezinme:

Inorder ile Gezinme

1. Sol alt ağaçta gezin
2. Düğümü ziyaret et
3. Sağ alt ağaçta gezin

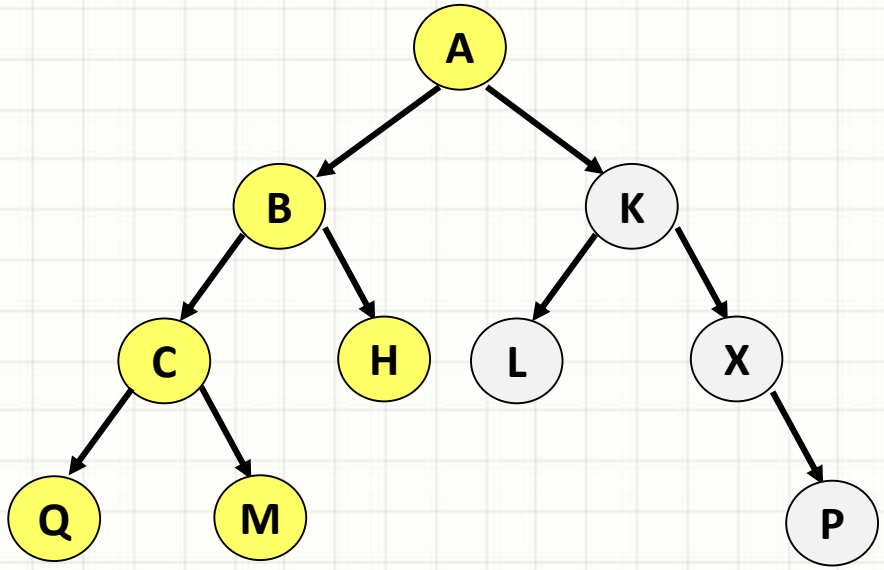


Inorder Gezinme:
b, d, a, e, c

Inorder ile Gezinme (devam…)

```
private void Ziyaret(İkiliAgacDugumu dugum)
{
    dugumler += dugum.veri + " ";
}
2 references
public void InOrder()
{
    dugumler = "";
    InOrderInt(kok);
}
3 references
private void InOrderInt(İkiliAgacDugumu dugum)
{
    if (dugum == null)
        return;
    InOrderInt(dugum.sol);
    Ziyaret(dugum);
    InOrderInt(dugum.sag);
}
```

Inorder ile Gezinme (devam…)

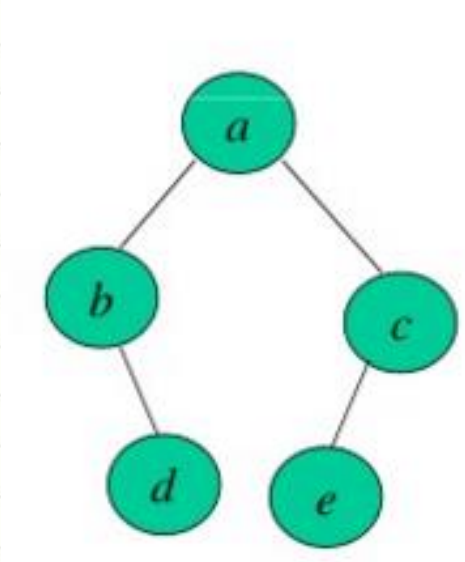



```
private void InOrderInt(İkiliAgacDugumu dugum)
{
    if (dugum == null)
        return;
    InOrderInt(dugum.sol);
    Ziyaret(dugum);
    InOrderInt(dugum.sag);
}
```

Inorder Gezinme:

Postorder ile Gezinme

1. Sol alt ağaçta gezin
2. Sağ alt ağaçta gezin
3. Düğümü ziyaret et

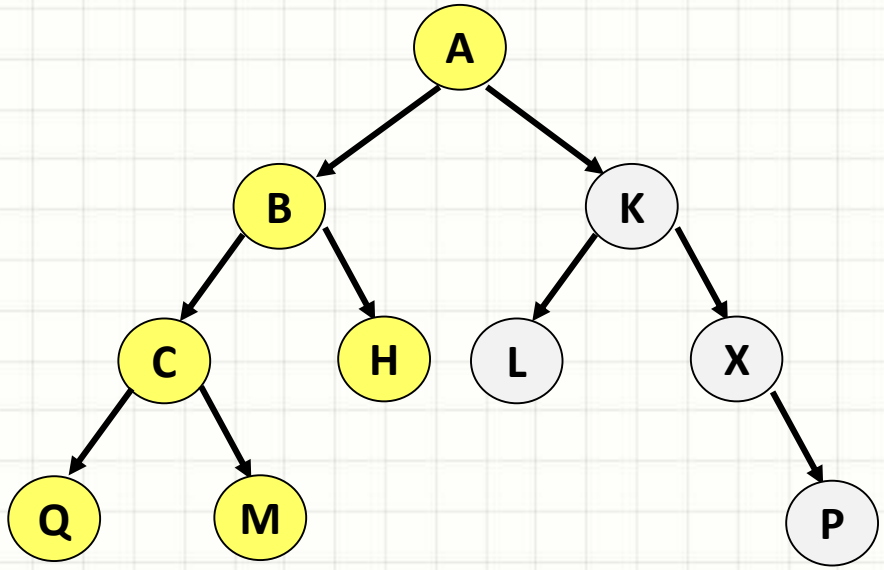


Postorder Gezinme:
d, b, e, c, a

Postorder ile Gezinme (devam…)

```
private void Ziyaret(İkiliAgacDugumu dugum)
{
    dugumler += dugum.veri + " ";
}
1 reference
public void PostOrder()
{
    dugumler = "";
    PostOrderInt(kok);
}
3 references
private void PostOrderInt(İkiliAgacDugumu dugum)
{
    if (dugum == null)
        return;
    PostOrderInt(dugum.sol);
    PostOrderInt(dugum.sag);
    Ziyaret(dugum);
}
```

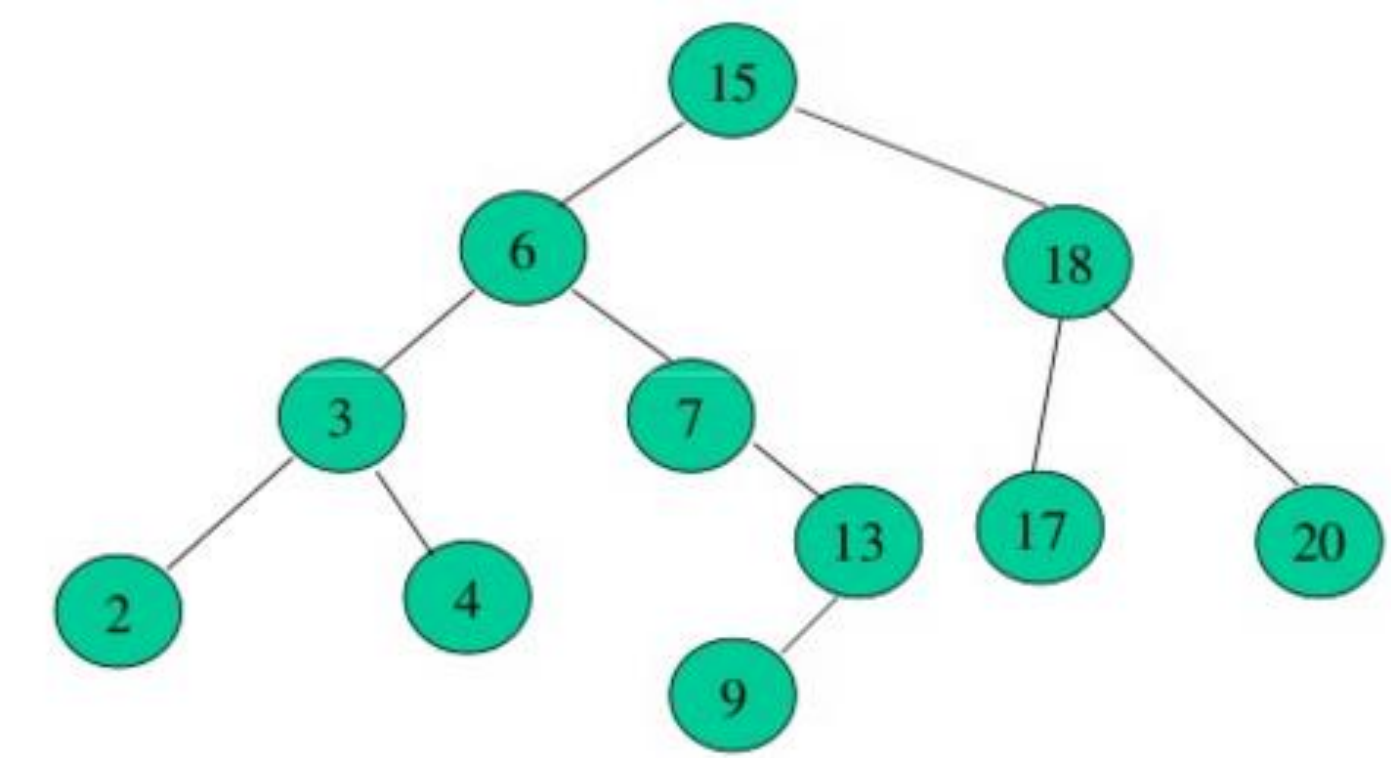
Postorder ile Gezinme (devam…)




```
private void PostOrderInt(İkiliAgacDugumu dugum)
{
    if (dugum == null)
        return;
    PostOrderInt(dugum.sol);
    PostOrderInt(dugum.sag);
    Ziyaret(dugum);
}
```

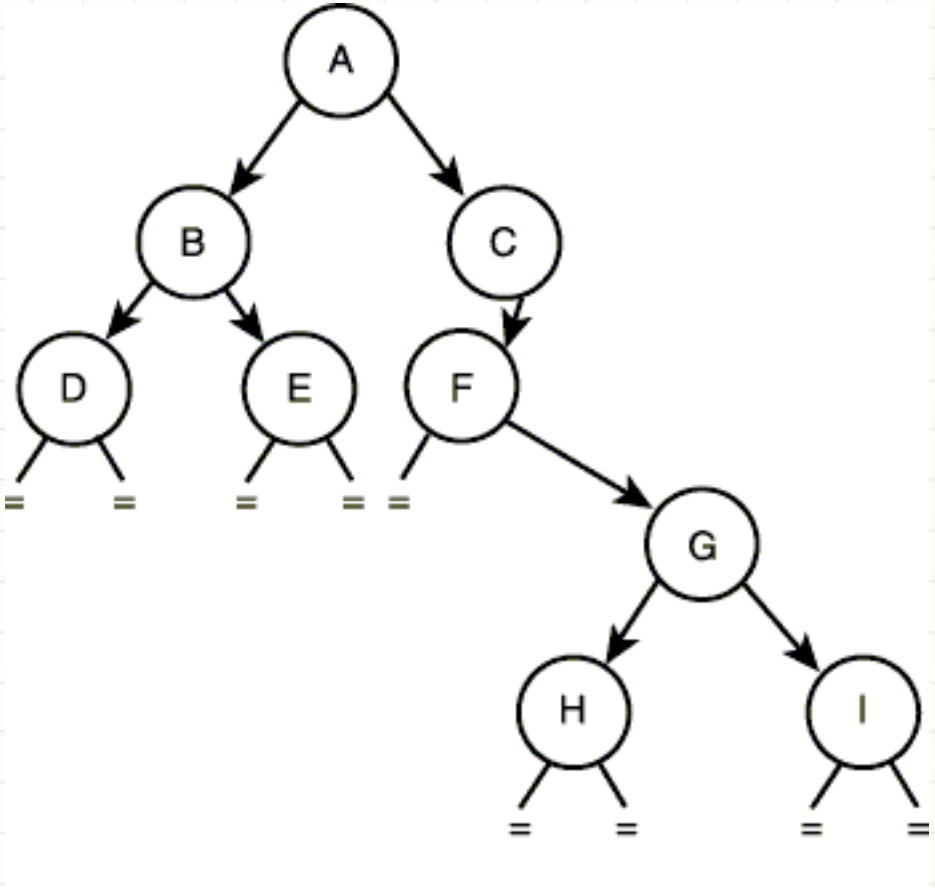
Postorder Gezinme:

Gezinme - Örnek 1



Preorder: 15, 6, 3, 2, 4, 7, 13, 9, 18, 17, 20
Inorder: 2, 3, 4, 6, 7, 9, 13, 15, 17, 18, 20
Postorder: 2, 4, 3, 9, 13, 7, 6, 17, 20, 18, 15

Gezinme - Örnek 2



Pre-order: A B D E C F G H I
In order: D B E A F H G I C
Post order: D E B H I G F C A

Gezinmeler Ne İşe Yarar?

Preorder:

- Ağacın kopyasını elde etme
- Düğümleri sayma
- Yaprakları sayma
- Matematiksel ifadeler için prefix gösterimini elde etme

Postorder:

- İkili ağacı silme
- Fonksiyonel dil derleyicilerinde
- Hesap makinesi programlarında
- Postfix matematiksel ifadelerinin oluşturulmasında

Inorder:

- İkili Arama Ağacında (Binary Search Tree) ağaçtaki bilgiyi sıralı halde yazdırmada

İkili Ağaç Uygulamaları

- **Binary Search Tree (İkili Arama Ağacı):** Birçok arama uygulamasında,
- **Binary Space Partition:** Hemen hemen tüm 3D video oyunlarında, hangi nesnelerin render edilmesi gerektiğini belirlemede,
- **Binary Tries:** Yüksek bant genişliğine sahip router cihazların router-tablolarını saklamada,
- **Hash Trees:** P2P programlarında, imza doğrulamada
- **Heaps:** Öncelik kuyruklarının daha etkin/verimli gerçekleştiriminde kullanılır. Öncelik kuyrukları işletim sisteminde prosesleri planlamak için kullanılır. Yapay zeka uygulamalarında A* yol bulma (path finding) algoritmasının gerçekleştiriminde

İkili Ağaç Uygulamaları (devam…)

- **GGM Trees:** Bir çok şifreleme uygulamasında, sözde rastgele sayıların ağacının oluşturulmasında
- **Treap:** Kablosuz ağlarda ve bellek tahsisinde
- **T-tree:** Bellek içi veritabanlarına (Datablitz, EXtremeDB, MySQL Cluster, Oracle TimesTen) ait işlemlerinde

Stackoverflow tartışmasına bakınız…

İkili Ağaç Gerçekleştirimine Dair Sorular

- Ağaçtaki **düğüm sayısını** nasıl bulurum?
- Ağaçtaki **yaprak sayısını** nasıl bulurum?
- Ağaç **yüksekliğini** nasıl bulurum?
- İkili ağacın **katı (strict) ikili ağaç** olup olmadığını nasıl anlarım?
- İkili ağaçtaki 2 yaprak düğümün en küçük ortak atasını nasıl bulurum (***LCA – Lowest Common Ancestor***)?
- Inorder ve Preorder gezinme sonuçları verilen ***ikili ağacı nasıl oluştururum***?
- Inorder ve Postorder gezinme sonuçları verilen ***ikili ağacı nasıl oluştururum***?

Ağaçtaki Düğüm Sayısını Bulma

```
public int DugumSayisi (İkiliAgacDugumu dugum)
{
    int count = 0;
    if (dugum != null)
    {
        count = 1;
        count += DugumSayisi(dugum.sol);
        count += DugumSayisi(dugum.sag);
    }
    return count;
}
```

Ağaçtaki Yaprak Sayısını Bulma

```
public int YaprakSayisi(İkiliAgacDugumu dugum)
{
    int count = 0;
    if (dugum != null)
    {
        if ((dugum.sol == null) && (dugum.sag == null))
            count = 1;
        else
            count = count + YaprakSayisi(dugum.sol) + YaprakSayisi(dugum.sag);
    }
    return count;
}
```

İki Gezinme Sonucundan Ağaç Elde Etme

- Eğer **Preorder** sonuç verildiyse ilk düğüm **köktür**.
- Eğer **Postorder** sonuç verildiyse son düğüm **köktür**.
- Kök bulunduktan sonra, Inorder gezinmeye göre, ***kökün sol ve sağ tarafındaki düğümler***; sol ve sağ alt ağaç olarak belirlenirler.
- Aynı teknik yeni bulunan sol ve sağ alt ağaç için özyinelemeli (recursive) olarak uygulanır.
- Gezinme sonuçlarından birisi **mutlaka** **Inorder** olmalıdır. Diğeri Preorder veya Postorder olabilir.

İki Gezinme Sonucundan Ağaç Elde Etme (devam…)

Inorder gezinme: 20 30 35 40 45 50 55 60 70

Preorder gezinme: 50 40 30 20 35 45 60 55 70

Soru: İkili ağacı elde ediniz.