Veri Yapıları

Ağaçlar

Ağaç Veri Modeli Temel Kavramları

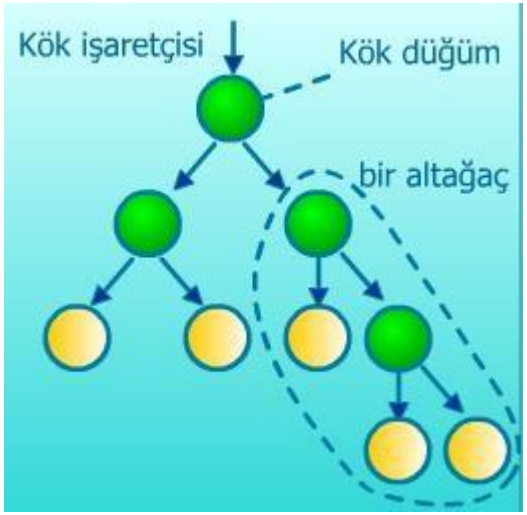
- Ağaç, bir kök işaretçisi, sonlu sayıda düğümleri ve onları birbirine bağlayan dalları olan bir veri modelidir. Aile soyağacında olduğu gibi hiyerarşik bir yapısı vardır ve orada geçen birçok kavram buradaki ağaç veri modelinde de tanımlıdır.

- Örneğin çocuk, kardeş düğüm, aile, ata gibi birçok kavram ağaç veri modelinde de kullanılır. Genel olarak, veri, ağacın düğümlerinde tutulur; dallarda ise geçiş koşulları vardır denilebilir.

- 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 Veri Modeli Temel Kavramları

- Bağlı listeler, yığınlar 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 node'lar ve kenarlardan (edge) oluşur.
  - Herbir node(düğüm) bir nesneyi gösterir.
  - Herbir kenar (bağlantı) iki node arasındaki ilişkiyi gösterir.
  - Arama işlemi bağlı dizilere göre çok hızlı yapılır.

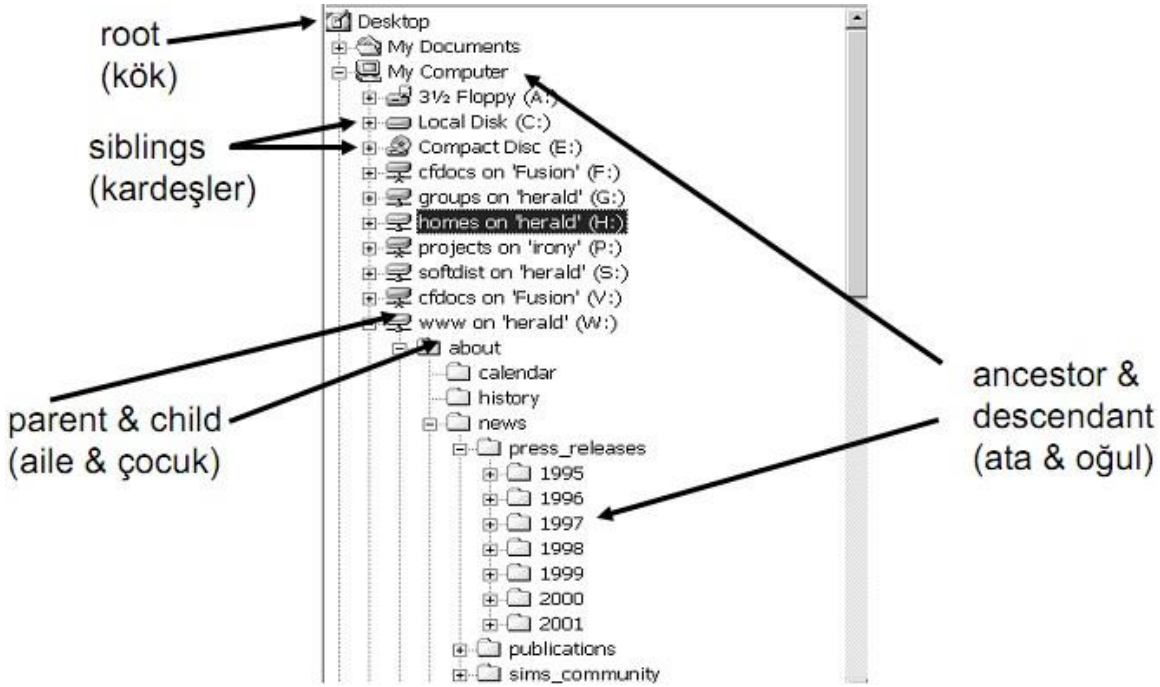


Ağaç Veri Modeli Temel Kavramları

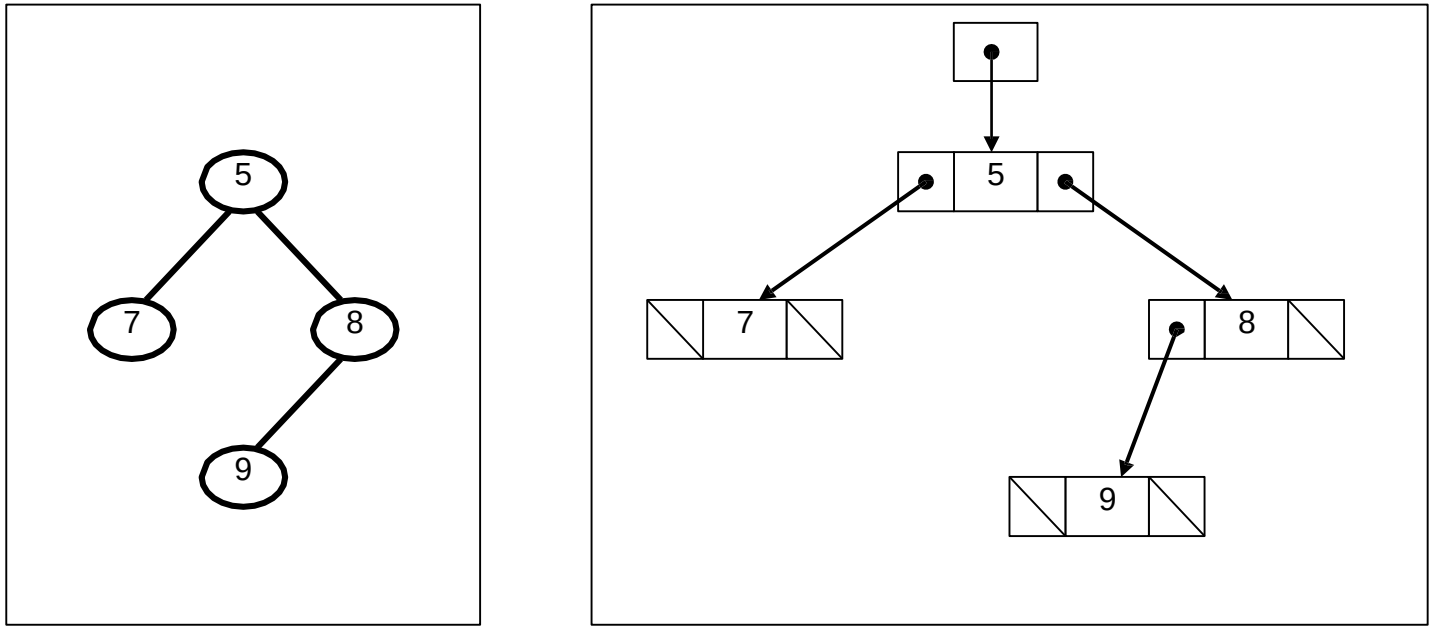
- Ağaçlardaki düğümlerden iki veya daha fazla bağ çıkabilir. İkili ağaçlar (binary trees), düğümlerinde en fazla iki bağ içeren (0,1 veya 2) ağaçlardır. Ağacın en üstteki düğümüne kök (root) adı verilir.
- **Uygulamaları:**
  - Organizasyon şeması
  - Dosya sistemleri
  - Programlama ortamları

Ağaç Veri Modeli Temel Kavramları

- Örnek ağaç yapısı



Ağaç Veri Modeli Temel Kavramları



- **Şekil 1:** İkili ağacın grafiksel gösterimleri

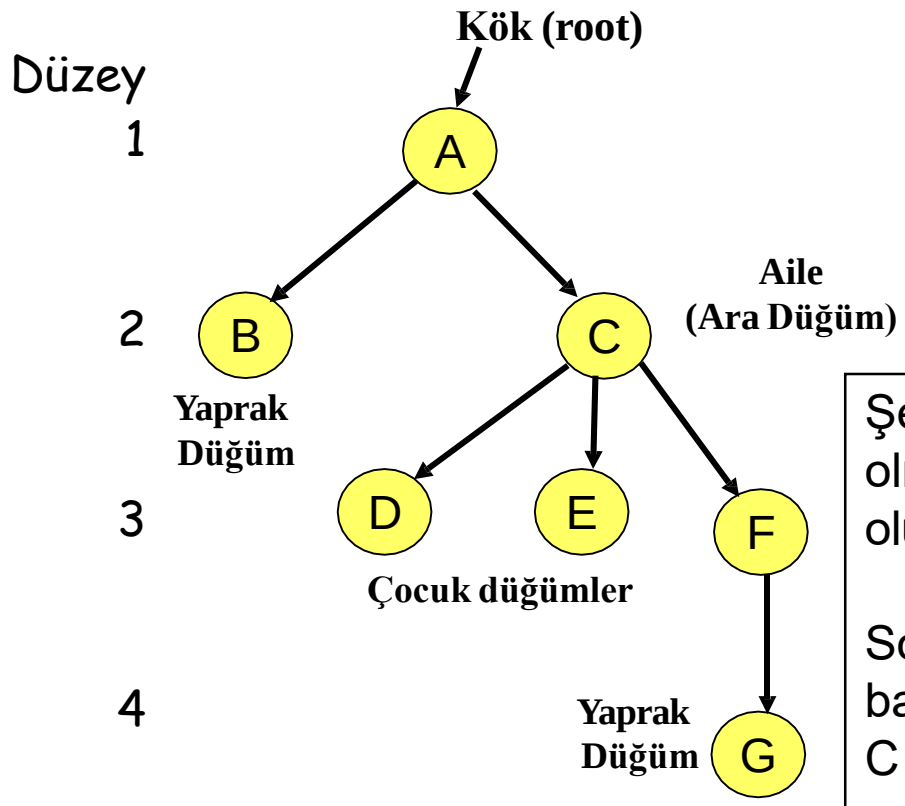
Ağaç Veri Modeli Temel Kavramları

- Şekil 1’de görülen ağacın düğümlerindeki bilgiler sayılardan oluşmuştur. Her düğümdeki sol ve sağ bağlar yardımı ile diğer düğümlere ulaşılır. Sol ve sağ bağlar boş ("NULL" = "/" = "\") da olabilir.

- Düğüm yapıları değişik türlerde bilgiler içeren veya birden fazla bilgi içeren ağaçlar da olabilir.

- Doğadaki ağaçlar köklerinden gelişip göğe doğru yükselirken veri yapılarındaki ağaçlar kökü yukarıda yaprakları aşağıda olacak şekilde çizilirler.

Ağaç Veri Modeli Temel Kavramları



- **Şekil :** Ağaçlarda düzeyler

Şekildeki ağaç, A düğümü kök olmak üzere 7 düğümden oluşmaktadır.

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.

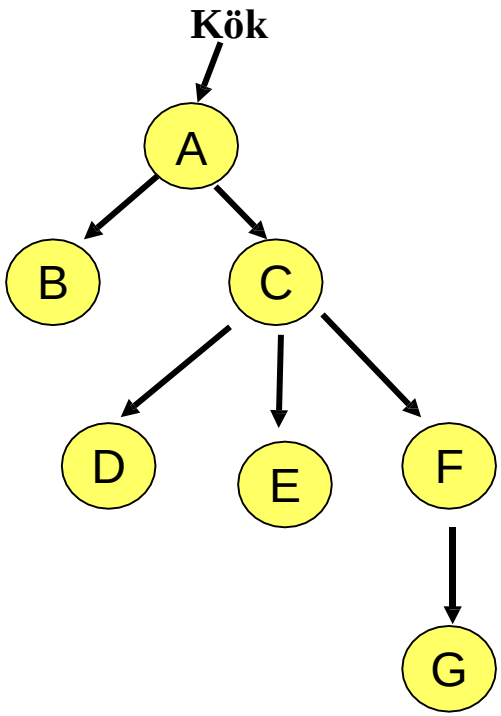
Ağaç Veri Modeline İlişkin Tanımlar

- **Düğüm (Node)**
  - Ağacın her bir elemanına düğüm adı verilir.
- **Kök Düğüm (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) ve Torun (Dedscendant)**
  - Aile düğümünün daha üstünde kalan düğümlere ata denilir; torun, bir düğümün çocuğuna bağlı olan düğümlere denir.

Ağaç Veri Modeline İlişkin Tanımlar

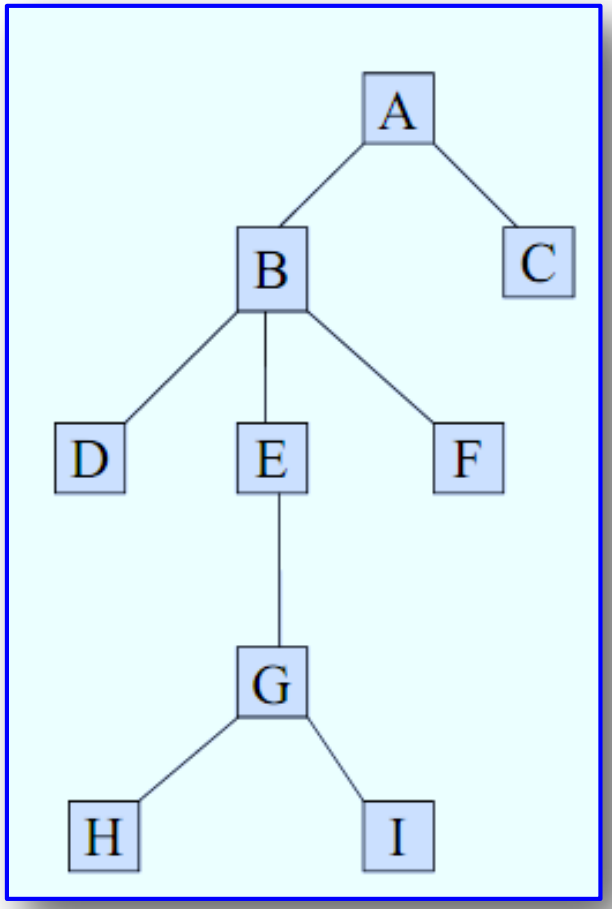
- **Derece (Degree)**
  - Bir düğümden alt hiyerarşiye yapılan bağlantıların sayısıdır; yani çocuk veya alt ağaç sayısıdır.
- **Düzey (Level) ve Derinlik (Depth)**
  - Düzey, iki düğüm arasındaki yolun üzerinde bulunan düğümlerin sayısıdır. Kök düğümün düzeyi 1, doğrudan köke bağlı düğümlerin düzeyi 2'dir. Bir düğümün köke olan uzaklığı ise derinliktir. Kök düğümün derinliği 1 dir.
- **Yaprak (Leaf)**
  - Ağacın en altında bulunan ve çocukları olmayan düğümlerdir.
- **Yükseklik (Height)**
  - Bir düğümün kendi silsilesinden en uzak yaprak düğüme olan uzaklığıdır.
- **Yol(Path)**
  - Bir düğümün aşağıya doğru (çocukları üzerinden) bir başka düğüme gidebilmek için üzerinden geçilmesi gereken düğümlerin listesidir.

Ağaç Veri Modeline İlişkin Tanımlar



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

Ağaç Veri Modeline İlişkin Tanımlar



Tanım	Değer
Düğüm sayısı	9
Yükseklik	4
Kök düğüm	A
Yapraklar	C, D, F, H, I
Düzey sayısı	5
H'nin ataları	E, B, A
B'nin torunları	G, H, I
E'nin kardeşleri	D, F
Sağ alt ağaç	Yok
Sol alt ağaç:	B

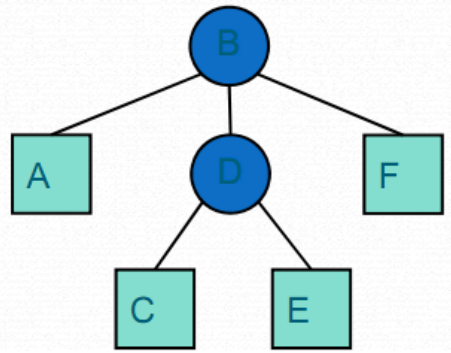
Ağaçların Bağlı Yapısı

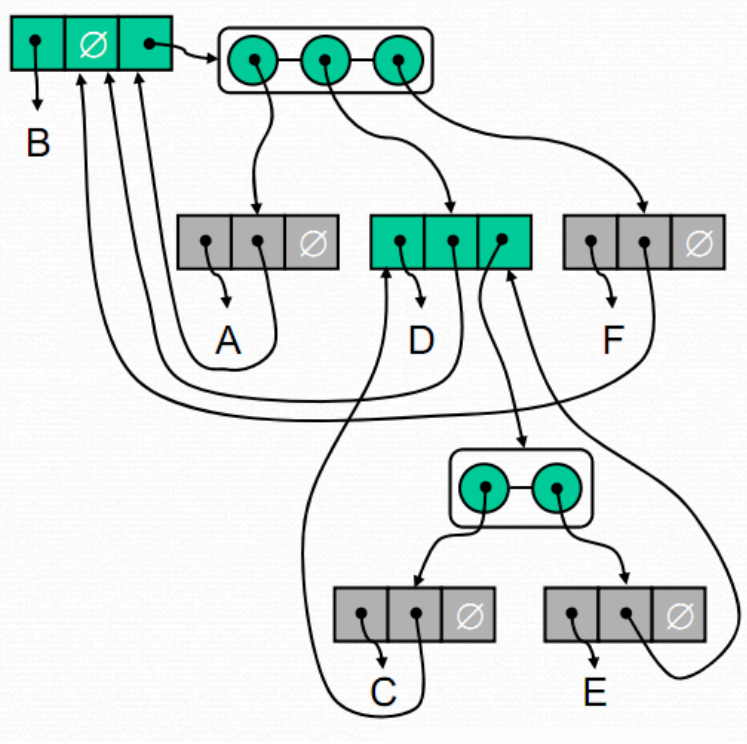
- Bir düğüm çeşitli bilgiler ile ifade edilen bir nesnedir. Her bir bağlantı için birer bağlantı bilgisi tutulur.
  - Nesne/Değer (Element)
  - Ana düğüm (Parent node)
  - Çocuk düğümlerin listesi

- Problem: Bir sonraki elemanın çocuk sayısını bilmiyoruz.

- Daha iyisi: Çocuk/Kardeş Gösterimi
  - Her düğümde iki bağlantı bilgisi tutularak hem çocuk hem de yandaki kardeş tutulabilir.
  - İstenildiği kadar çocuk/kardeş olabilir.

```
JAVA Declaration
class AgacDugumu {
    int eleman;
    AgacDugumu ilkCocuk;
    AgacDugumu kardes; }
```





Ağaç Türleri

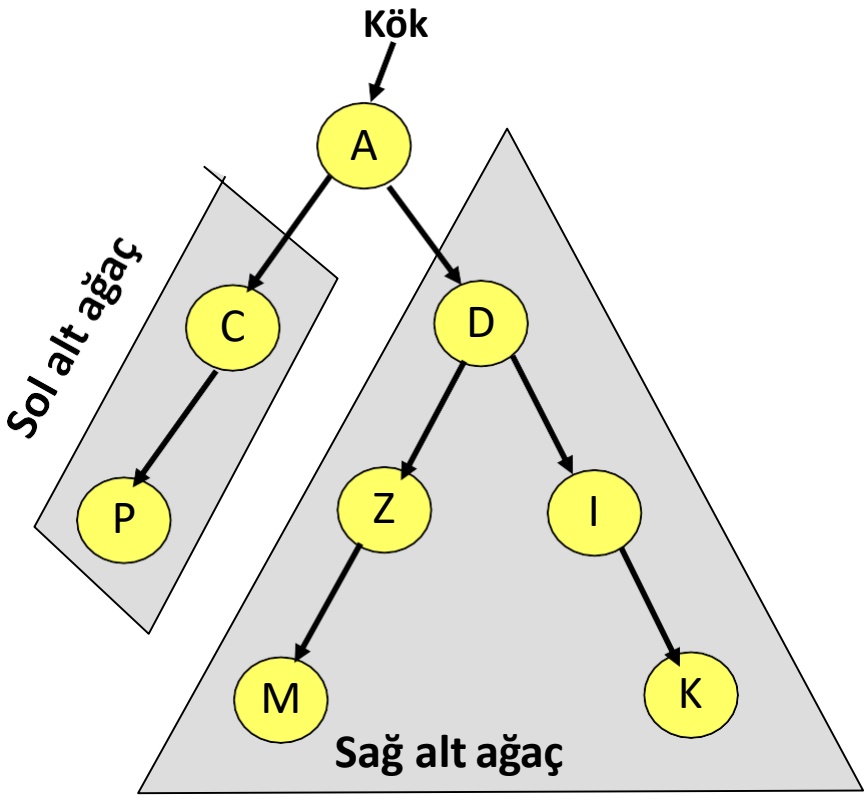
- En çok bilinen ağaç türleri ikili arama ağacı olup kodlama ağacı, sözlük ağacı, kümeleme ağacı gibi birçok ağaç uygulaması vardır.
  - **İkili Arama Ağacı (Binary Search Tree):**
    - İkili arama ağacında bir düğüm en fazla iki tane çocuğa sahip olabilir ve alt/çocuk bağlantıları belirli bir sırada yapılır.
  - **Kodlama Ağacı (Coding Tree):**
    - Bir kümedeki karakterlere kod ataması için kurulan ağaç şeklidir. Bu tür ağaçlarda kökten başlayıp yapraklara kadar olan yol üzerindeki bağlantı değerleri kodu verir.
  - **Sözlük Ağacı(Dictonary Tree):**
    - Bir sözlükte bulunan sözcüklerin tutulması için kurulan bir ağaç şeklidir. Amaç arama işlemini en performanslı bir şekilde yapılması ve belleğin optimum kullanılmasıdır.
  - **Kümeleme Ağacı (Heap Tree):**
    - Bir çeşit sıralama ağacıdır. Çocuk düğümler her zaman aile düğümlerinden daha küçük değerlere sahip olur.

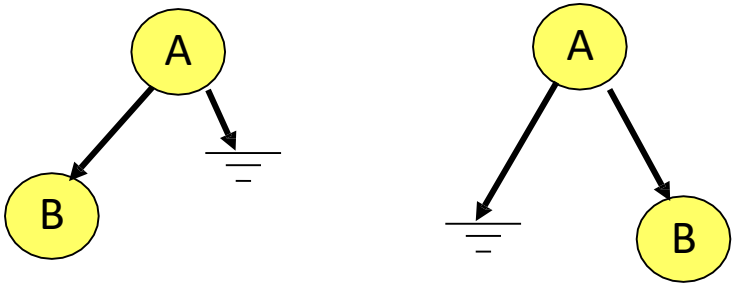
İkili Ağaç (Binary Tree) :

- Sonlu düğümler kümesidir. Bu küme boş bir küme olabilir (empty tree). Boş değilse şu kurallara uyar.
  - Kök olarak adlandırılan özel bir düğüm vardır.
  - Her düğüm en fazla iki düğüme bağlıdır.
    - Left child : Bir node'un sol işaretçisine bağlıdır.
    - Right child : Bir node'un sağ işaretçisine bağlıdır.
  - Kök hariç her düğüm bir daldan gelmektedir.
  - Tüm düğümlerden yukarı doğru çıkıldıkça sonuçta köke ulaşılır.

İkili Ağaç (Binary Tree) :

- Bilgisayar bilimlerinde en yaygın ağaçtır.

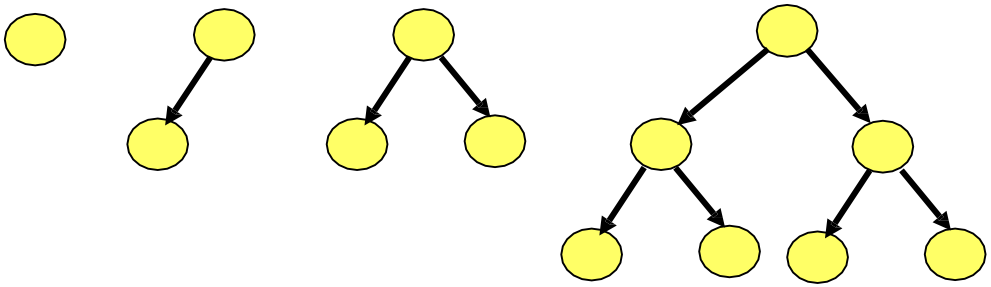




İki farklı ikili ağaç

İkili Ağaç (Binary Tree)

- N tane düğüm veriliyor, İkili ağacın minimum derinliği nedir.



Derinlik 1: N = 1, 1 düğüm $2^1 - 1$

Derinlik 2: N = 2, 3 düğüm, $2^2 - 1$ düğüm

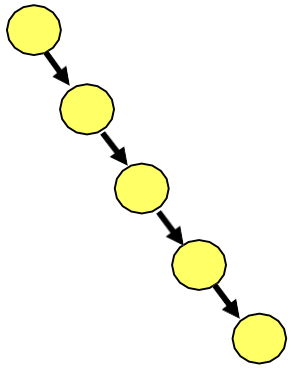
Herhangi bir d derinliğinde, N = ?

Derinlik d: N = $2^d - 1$ düğüm (tam bir ikili ağaç)

En küçük derinlik: **Θ(log N)**

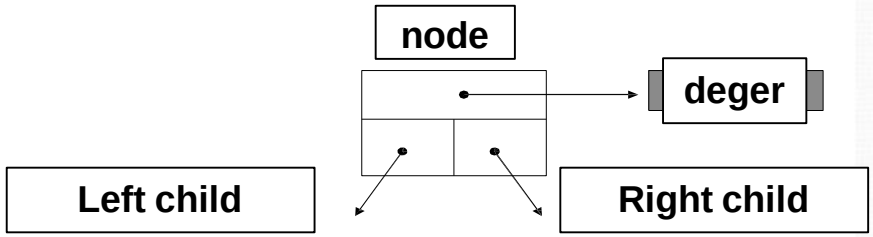
İkili Ağaç

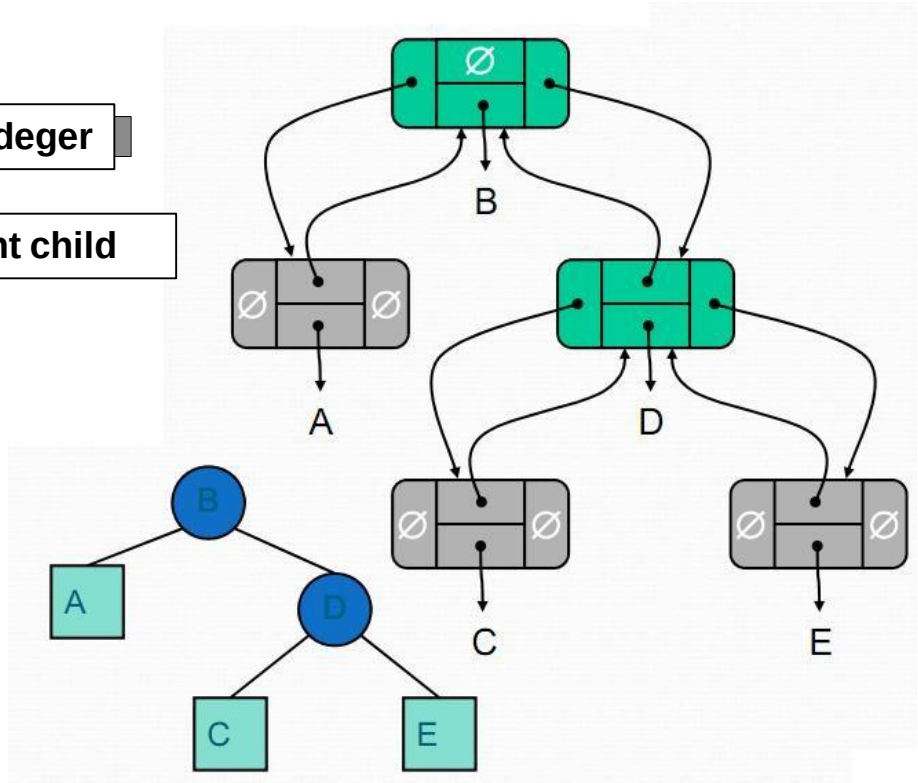
- N düğümlü ikili ağacın minimum derinliği: **Θ(log N)**
- İkili ağacın maksimum derinliği ne kadardır?
  - Dengesiz ağaç: Ağaç bir bağlantılı liste olursa!
  - Maksimum derinlik = N
- Amaç: Arama gibi operasyonlarda bağlantılı listeden daha iyi performans sağlamak için derinliğin log N de tutulması gerekmektedir.



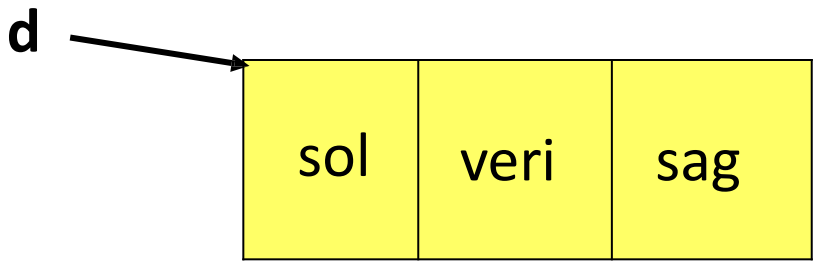
- Bağlantılı liste
- Derinlik = N

İkili Ağaç Bağlı Liste Yapısı

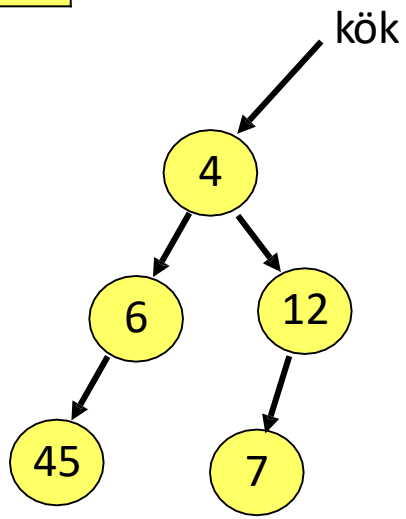




İkili Ağaç Gerçekleştirimi



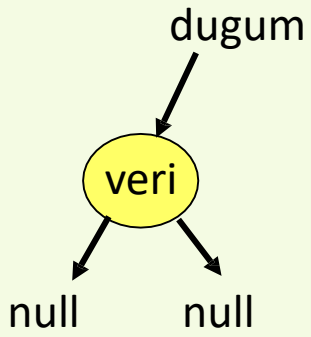

```
public class İkiliAgacDugumu {
    public İkiliAgacDugumu sol;
    public int veri;
    public İkiliAgacDugumu sag;
}
```



İkili Ağaç Gerçekleştirimi

- İkili ağaç için düğüm oluşturma. Her defasında sol ve sağı boş olan bir düğüm oluşturulur.
  - `/* İkili ağaç düğümü oluşturur. */ İkiliAgacDugumu DugumOlustur(int veri){ İkiliAgacDugumu dugum = new İkiliAgacDugumu(); dugum.veri = veri;`

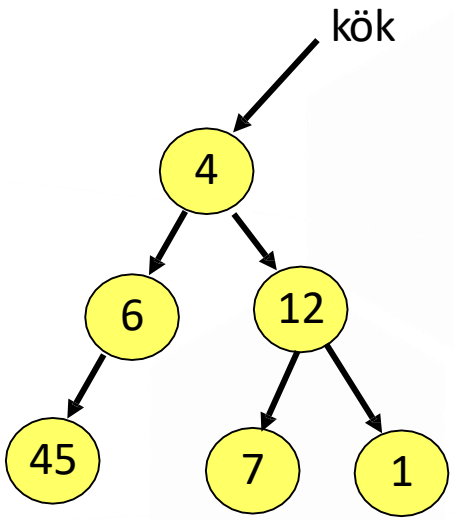
```
    dugum.sol = null;
    dugum.sag = null;
    return dugum;
}
```



İkili Ağaç Gerçekleştirimi

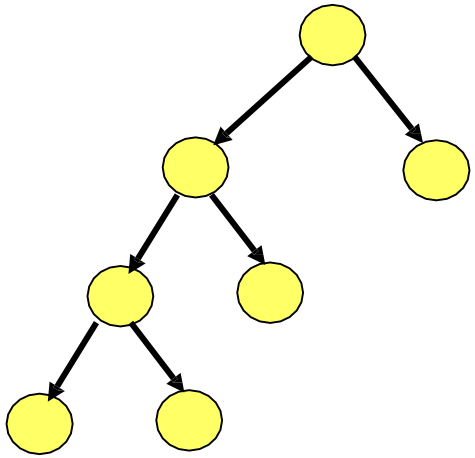
- İteratif düğüm oluşturma ve ağaca ekleme.

```
İkiliAgacDugumu dugum = null;

public static void main main(){
   kok = DugumOlustur(4);

   kok.sol = DugumOlustur(6);
   kok.sag = DugumOlustur(12);
   kok.sol.sol = DugumOlustur(45);
   kok.sag.sol = DugumOlustur(7);
   kok.sag.sag = DugumOlustur(1);
} /* main */
```

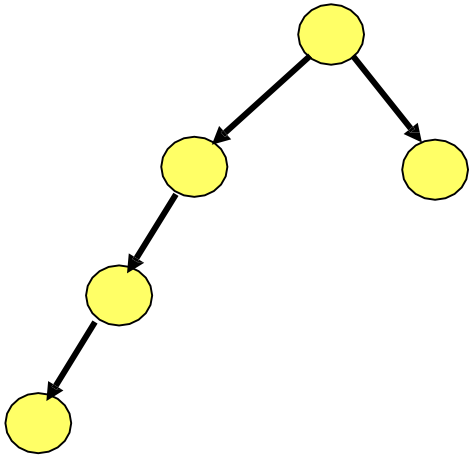


Proper (düzgün) Binary Trees

- Yaprak olmayan düğümlerin tümünün iki çocuğu olan T ağacı proper(düzgün) binary tree olarak adlandırılır.



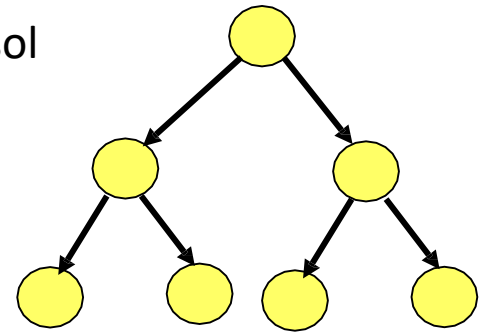
- Proper Tree



ImProper Tree

Full Binary Tree

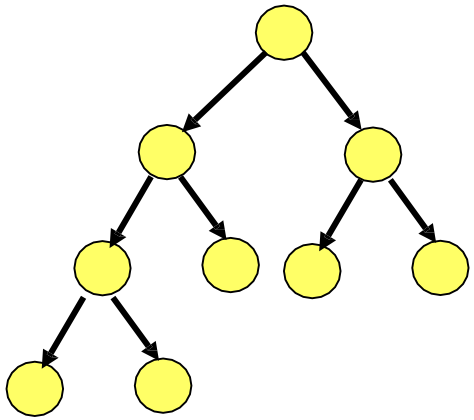
- Full binary tree:
  - 1- Her yaprağı aynı derinlikte olan
  - 2- Yaprak olmayan düğümlerin tümünün iki çocuğu olan ağaç Full (Strictly) Binary Tree'dir.
- Bir full binary tree'de n tane yaprak varsa bu ağaçta toplam 2n-1 düğüm vardır. Başka bir şekilde ifade edilirse,
  - Eğer T ağacı boş ise, T yüksekliği 0 olan bir full binary ağaçtır.
  - T ağacının yüksekliği h ise ve yüksekliği h'den küçük olan tüm node'lar iki child node'a sahipse, T full binary tree'dir.
  - Full binary tree'de her node aynı yüksekliğe eşit sağ ve sol altağaçlara sahiptir.



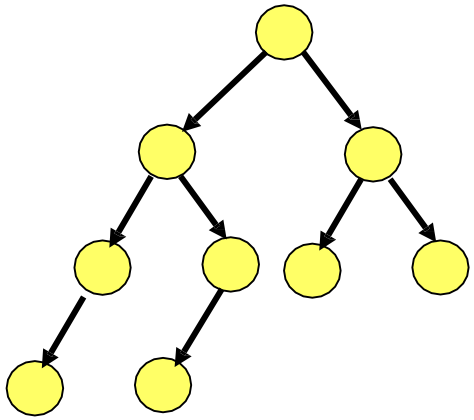
Complete Binary Tree

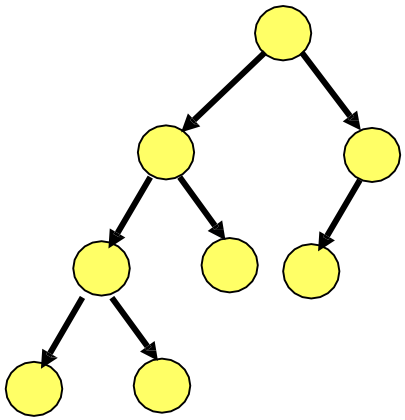
- Full binary tree'de, yeni bir derinliğe soldan sağa doğru düğümler eklendiğinde oluşan ağaçlara Complete Binary Tree denilir.
- Böyle bir ağaçta bazı yapraklar diğerlerinden daha derindir. Bu nedenle full binary tree olmayabilirler. En derin düzeyde düğümler olabildiğince soldadır.
-
  - T, n yükseklikte complete binary tree ise, tüm yaprak node'ları n veya n-1 derinliğindedir ve yeni bir derinliğe soldan sağa doğru ekleme başlanır.
  - Her node iki tane child node'a sahip olmayabilir.

Complete Binary Tree



Complete

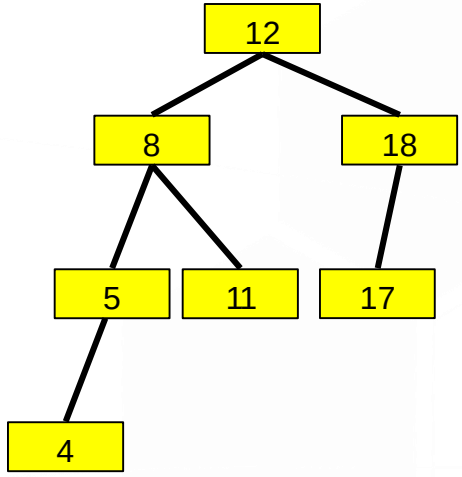




incomplete

Balanced Binary Trees

- Yüksekliği ayarlanmış ağaçlardır.
- Bütün düğümler için sol altağacın yüksekliği ile sağ altağacın yüksekliği arasında en fazla bir fark varsa balanced binary tree olarak adlandırılır.
- Complete binary tree'ler aynı zamanda balanced binary tree'dir.
- Her balanced binary tree, complete binary tree olmayabilir. (Neden?)

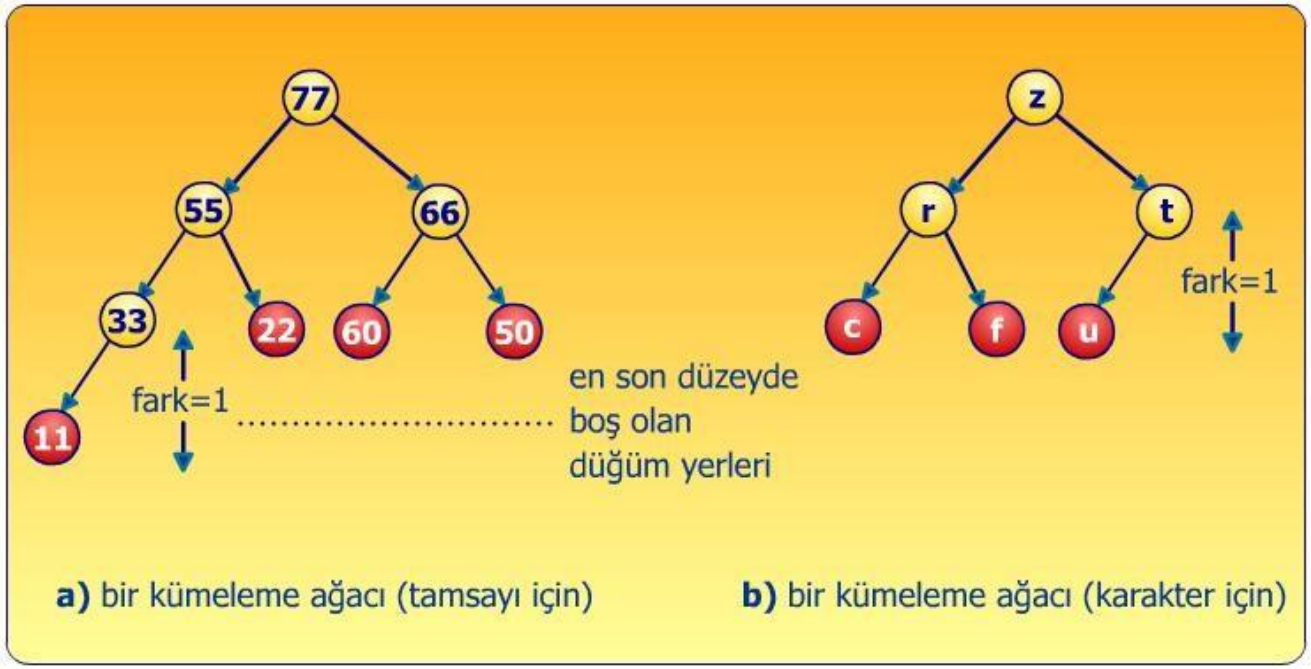


Heap Tree (Kümele Ağacı)

- Kümeleme ağacı, ağaç oluşturulurken değerleri daha büyük olan düğümlerin yukarıya, küçük olanların da aşağıya eklenmesine dayanan tamamlanmış bir ağaçtır.
- Böylece 0. düzeyde, kökte, en büyük değer bulunurken yaprakların olduğu *k*. düzeyde en küçük değerler olur. Yani büyükten küçüğe doğru bir kümeleme vardır; veya tersi olarak küçükten büyüğe kümeleme de yapılabilir.
- Aksi belirtilmediği sürece kümeleme ağacı, sayısal veriler için büyükten küçüğe, sözcükler için alfabetik sıralamaya göre yapılır.
- Kümeleme ağacı bilgisayar biliminde bazı problemlerin çözümü için çok uygundur; hatta birebir uyuşmaktadır denilebilir. Üstelik, hem bellek alanı hem de yürütme zamanı açısından getirisi olur.

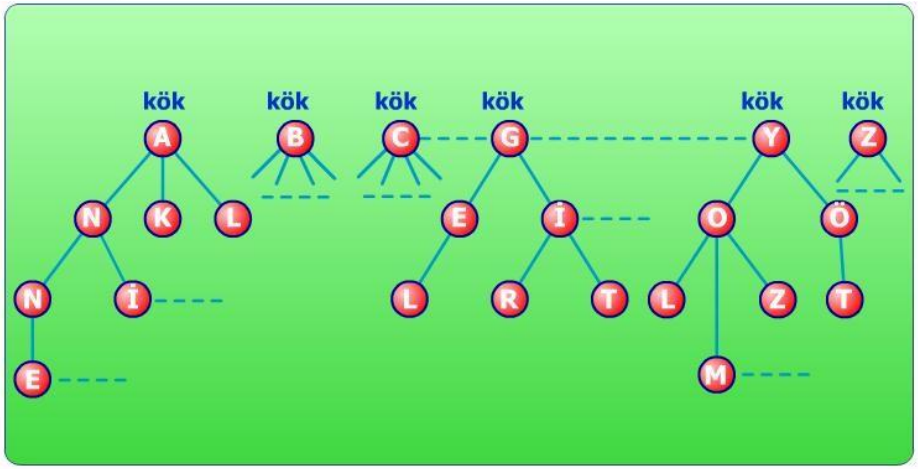
Heap Tree (Kümele Ağacı)

- Bir düğüm her zaman için çocuklarından daha büyük değere sahiptir.
- Yaprak düğümlerin herhangi ikisi arasındaki fark en fazla 1 dir.
- En son düzey hariç tüm düzeyler düğümlerle doludur.
- En son düzeyde oluşan boşluklar sadece sağ taraftadır
- Sıralama işlemlerinde kullanılır.



Trie Ağacı/Sözlük Ağacı

- Sözlük ağacı, bir sözlükte bulunan sözcükleri tutmak ve hızlı arama yapabilmek amacıyla düşünülmüştür; bellek gereksinimi arttırmadan, belki de azaltarak, onbinlerce, yüzbinlerce sözcük bulunan bir sözlükte 10-15 çevrim yapılarak aranan sözcüğün bulunması veya belirli bir karakter kadar uyuşanının bulunması için kullanılmaktadır.
- Sözlük ağacı, sıkıştırma, kodlama gibi sözlük kurulmasına dayalı algoritmalarda ve bir dilin sözlüğünü oluşturmada kullanılmaktadır.



Geçiş İşlemleri
İkili Ağaçlar Üzerindeki Geçiş İşlemleri

- Geçiş (traverse), ağaç üzerindeki tüm düğümlere uğrayarak gerçekleştirilir. Ağaçlar üzerindeki geçiş işlemleri, ağaçtaki tüm bilgilerin listelenmesi veya başka amaçlarla yapılır.

- Doğrusal veri yapılarında baştan sona doğru dolaşmak kolaydır. Ağaçlar ise düğümleri doğrusal olmayan veri yapılarıdır. Bu nedenle farklı algoritmalar uygulanır.

İkili Ağaçlar Üzerindeki Geçiş İşlemleri

- **Preorder (depth-first order) Dolaşma (Traversal) (Önce Kök)**
  - **Köke uğra (visit)**
  - **Sol alt ağacı preorder olarak dolaş.**
  - **Sağ alt ağacı preorder olarak dolaş.**
- **Inorder (Symmetric order) Dolaşma(Ortada Kök)**
  - **Sol alt ağacı inorder'a göre dolaş**
  - **Köke uğra (visit)**
  - **Sağ alt ağacı inorder'a göre dolaş.**
- **Postorder Dolaşma (Sonra Kök)**
  - **Sol alt ağacı postorder'a göre dolaş**
  - **Sağ alt ağacı postorder'a göre dolaş.**
  - **Köke uğra (visit)**
- **Level order Dolaşma (Genişliğine dolaşma)**
  - **Köke uğra**
  - **Soldan sağa ağacı dolaş**

Preorder Geçişi

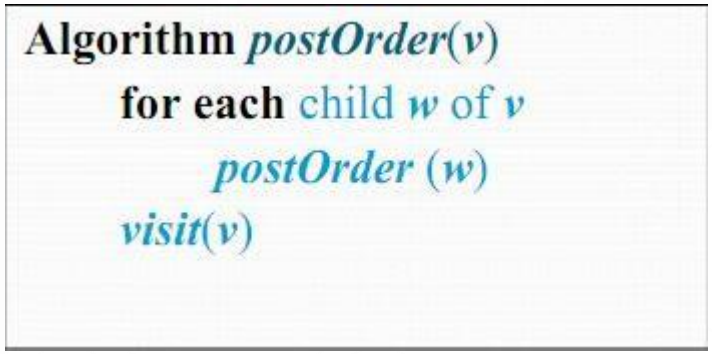
- Preorder geçişte, bir düğüm onun neslinden önce ziyaret edilir.
- Uygulama: yapılandırılmış bir belgenin yazdırılması

```
Algorithm preOrder(v)
    visit(v)
    for each child w of v
        preorder (w)
```

```
OnceKok(IkiliAgacDugumu kok)
{
   if (kok == null) return;
   System.out.print(kok.veri+" ");
   OnceKok(kok.sol);
   OnceKok(kok.sag);
}
```

Postorder Geçişi

- Postorder geçişte, bir düğüm onun neslinden sonra ziyaret edilir.
- Uygulama: Bir klasör ve onun alt klasörlerindeki dosyaların kullandıkları alanın hesaplanması.



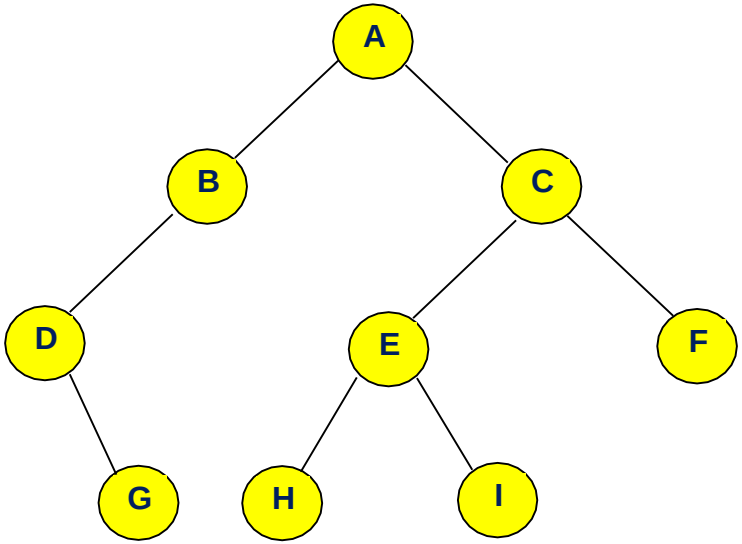

```
SonraKok(IkiliAgacDugumu kok)
  {
  if (kok == null) return;
  SonraKok(kok.sol);  SonraKok(kok.sag);
  System.out.print(kok.veri+" ");
  }
```

Inorder Geçişi

- Inorder geçişte, bir düğüm sol alt ağaçtan sonra ve sağ alt ağaçtan önce ziyaret edilir.
- Uygulama: ikili ağaç çizmek
- x(v) = v düğümünün inorder sıralaması (rank)
- y(v) = v'nin derinliği


```
OrtadaKok (IkiliAgacDugumu kok)
{
   if (kok == null) return;
   OrtadaKok(kok.sol);
   System.out.print(kok.veri+" ");
   OrtadaKok(kok.sag);
}
```

İkili Ağaçlar Üzerindeki Geçiş İşlemleri



PreOrder : A B D G C E H I F

InOrder : D G B A H E I C F

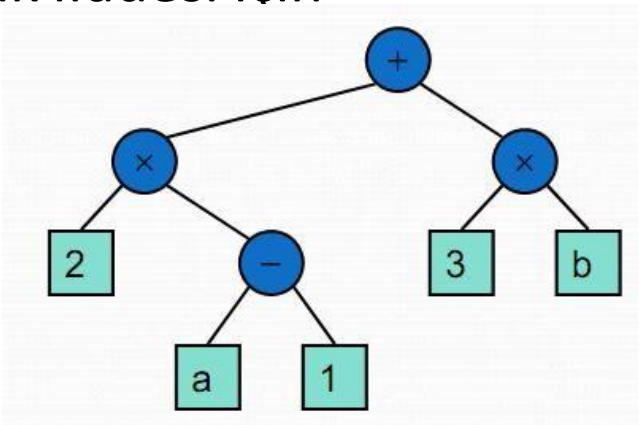
PostOrder : G D B H I E F C A

LevelOrder: A B C D E F G H I

Bağıntı (İfade) Ağaçları (Expression Tree)

- Bağıntı ağaçları bir matematiksel bağıntının ağaç şeklinde tutulması için tanımlanmıştır. Bir aritmetik ifade ile ilişkili ikili ağaçtır.
- Ağacın genel yapısı:
  - Yaprak düğüm = değişken/sabit değer
  - Kök veya ara düğümler = operatörler
  - Birçok derleyicide kullanılır. Parantez gereksinimi yoktur.

- Örnek: (2 × (a − 1) + (3 × b)) aritmetik ifadesi için ağaç yapısı



Bağıntı Ağaçlarında Geçiş

- preOrder, (refix)- Ön-takı
  - + * 2 3 / 8 4
- inOrder, (inf ix)- İç-takı
  - 2 * 3 + 8 / 4
- postOrder, (postfix) Son-takı
  - 2 3 * 8 4 / +
- levelOrder,
  - + * / 2 3 8 4