AĞAÇ (TREE) Veri Modeli

Ağaç Veri Modeli Temel Kavramları

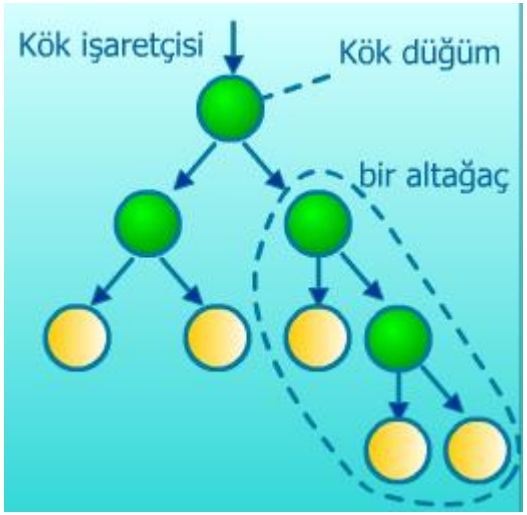
- Ağaç, bir kök işaretçisi, sonlu sayıda düğümleri ve onları birbirine bağlayan dalları olan bir veri modelidir; aynı aile soyağacında olduğu gibi hiyerarşik bir yapısı vardır ve orada geçen birçok kavram buradaki ağaç veri modelinde de tanımlıdır.

- 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, 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 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.

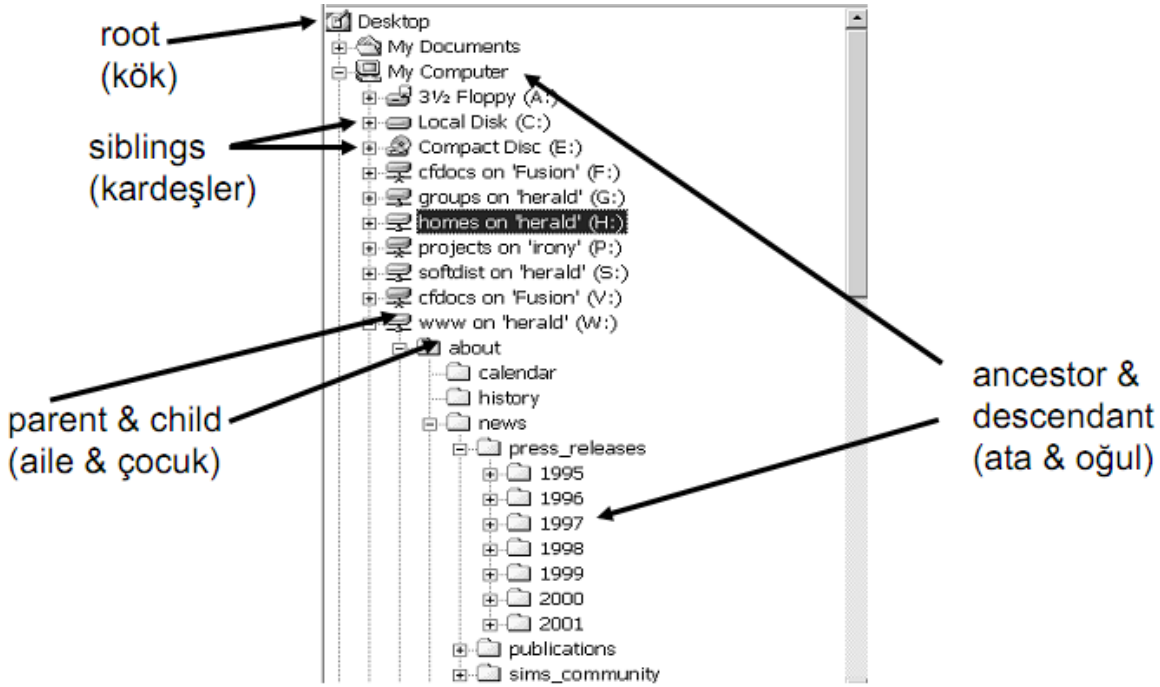


Ağaç Veri Modeli Temel Kavramları

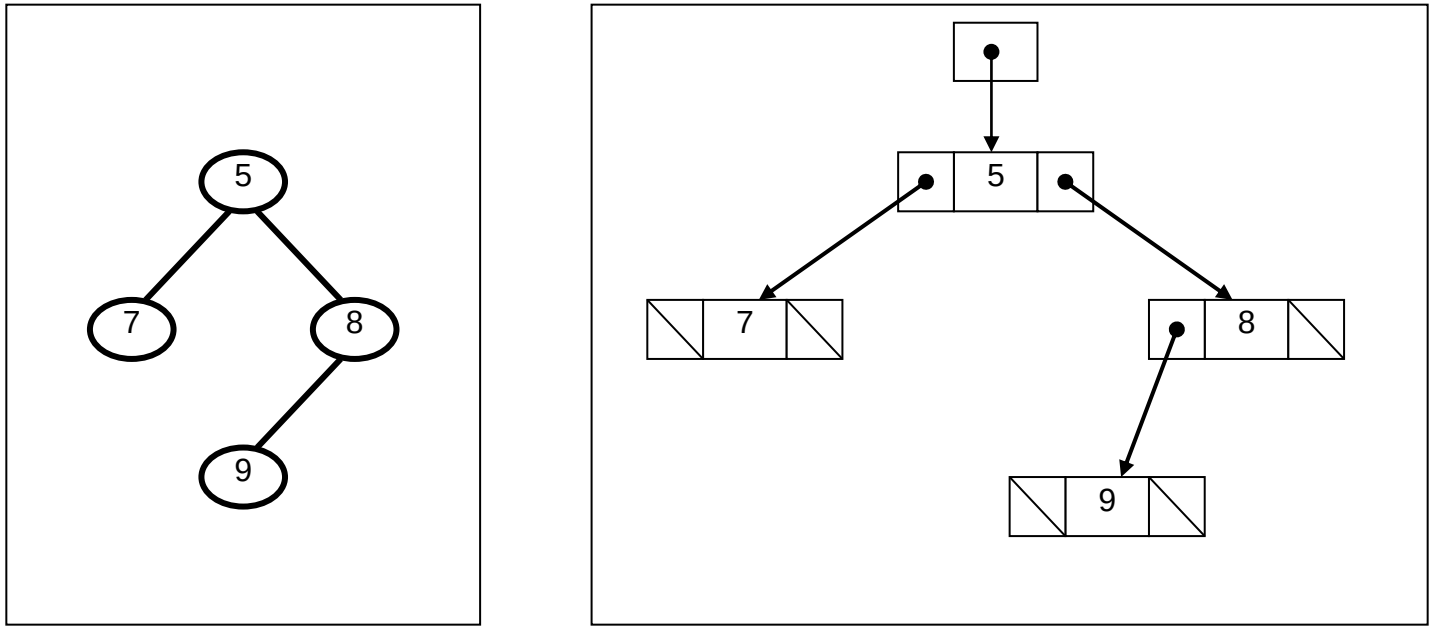
- Ağaçlardaki düğümlerden iki veya daha fazla bağ çıkabilir. İkili ağaçlar (binary trees), düğümlerinde en fazla iki bağ içeren (0,1 veya 2) ağaçlardır. Ağacın en üstteki düğümüne kök (root) adı verilir.
- **Uygulamaları:**
  - Organizasyon şeması
  - Dosya sistemleri
  - Programlama ortamları

Ağaç Veri Modeli Temel Kavramları

- Örnek ağaç yapısı



Ağaç Veri Modeli Temel Kavramları

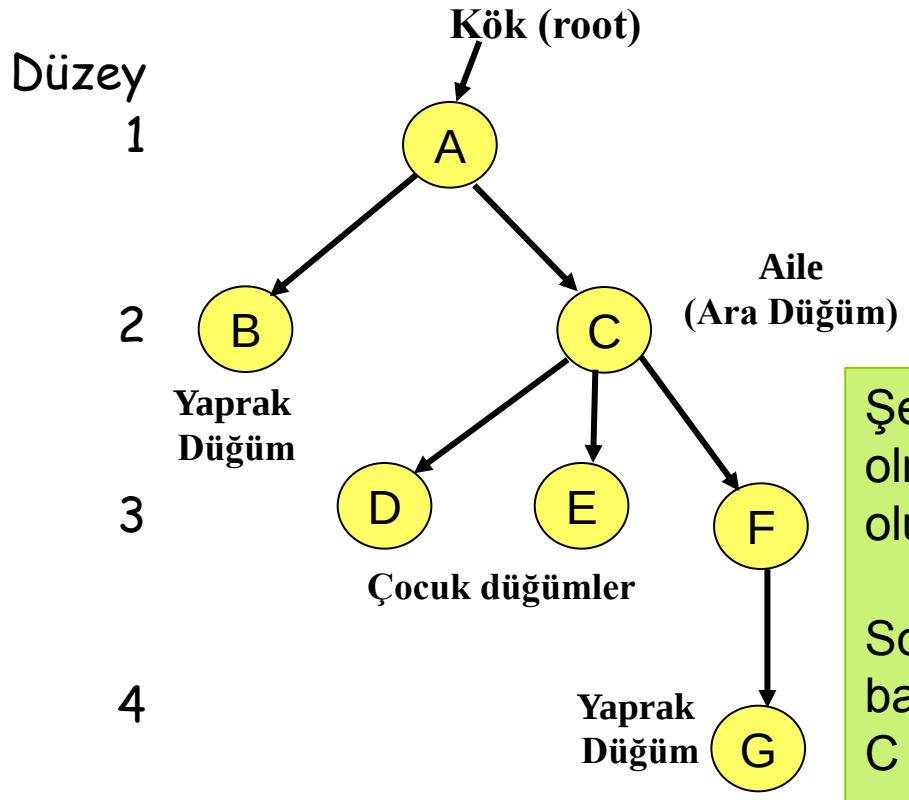


- **Şekil 1:** İkili ağacın grafiksel gösterimleri

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.

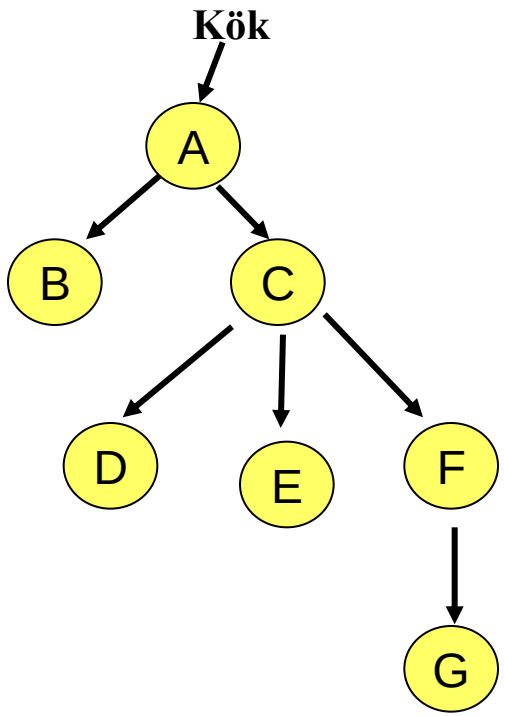
Ağaç Veri Modeline İlişkin Tanımlar

- **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

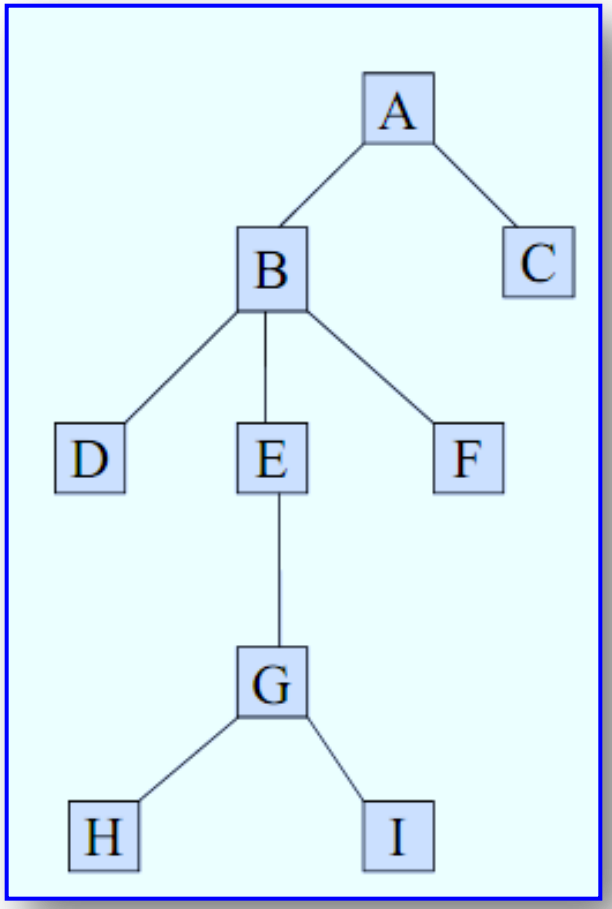
- **Derece (Degree)**
  - Bir düğümden alt hiyerarşiye yapılan bağlantıların sayısıdır; yani çocuk veya alt ağaç sayısıdır.
- **Düzey (Level) ve Derinlik (Depth)**
  - Düzey, iki düğüm arasındaki yolun üzerinde bulunan düğümlerin sayısıdır. Kök düğümün düzeyi 1, doğrudan köke bağlı düğümlerin düzeyi 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

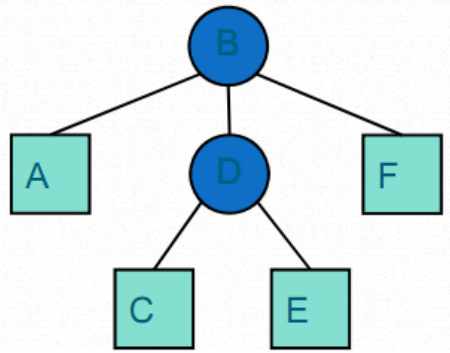


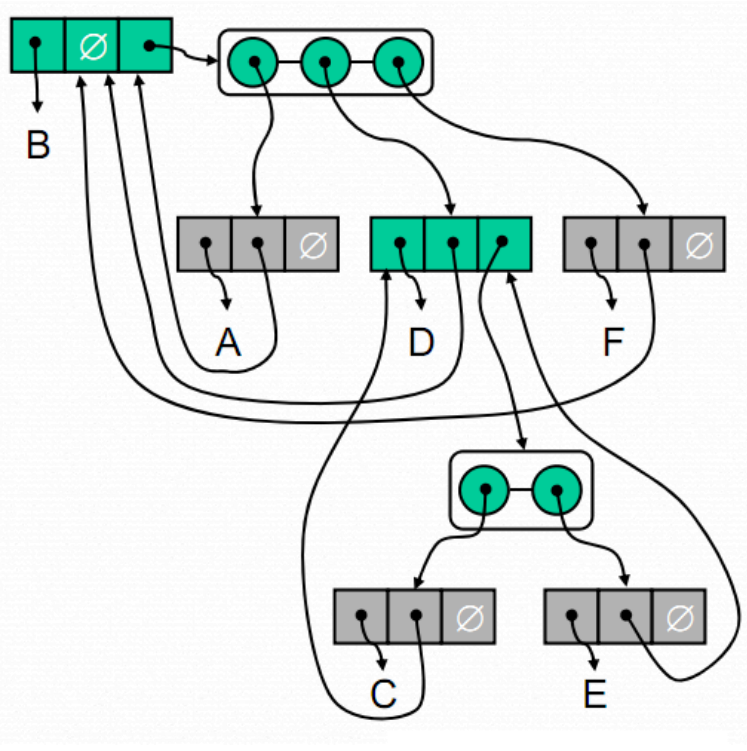
Tanım	Değer
Düğüm sayısı	9
Yükseklik	4
Kök düğüm	A
Yapraklar	C, D, F, H, I
Düzey sayısı	5
H'nin ataları	E, B, A
B'nin torunları	G, H, I
E'nin kardeşleri	D, F
Sağ alt ağaç	Yok
Sol alt ağaç:	B

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 İşlemleri

- Genel Yöntemler:
  - integer size()
  - boolean isEmpty()
  - elements()
  - positions()
- Erişim yöntemleri:
  - root()
  - parent(p)
  - children(p)
- Sorgu (Query) Yöntemleri:
  - boolean isInternal(p)
  - boolean isExternal(p)
  - boolean isRoot(p)
- Güncelleme (Update) Yöntemi:
  - object replace (p, o)
- Ek yöntemler de tanımlanabilir

Ağaç Türleri

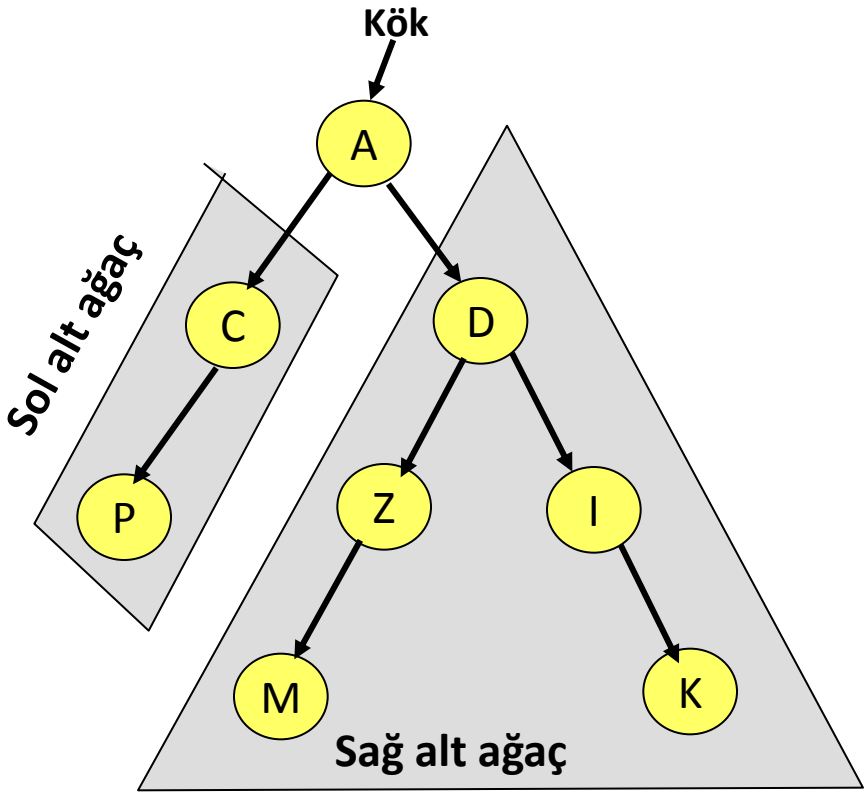
- En çok bilinen ağaç türleri ikili arama ağacı olup kodlama ağacı, sözlük ağacı, kümeleme ağacı gibi birçok ağaç uygulaması vardır.
  - **İkili Arama Ağacı (Binary Search Tree):**
    - İkili arama ağacında bir düğüm en fazla iki tane çocuğa sahip olabilir ve alt/çocuk bağlantıları belirli bir sırada 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.

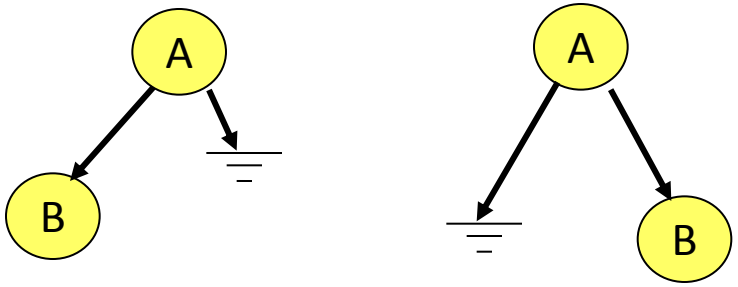
İkili Ağaç (Binary Tree) :

- 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.

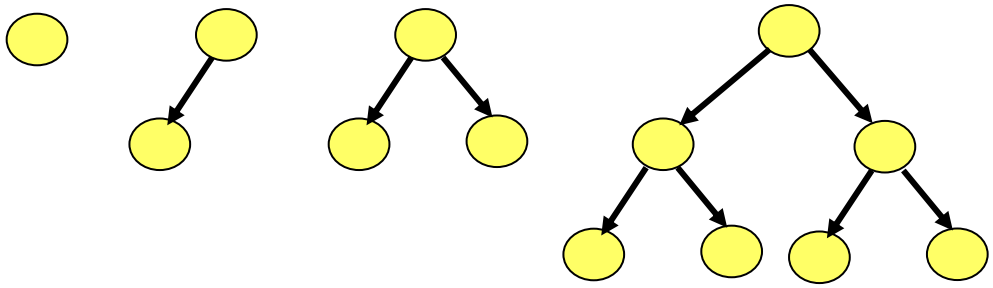




İki farklı ikili ağaç

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

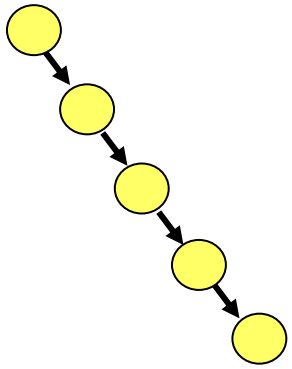
Herhangi bir d derinliğinde, N = ?

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

En küçük derinlik: **$\Theta(\log N)$**

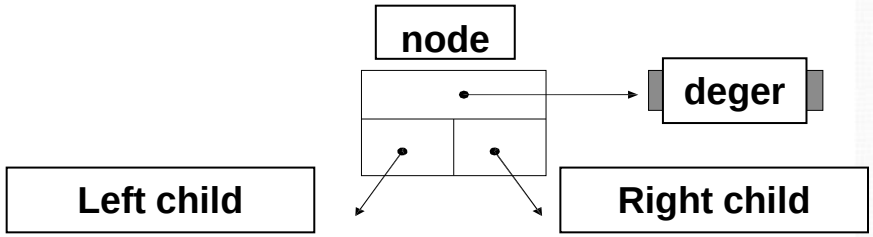
İkili Ağaç

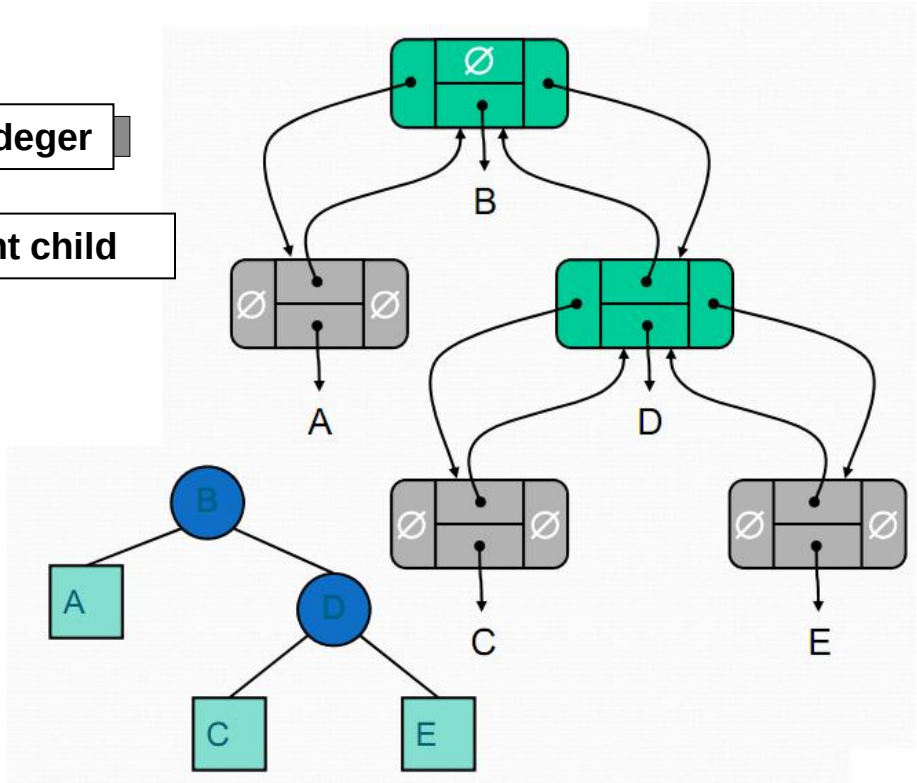
- N düğümlü ikili ağacın minimum derinliği: **Θ(log N)**
- İkili ağacın maksimum derinliği ne kadardır?
  - Dengesiz ağaç: Ağaç bir bağlantılı liste olursa!
  - Maksimum derinlik = N
- Amaç: Arama gibi operasyonlarda bağlantılı listeden daha iyi performans sağlamak için derinliğin log N de tutulması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



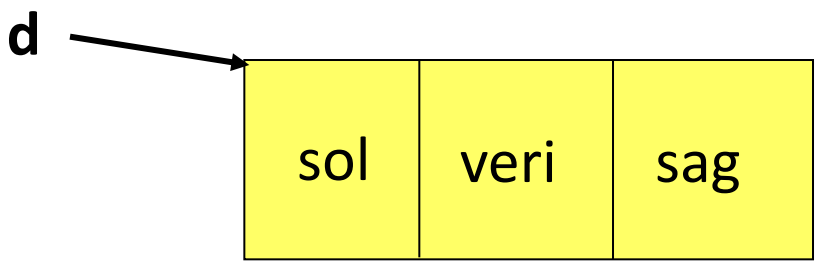
- Bağlantılı liste
- Derinlik = N

İkili Ağaç Bağlı Liste Yapısı

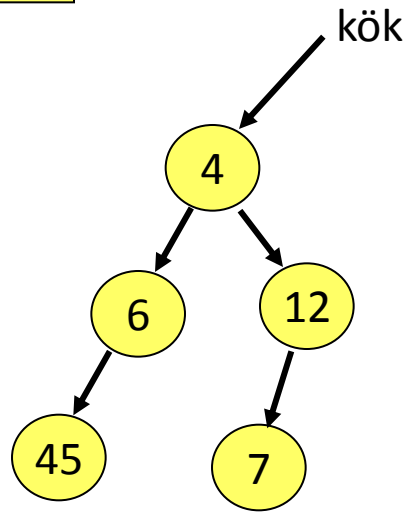




İkili Ağaç Gerçekleştirimi



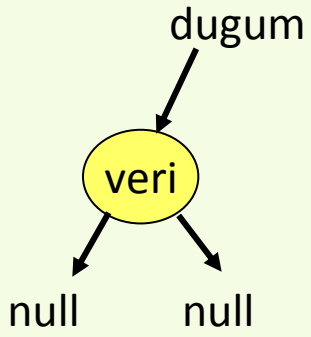

```
public class İkiliAgacDugumu {
    public İkiliAgacDugumu sol;
    public int veri;
    public İkiliAgacDugumu sag;
}
```



İkili Ağaç Gerçekleştirimi

- İkili ağaç için düğüm oluşturma. Her defasında sol ve sağı boş olan bir düğüm oluşturulur.

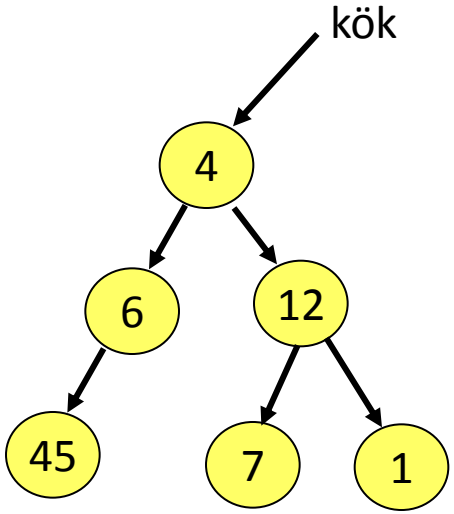
```
/* İkili ağaç düğümü oluşturur. */
  İkiliAgacDugumu DugumOlustur(int veri){
  İkiliAgacDugumu dugum = new İkiliAgacDugumu();
  dugum.veri = veri;
  dugum.sol = null;
  dugum.sag = null;
  return dugum;
}
```



İkili Ağaç Gerçekleştirimi

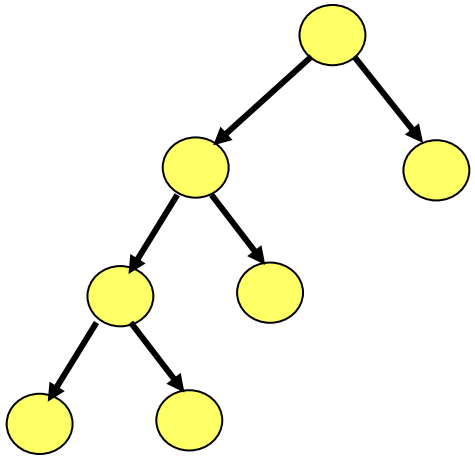
- İteratif düğüm oluşturma ve ağaca ekleme.

```
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 */
```

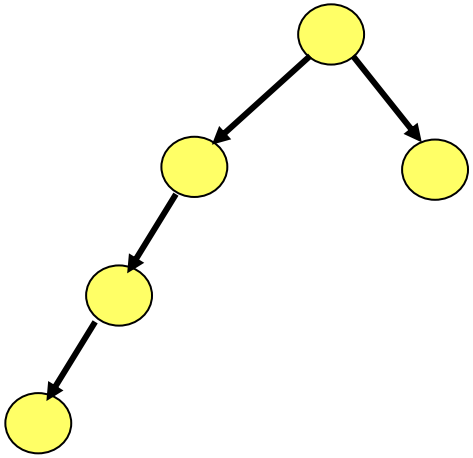


Proper (düzgün) Binary Trees

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



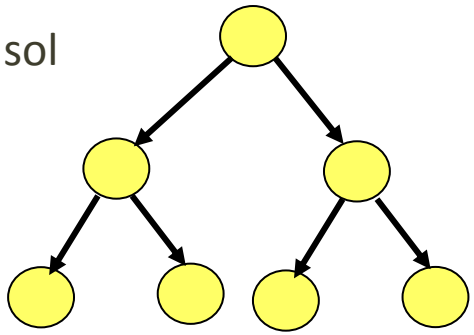
- Proper Tree



ImProper Tree

Full Binary Tree

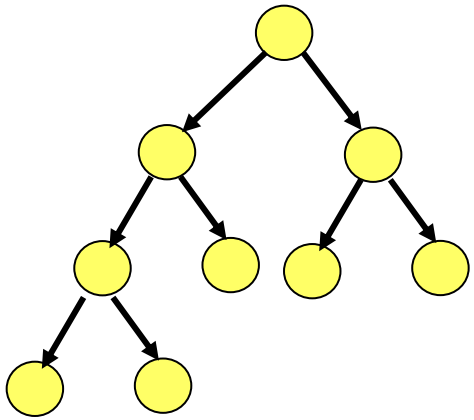
- Full binary tree:
  - 1- Her yaprağı aynı derinlikte olan
  - 2- Yaprak olmayan düğümlerin tümünün iki çocuğu olan ağaç Full (Strictly) Binary Tree'dir.
- Bir full binary tree'de n tane yaprak varsa bu ağaçta toplam 2n-1 düğüm vardır. Başka bir şekilde ifade edilirse,
  - Eğer T ağacı boş ise, T yüksekliği 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.



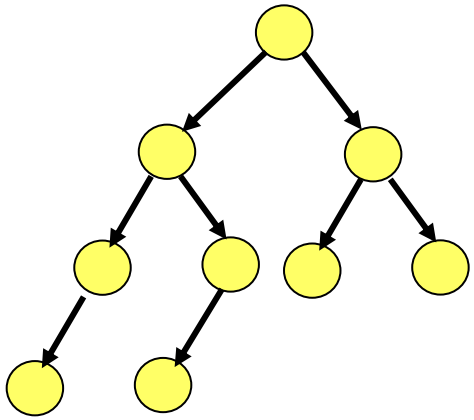
Complete Binary Tree

- 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.

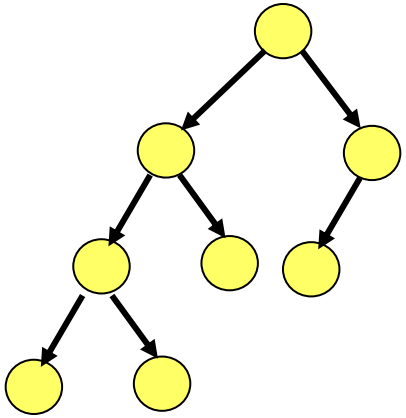
Complete Binary Tree



- Complete

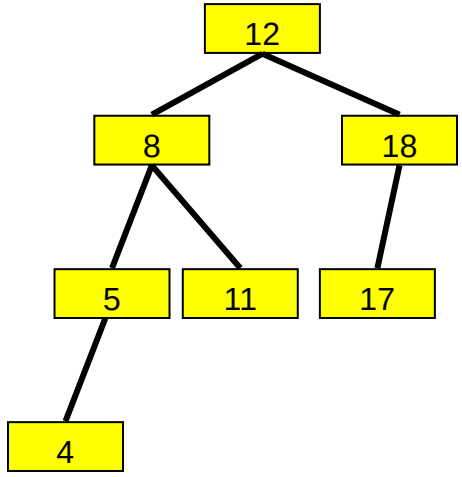


incomplete



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?)

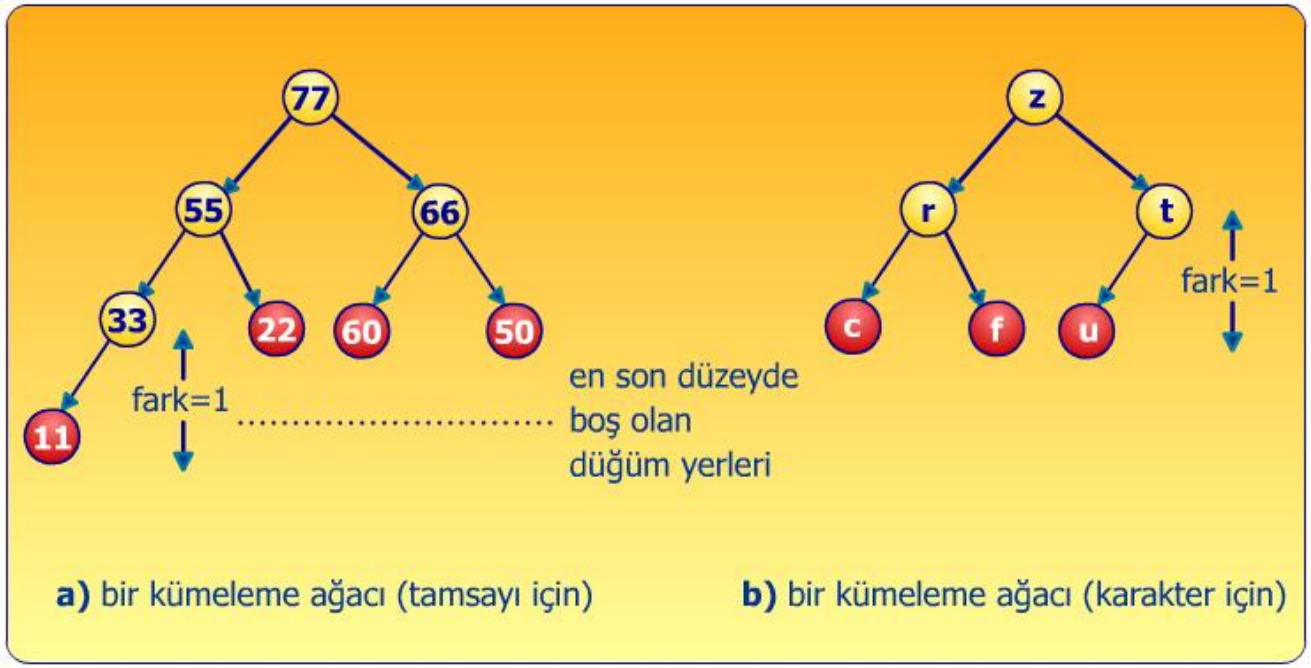


Heap Tree (Kümele Ağacı)

- 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.

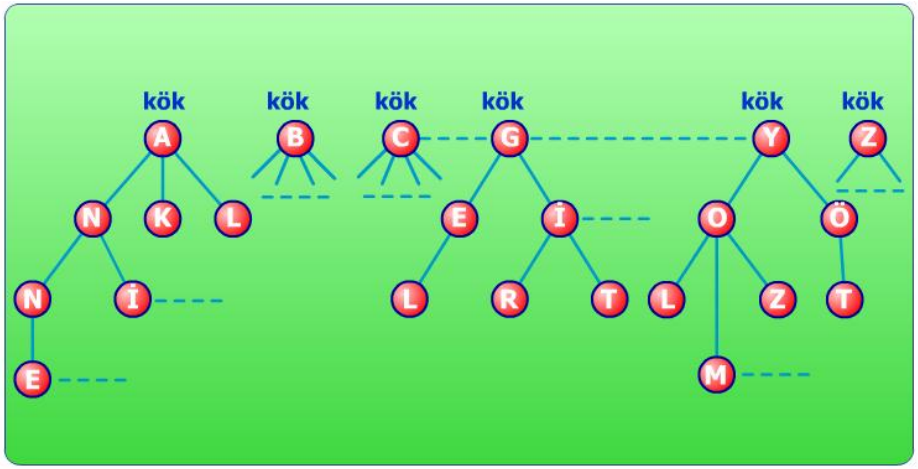
Heap Tree (Kümele Ağacı)

- 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.



Ödev

- Aşağıda verilen ağaç yapılarını araştırınız. Bu ağaçlara ait bilgileri Word ve Powerpoint ortamında hazırlayıp getiriniz.
  - Sözlük Ağaçları
  - Kodlama Ağaçları
  - Sıkıştırma Ağaçları
  - Bağıntı Ağaçları
  - Kümele Ağacı

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.

```
Algorithm postOrder(v)
    for each child w of v
        postOrder (w)
    visit(v)
```

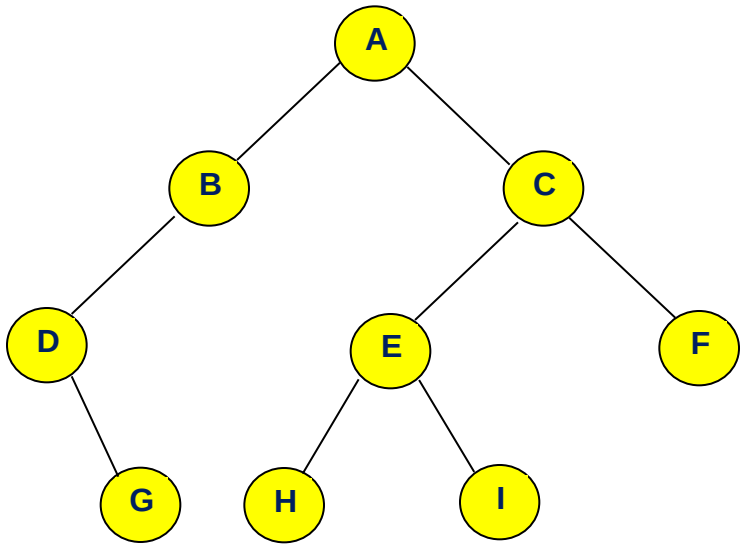
```
SonraKok(IkiliAgacDugumu kok)
{
    if (kok == null) return;
    SonraKok(kok.sol);
    SonraKok(kok.sag);
    System.out.print(kok.veri+" ");
}
```

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

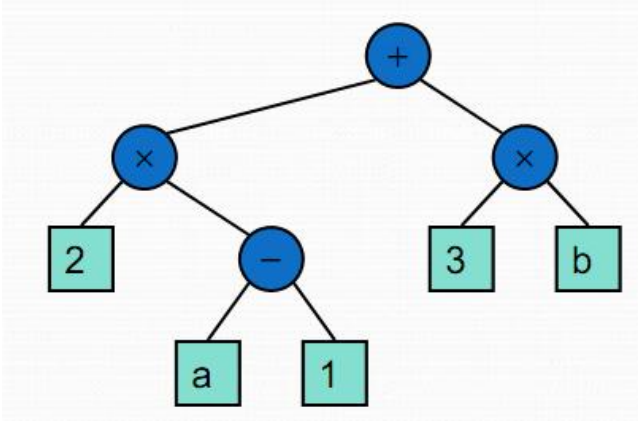
PostOrder : G D B H I E F C A

LevelOrder: A B C D E F G H I

Bağıntı (İfade) Ağaçları (Expression Tree)

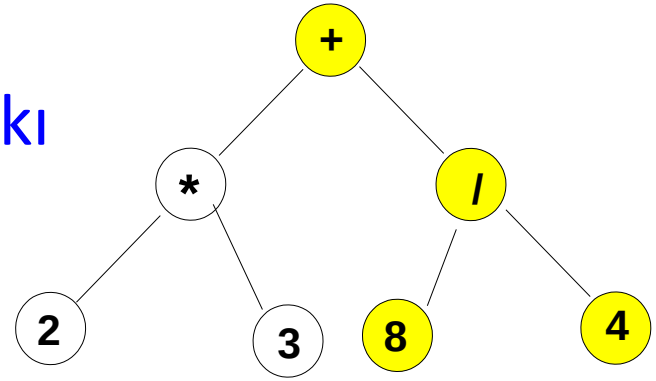
- Bağıntı ağaçları bir matematiksel bağıntının ağaç şeklinde tutulması için tanımlanmıştır. Bir aritmetik ifade ile ilişkili ikili ağaçtır.
- Ağacın genel yapısı:
  - Yaprak düğüm = değişken/sabit değer
  - Kök veya ara düğümler = operatörler
  - Birçok derleyicide kullanılır. Parantez gereksinimi yoktur.

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



Bağıntı Ağaçlarında Geçiş

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

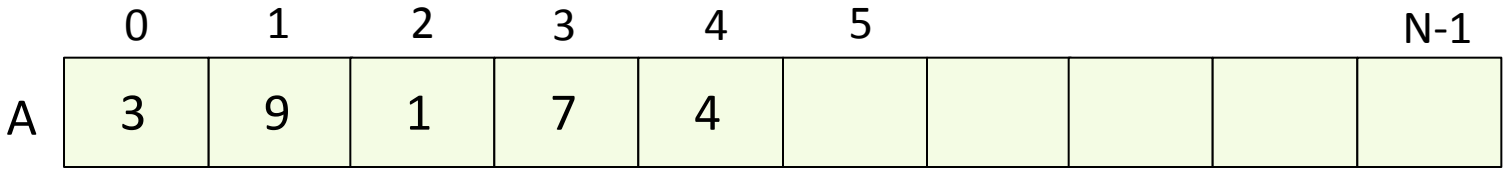


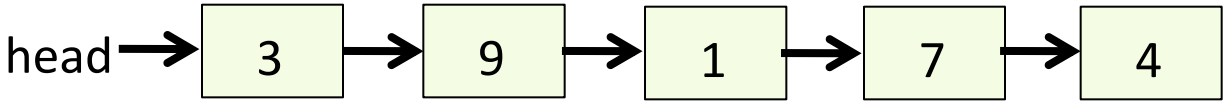
Arama Ağaçları

- Bir veri yapısı içerisine çok sayıda (anahtar, değer) çiftleri saklamak istediğimizi varsayalım.
- Aşağıdaki işlemleri etkili bir şekilde yerine getirebilecek bir veri yapısına ihtiyacımız var.
  - Ekle(anahtar, değer)
  - Sil(anahtar, değer)
  - Bul(anahtar)
  - Min()
  - Max()
- Alternatif veri yapıları?
  - Dizi kullanmak
  - Bağlantılı liste kullanmak

Arama Ağaçları

Örnek: Yandaki değerleri saklayalım: 3, 9, 1, 7, 4





Operasyon	Sırasız Dizi	Sıralı Dizi	Sırasız Liste	Sıralı List
Bul (Arama)	O(N)	O(logN)	O(N)	O(N)
Ekle	O(1)	O(N)	O(1)	O(N)
Sil	O(1)	O(N)	O(1)	O(1)

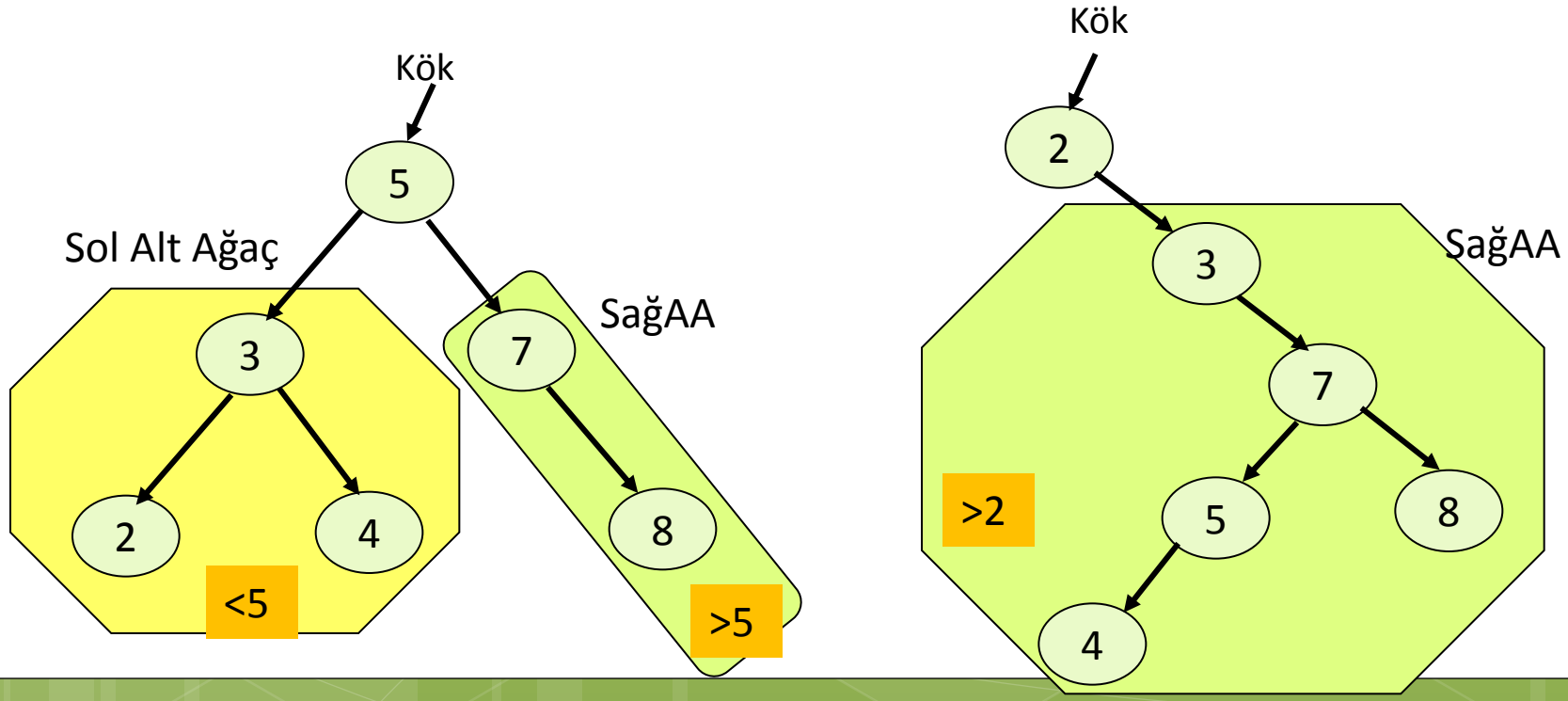
Bul/Ekle/Sil işlemlerinin hepsini O(logN) de yapabilir miyiz?

Kullanılan Verimli Arama Ağaçları

- Fikir: Verileri arama ağacı yapısına göre düzenlersek arama işlemi daha verimli olacaktır.
  - İkili Arama Ağacı (Binary search tree (BST))
  - AVL Ağacı
  - Splay Ağacı
  - 2-3-4 Ağacı
  - Red-Black Ağacı
  - B Ağacı ve B+ Ağacı

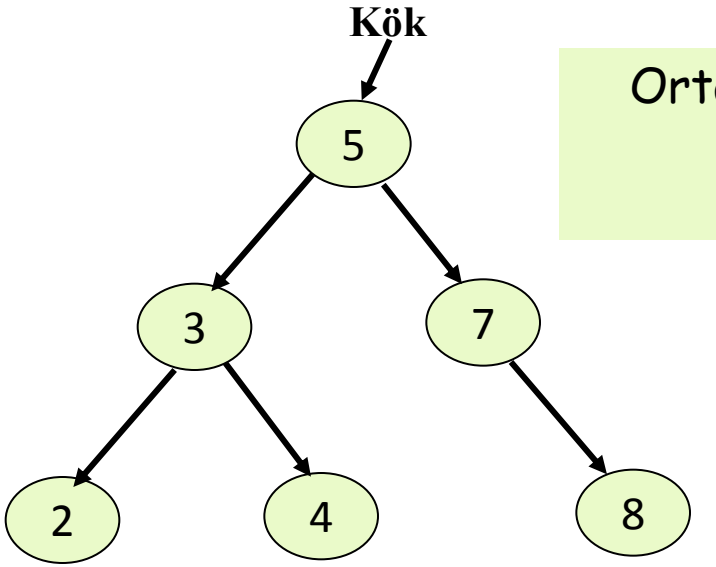
İkili Arama Ağacı (Binary Search Tree)

- İkili Arama Ağacı her bir düğümdeki değerlere göre düzenlenir:
  - Sol alt ağaçtaki tüm değerler kök düğümünden küçüktür.
  - Sağ alt ağaçtaki tüm değerler kök düğümünden büyüktür.



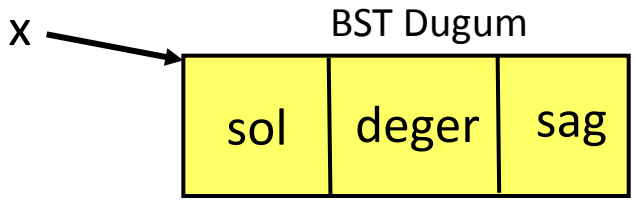
İkili Arama Ağacında Sıralama

- İkili arama ağacı önemli özelliklerinden birisi Ortada-kök (Inorder) dolaşma algoritması ile düğümlere sıralı bir şekilde ulaşılmasını sağlar.



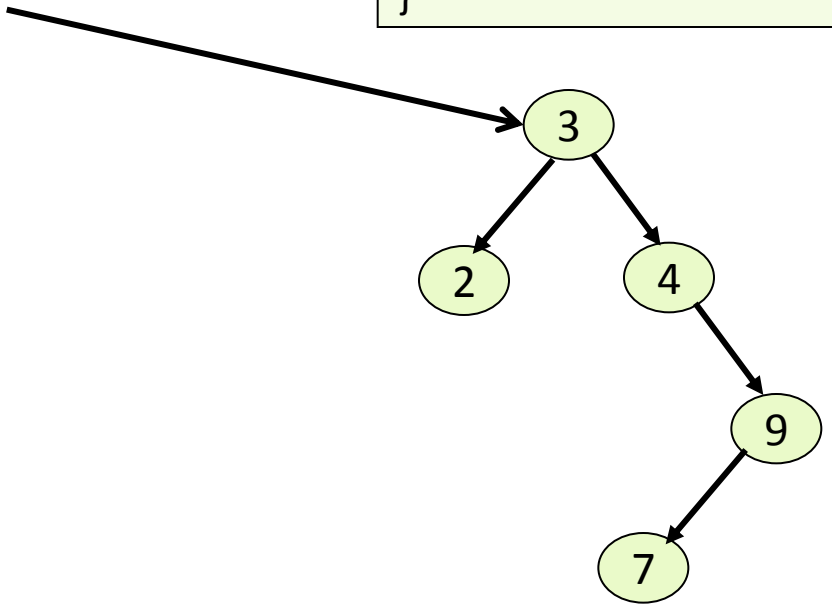
Ortada-kök sonucu
2 3 4 5 7 8

BST İşlemleri- Tanımlama




```
public class BSTDugum
{
  public BSTDugum sol;
  public int deger;
  public BSTDugum sag;
}
```

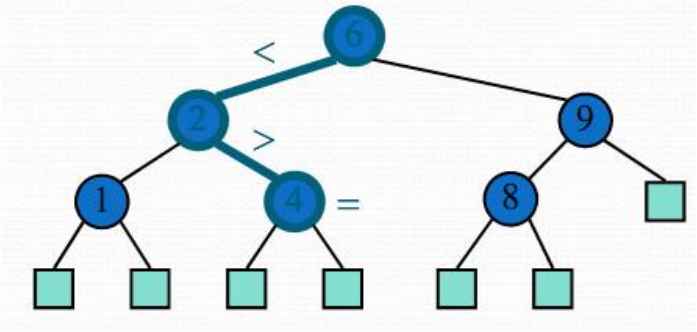
```
/* İKİLİ ARAMA AĞACI */
public class BST {
  Private BSTDugum kok;

  public BST(){kok=null;}
  public void Ekle(int deger);
  public void Sil(int deger);
  public BSTNode Bul(int key);
  public BSTNode Min();
  public BSTNode Max();
};
```



BST İşlemleri-Arama

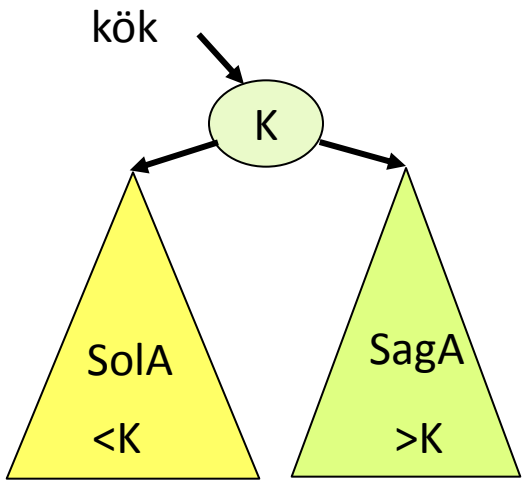
- Bir k anahtarını aramak için, kök düğümden başlayarak aşağı doğru bir yol izlenir.
- Bir sonraki ziyaret edilecek düğüm, k anahtar değerinin geçerli düğümün anahtar değeriyle karşılaştırılması sonucuna bağlıdır.
- Eğer yaprağa ulaşıldıysa anahtar bulunamamıştır ve null değer geri döndürülür.
- Örnek: Bul(4)

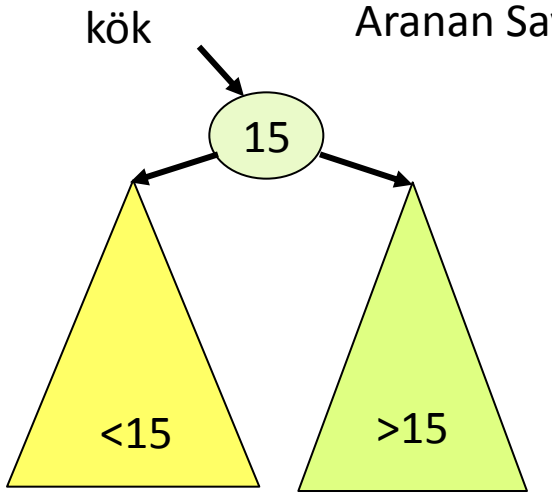
```
Algorithm TreeSearch(k, v)
    if T.isExternal (v)
        return v
    if k < key(v)
        return TreeSearch(k, T.left(v))
    else if k = key(v)
        return v
    else { k > key(v) }
        return TreeSearch(k, T.right(v))
```



BST İşlemleri- Arama

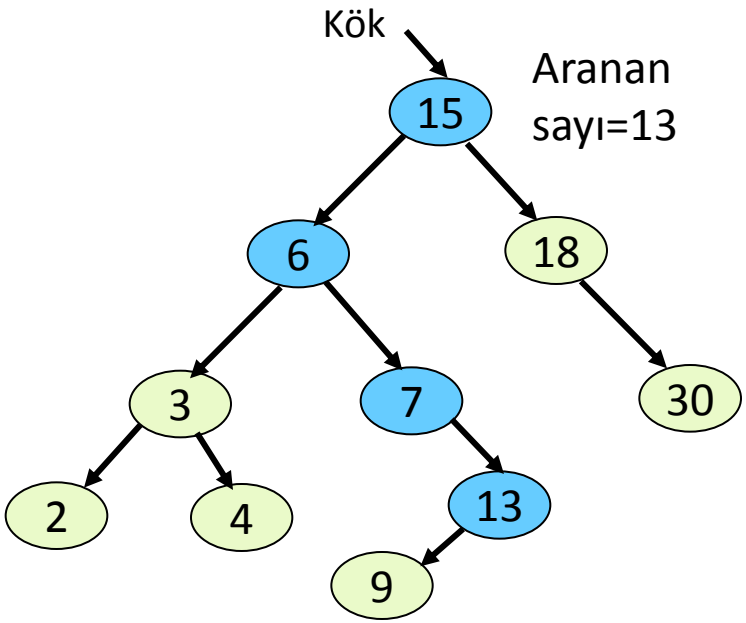
- Değeri içeren düğümü bul ve bu düğümü geri döndür.






```
1.  Arama işlemine kökten başla
2.  if (aranaDeger == kok.deger)      return kok;
3.  if (aranaDeger < kok.deger)       Ara SolAltAğaç
4.  else                              Ara SagAltAğaç
```

BST İşlemleri- Arama




```
public BSTDugum Bul(int deger)
{  return Bul2(kok, deger); }
```

```
public BSTDugum Bul2(BSTDugum kok, int deger)
{
  if (kok == null) return null;
  if (deger == kok.deger)
    return kok;
  else if (deger < kok.deger)
    return Bul2(kok.sol, deger);
  else /* deger > kok.deger */
    return Bul2(kok.sag, deger);
}
```

Mavi renkli düğümler arama sırasında ziyaret edilen düğümlerdir.

Algoritmanın çalışma karmaşıklığı O(d)'dir. (d = ağacın derinliği)

BST İşlemleri-Arama

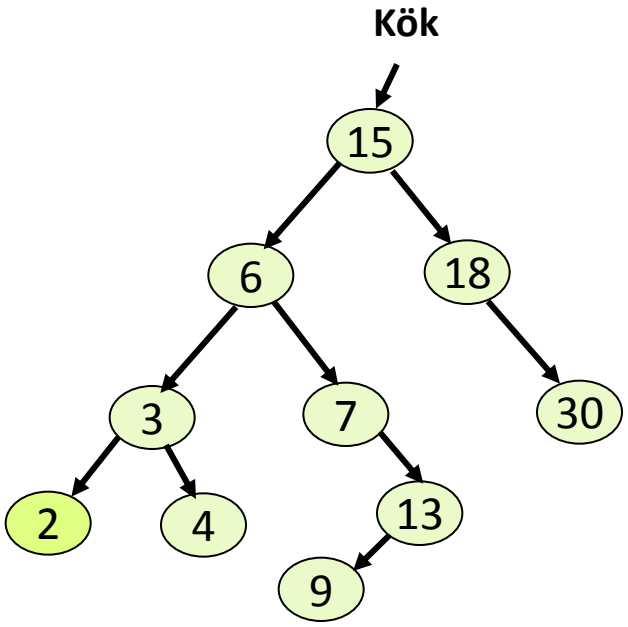
- Aynı algoritma while döngüsü yardımıyla yinelemeli şekilde yazılabilir. Yinelemeli versiyon özyinelemeli versiyona göre daha verimli çalışır.

```
public BSTDugum Bul(int deger){
  BSTDugum p = kok;
  while (p){
    if (deger == p.deger)      return p;
    else if (deger < p.deger)  p = p.sol;
    else /* deger > p.deger */  p = p.sag;
  } /* while-bitti */
  return null;
} //bul-Bitti
```

BST İşlemleri- Min

- 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.

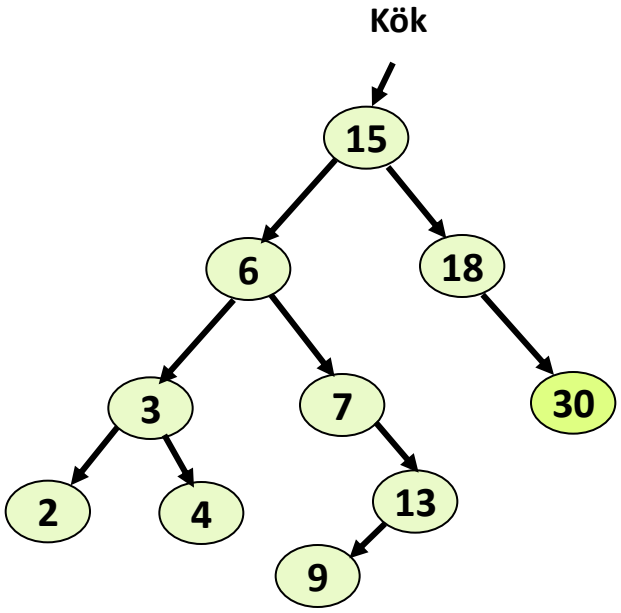
```
public BSTDugum Min()
{
  if (kok == null)
    return null;
  BSTDugum p = kok;
  while (p.sol != null){
    p = p.sol;
  }
  return p;
}
```



BST İşlemleri-Max

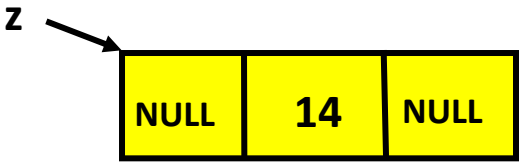
- Ağaçtaki en büyük elemanı içeren düğümü bulur ve geri döndürür.
  - En büyük elemanı içeren düğüm en sağdaki düğümde bulunur.
  - Kökten başlayarak devamlı sağa gidilerek bulunur.

```
public BSTDugum Max(){
  if (kok == null)
    return null;

  BSTDugum p = kok;
  while (p.sag != null){
    p = p.sag;
  }

  return p;
}
```

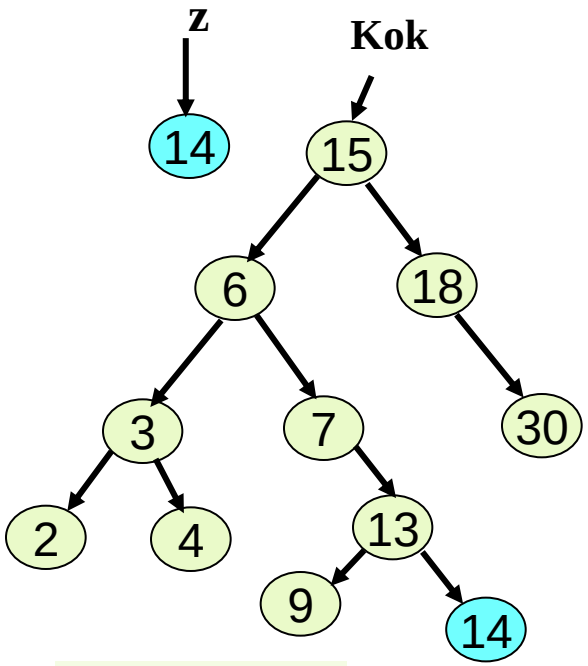


BST İşlemleri– Ekle(int deger)

- Eklenecek değeri içeren “z” isimli yeni bir düğüm oluştur.
- Ö.g.: Ekle 14
- Kökten başlayarak ağaç üzerinde eklenecek sayıyı arıyormuş gibi aşağıya doğru ilerle.
- Yeni düğüm aramanın bittiği düğümün çocuğu olmalıdır.



Eklenecek “z” düğümü.
z.deger = 14



Eklemeden sonra

BST İşlemleri- Ekle(int deger)

```
public void Ekle(int deger) {
  BSTDugum pp = null;  /* pp p'nin ailesi */
  BSTDugum p = kok;   /* Kökten başla ve aşağıya doğru ilerle*/
  while (p) {
    pp = p;
    if (deger == p.deger)      return;  /* Zaten var */
    else if (deger < p.deger)    p = p.sol;
    else /* deger > p.deger */ p = p.sag;
  }
 /* Yeni değeri kaydedeceğimiz düğüm */
  BSTDugum z = new BSTDugum();
  z.deger = deger; z.sol = z.sag = null;
  if (kok == null) kok = z; /* Boş ağaca ekleme */
  else if (deger < pp.deger) pp.sag = z;
  else pp.sol = z;
} // ekleme işlemi bitti.
```

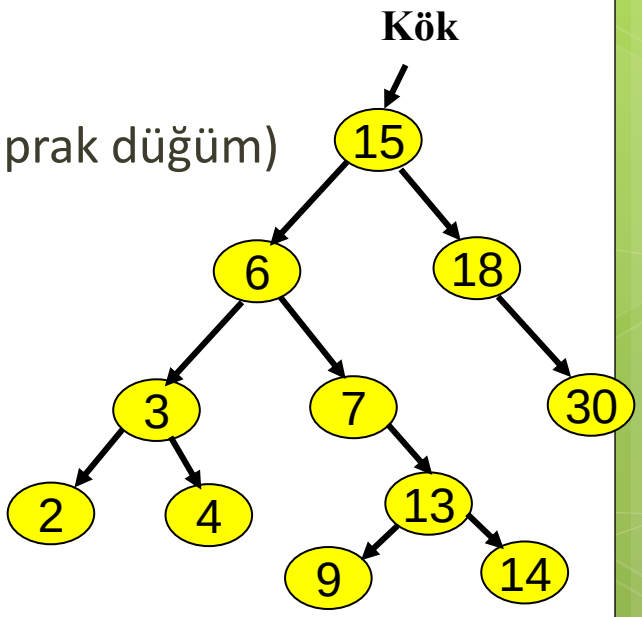
BST-Silme Kaba Kod

- while(silinecek düğümün ve ailesinin adresini bulana kadar) {
- q <- silinecek düğümün, qa<- ailesinin adresi;
- if(silinmek istenen bulunamadı ise)
  - yapacak birşey yok dön;
- if(silinecek düğümüm iki alt çocuğu da varsa)
  - sol alt ağacın en büyük değerli düğümünü bul;
  - (veya denge bozulmuş ise sağ alt ağacın enküçük değerli düğümünü bul)
  - bu düğümdeki bilgiyi silinmek istenen düğüme aktar;
  - bu aşamada en fazla bir çocuğu olan düğümü sil;
- silinen düğümün işgal ettiği bellek alanını serbest bırak;
  }

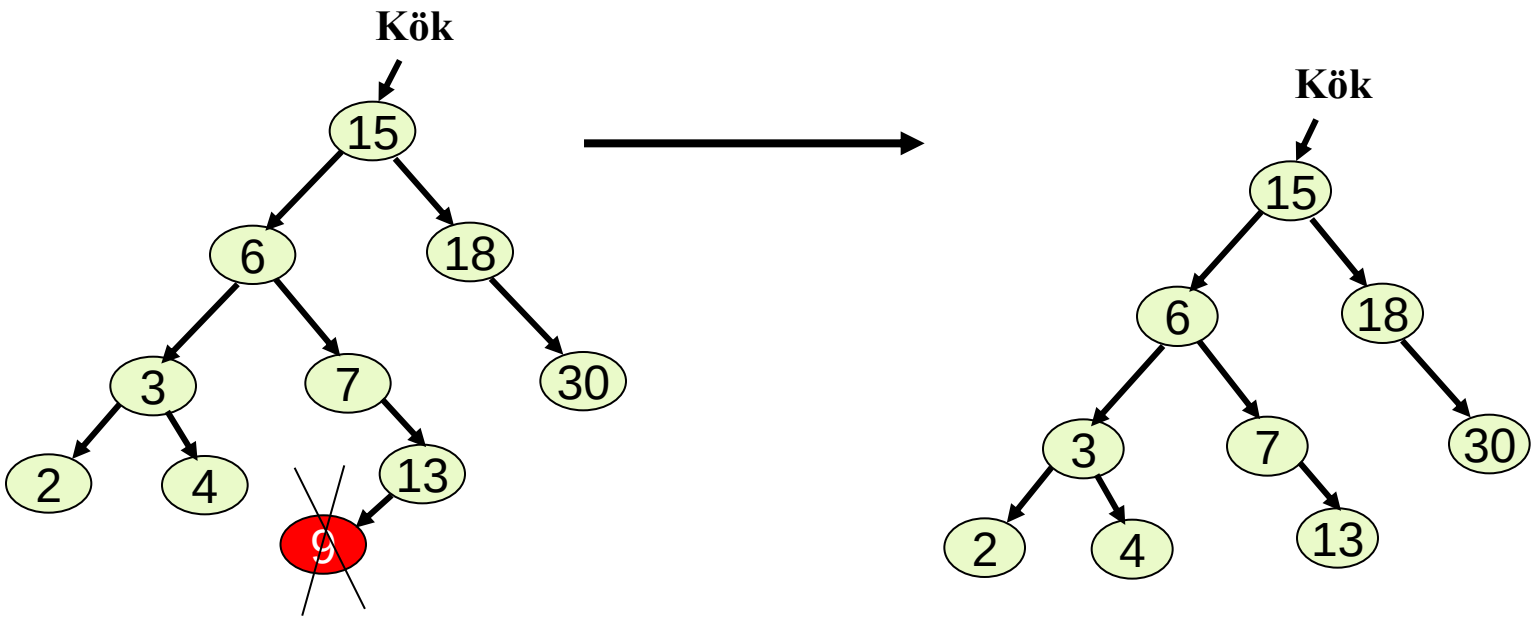
BST İşlemleri- Sil(int deger)

- Silme işlemi biraz karmaşıktır.
- 3 durum var:

1) Silinecek düğümün hiç çocuğu yoksa (yaprak düğüm)
   - Sil 9
2) Silinecek düğümün 1 çocuğu varsa
   - Sil 7
3) Silinecek düğümün 2 çocuğu varsa
   - Sil 6



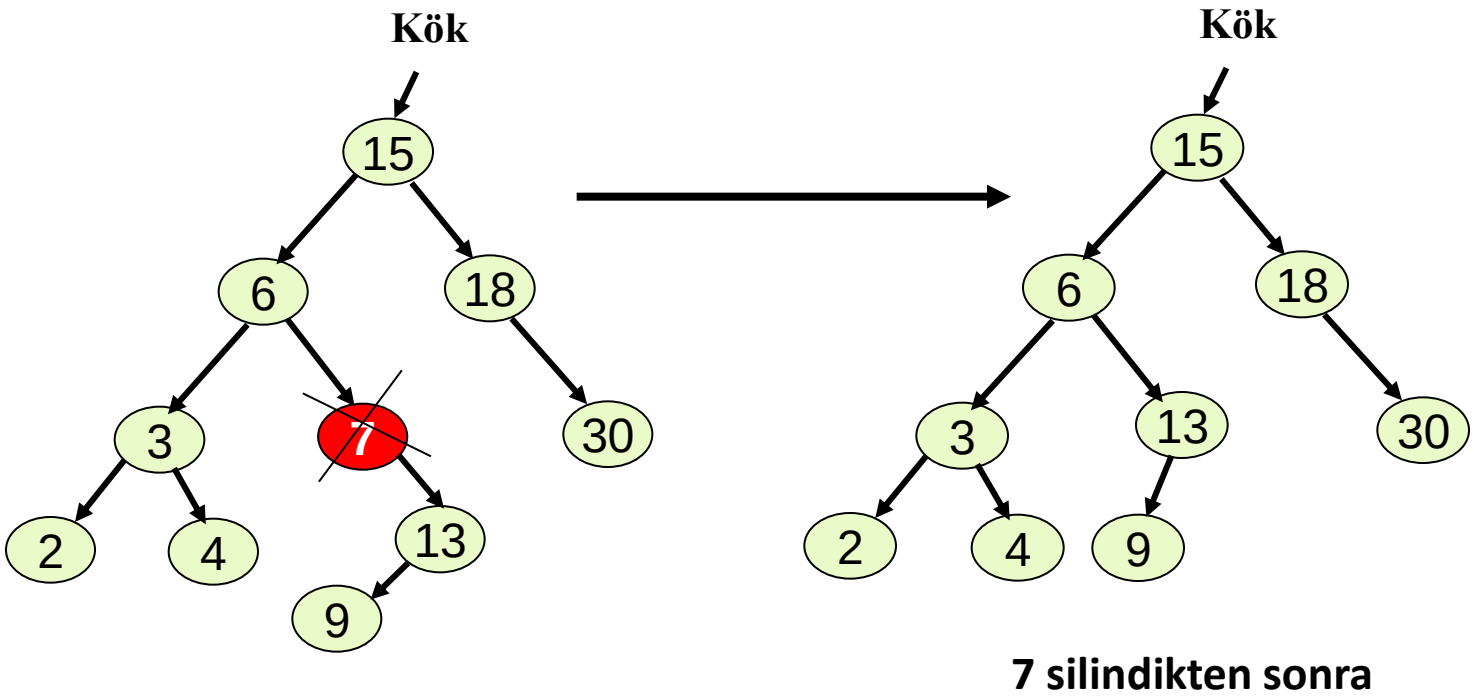
Silme: Durum 1 – Yaprak Düğümü Silme



Sil 9: Düğümü kaldırın ve bağlantı kısmını güncelleyin

9 silindikten sonra

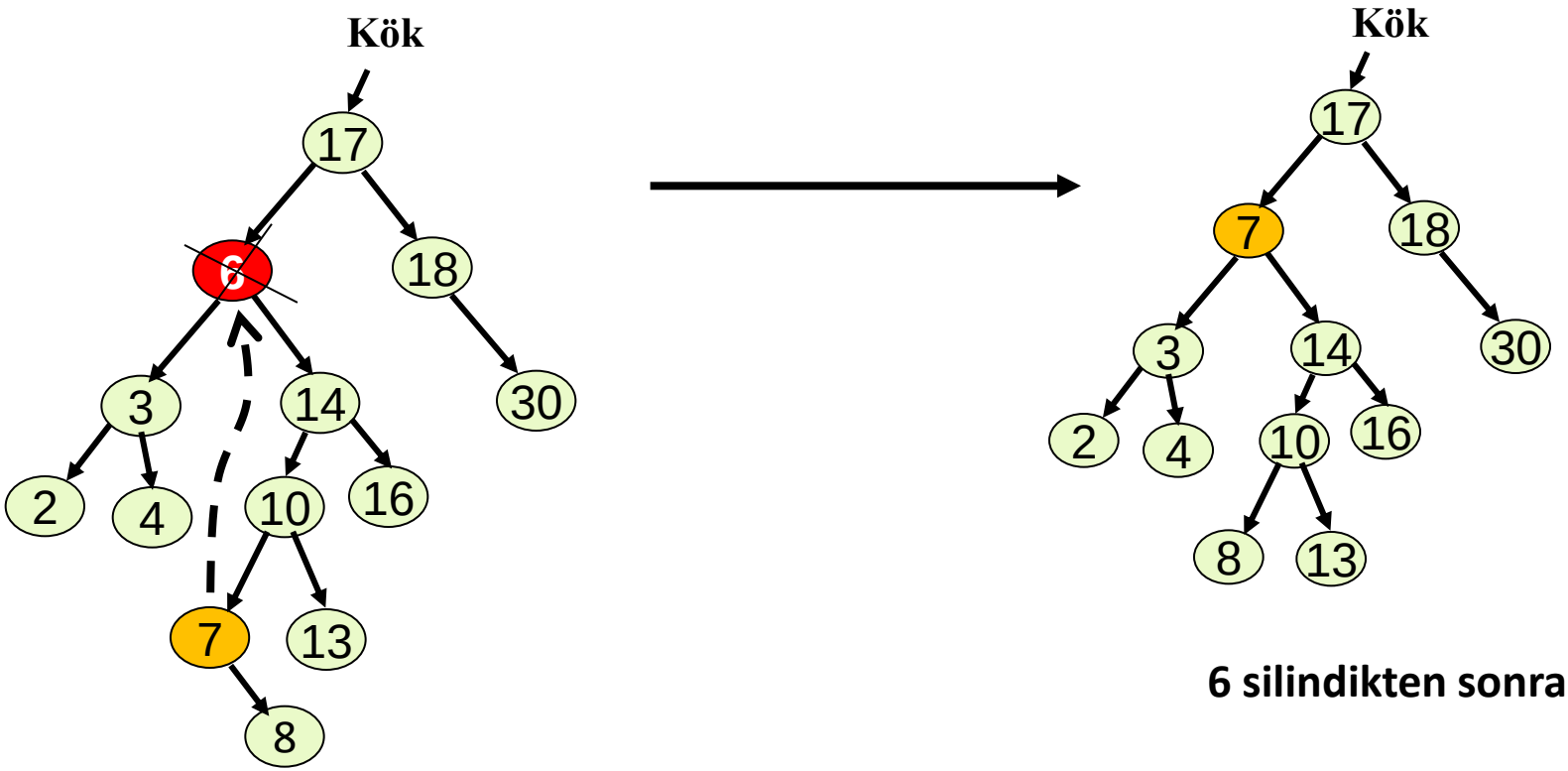
Silme: Durum 2 – 1 Çocuklu Düğüm



7 silindikten sonra

Sil 7: Silinecek düğümün ailesi ve çocuğu arasında bağ kurulur

Silme: Durum 3 – 2 Çocuklu Düğüm



6 silindikten sonra

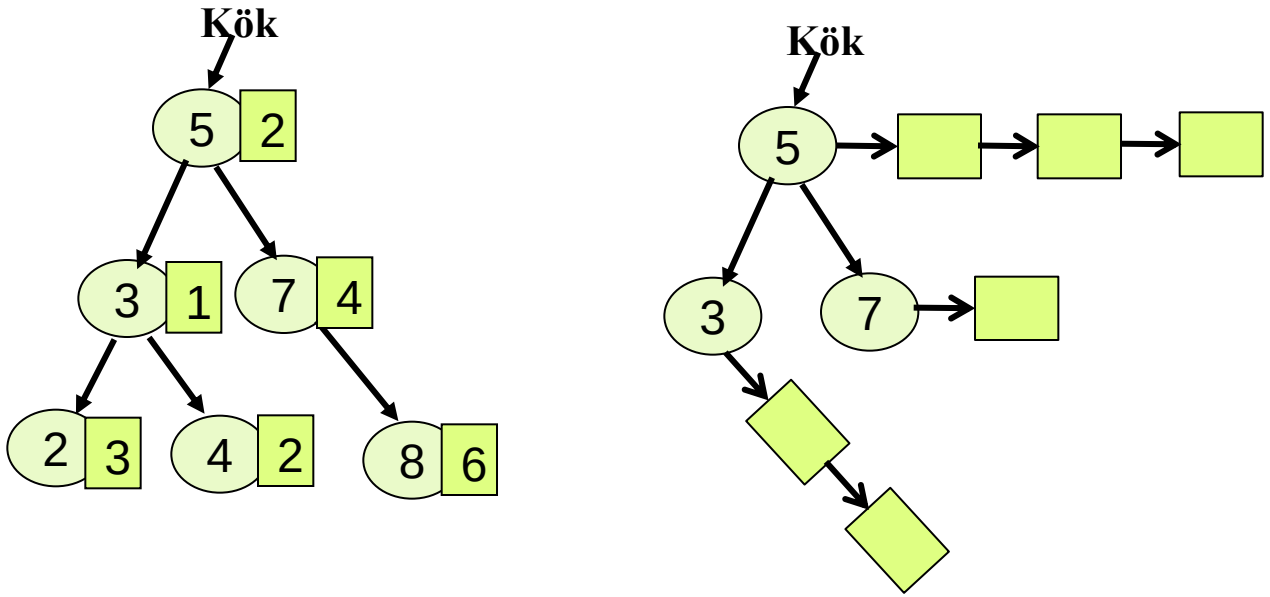
Sil 6:

1) **Sağ alt ağaçtaki en küçük eleman bulunur.(7)**
2) **Bu elemanın sol çocuğu olmayacaktır.**
3) **6 ve 7 içeren düğümlerin içeriklerini değiştirin**
4) **6 nolu eleman 1 çocuğu varmış gibi silinir.**

Not: Sağ alt ağaçtaki en küçük eleman yerine sol alt ağaçtaki en büyük eleman bulunarak aynı işlemler yapılabilir.

BST-Aynı Sayılarla Başa Çıkma

- Ağaç içerisindeki aynı sayılarla aşağıda verilen iki şeklide başa çıkılabilir:
  - Düğümde saklanan bir sayaç değişkeni ile
- veya
  - Düğümde kullanılan bağlantılı liste ile

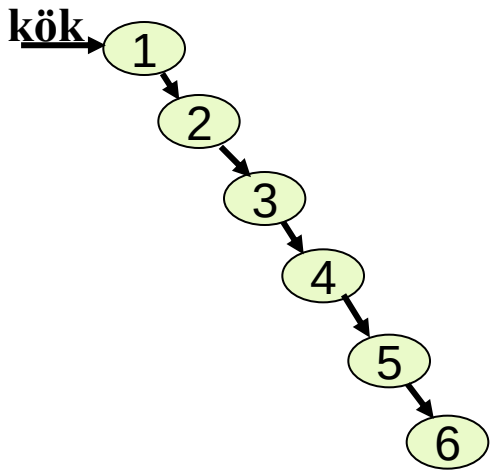


İkili Arama Ağacı Uygulamaları

- İkili arama ağacı harita, sözlük gibi birçok uygulamada kullanılır.
  - İkili arama ağacı (anahtar, değer) çifti şeklinde kullanılacak sistemler için uygundur.
    - Ö.g.: Şehir Bilgi Sistemi
      - Posta kodu veriliyor , şehir ismi döndürülüyor. (posta kodu/ Şehir ismi)
    - Ö.g.: telefon rehberi
      - İsim veriliyor telefon numarası veya adres döndürülüyor. (isim, Adres/Telefon)
    - Ö.g.: Sözlük
      - Kelime veriliyor anlamı döndürülüyor. (kelime, anlam)

İkili Arama Ağacı

- Bul, Min, Max, Ekle, Sil işlemlerinin karmaşıklığı O(d)
- Fakat d ağacın derinliğine bağlı.
- Örnek: 1,2,3,4,5,6 sayılarını sıralı bir şekilde ekleyelim.
- Ortaya çıkan ağaç bağlantılı listeye benzemektedir. Dolayısıyla karmaşıklık O(n) şeklinde olacaktır.



- Daha iyisi yapılabilir mi?
- Ağacımızı dengeli yaparsak evet
  1. AVL-ağaçları
  2. Splay ağaçları
  3. 2-3-4 Ağaçları
  4. Red-Black ağaçları
  5. B ağaçları, B+ agaçları

Ödev

- İkilik binary ağacında minimum ve maksimum değeri bulan metodları yazınız.
- İkilik binary ağacında lever-order dolaşmayı yapınız. (**Kuyruk yapısı kullanılacak**)
- İkili arama ağcını Oluşturma-Arama-Ekleme-Silme olaylarını bir bütün olarak gerçekleştiriniz. (Önerilen Ders kitabından yararlanabilirsiniz)

Örnek Programlar

İkili arama Ağacı–Java

```
class TreeNode  // Düğüm Sınıfı
 {
  public int data;
  public TreeNode leftChild;
  public TreeNode rightChild;
  public void displayNode()  {  System.out.print(" "+data+" ");  }
 }

// Ağaç Sınıfı
class Tree
{
  private TreeNode root;
  public Tree()   { root = null; }

  public TreeNode getRoot()  {  return root;  }
```

İkili arama Ağacı–Java

```
// Ağacın preOrder Dolaşılması
public void preOrder(TreeNode localRoot)
{
 if(localRoot!=null)    {
  localRoot.displayNode();
  preOrder(localRoot.leftChild);
  preOrder(localRoot.rightChild);
 }
}

// Ağacın inOrder Dolaşılması
public void inOrder(TreeNode localRoot)
{
 if(localRoot!=null)
 {
  inOrder(localRoot.leftChild);
  localRoot.displayNode();
  inOrder(localRoot.rightChild);
 }
}
```

İkili arama Ağacı–Java

```
// Ağacın postOrder Dolaşılması
public void postOrder(TreeNode localRoot)
{
  if(localRoot!=null)
  {
    postOrder(localRoot.leftChild);
    postOrder(localRoot.rightChild);
    localRoot.displayNode();
  }
}

// Ağaca bir düğüm eklemeyi sağlayan metot
public void insert(int newdata)
{
  TreeNode newNode = new TreeNode();
  newNode.data = newdata;

if(root==null)
    root = newNode;
```

İkili arama Ağacı–Java

```
else    {
    TreeNode current = root;      TreeNode parent;
    while(true)     {
     parent = current;
     if(newdata<current.data)
     {  current = current.leftChild;
      if(current==null)
      {
       parent.leftChild=newNode;
       return;
      }
     } else
     {
      current = current.rightChild;
      if(current==null)
      {          parent.rightChild=newNode;          return;        }
     }
    }  // end while
   } // end else not root
  } // end insert()
} // class Tree
```

İkili arama Ağacı–Java

```
// BinTree Test sınıfı
class BinTree
{
  public static void main(String args[])
  {
    Tree theTree = new Tree();

    // Ağaca 10 tane sayı yerleştirilmesi
    System.out.println("Sayılar : ");
    for (int i=0;i<10;++i) {
      int sayi = (int) (Math.random()*100);
      System.out.print(sayi+" ");
      theTree.insert(sayi);
    };

    System.out.print("\nAğacın InOrder Dolaşılması   : ");
    theTree.inOrder(theTree.getRoot());
    System.out.print("\nAğacın PreOrder Dolaşılması  : ");
    theTree.preOrder(theTree.getRoot());
    System.out.print("\nAğacın PostOrder Dolaşılması : ");
    theTree.postOrder(theTree.getRoot());
  }
}
```

İkili arama Ağacı– C++

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
/* Ağaca ait düğüm yapısı tanımlanıyor */
struct agacdugum {
  struct agacdugum *soldal;
  int data;
  struct agacdugum *sagdal;
};
// Düğüm yapısı için değişken tanımlarının yapıldığı kısım
typedef struct agacdugum AGACDUGUM;
typedef struct agacdugum * AGACDUGUMPTR;
```

İkili arama Ağacı– C++

```
// Ağaca düğüm eklemeyi saglayan fonksiyon yapisi
AGACDUGUMPTR dugumekle(AGACDUGUMPTR agacptr, int veri) {
/* her defasında tek dallı ağaç oluşturuluyor. Daha sonra ikili arama ağacındaki kurala göre sol veya sağ dala yerleştiriliyor. */

 if(agacptr==NULL)
  {
/*eger agaç işaretçisi boş ise ağaca eklenecek yeni düğüm için hafızada yer ayrılıyor*/

    agacptr =(agacdugum *) malloc(sizeof(agacdugum));
   if (agacptr!=NULL)
   {
  // Düğümler tek hücre, sağ ve sol dalları boş olarak oluşturuluyor
  //  printf("Ağaça veri eklendi\n ");
     agacptr->data = veri;
     agacptr->soldal = NULL;
     agacptr->sagdal= NULL;
   }
   else printf("%d eklenemedi. Bellek yetersiz.\n",veri);
  }
```

İkili arama Ağacı– C++

```
/*Gelen veri değeri daha önce girilen değerler ile karşılaştırılıp
uygun düğümün sol veya sağ dalına yerleştiriliyor.*/
  else

    if(veri<agacptr->data){ printf("Ağaçın soluna veri eklendi\n ");
      agacptr->soldal = dugumekle(agacptr->soldal,veri);}
    else

    if(veri>agacptr->data){printf("Ağaçın sağına veri eklendi\n ");
      agacptr->sagdal = dugumekle(agacptr->sagdal,veri);}

// eğer girilen değer ler daha önce var ise alınmıyor.
    else printf("Eşit olduğu için alınmadı\n ");
    return agacptr;
}
```

İkili arama Ağacı– C++

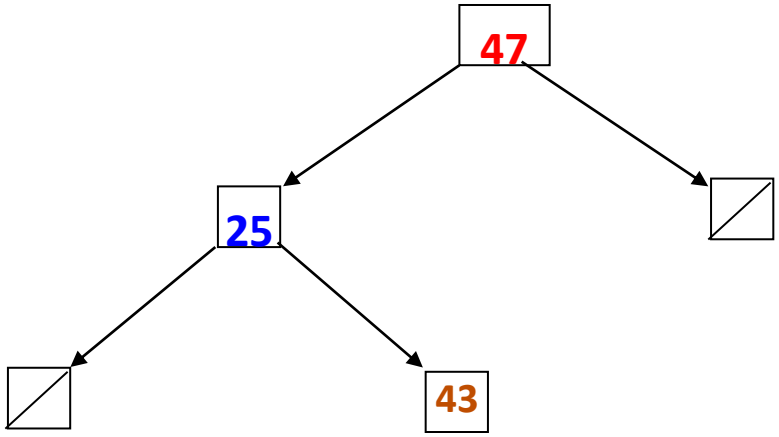
```
/* Agacın inorder dolaşılması */
void inorder(AGACDUGUMPTR agacptr) {
 if (agacptr != NULL) {
   inorder(agacptr->soldal);
   printf("%3d",agacptr->data);
   inorder(agacptr->sagdal);   }            }
/* Ağacın preorder dolaşılması */
void preorder(AGACDUGUMPTR agacptr) {
 if (agacptr != NULL) {
   printf("%3d",agacptr->data);
   preorder(agacptr->soldal);
   preorder(agacptr->sagdal);  }            }
/* Ağacın postorder dolaşılması */
void postorder(AGACDUGUMPTR agacptr) {
 if (agacptr != NULL) {
   postorder(agacptr->soldal);
   postorder(agacptr->sagdal);
   printf("%3d",agacptr->data); }            }
```

İkili arama Ağacı– C++

```
void main() {
    int i, dugum;
    AGACDUGUMPTR agacptr = NULL;
    for(i=0; i<12; ++i)
     { /* Ağaca yerleştirilecek sayılar */
       scanf("%d",&dugum);  printf("\n");
    // girilen değeri düğüm ekleme fonksiyonuna gönderiyoruz.
       agacptr = dugumekle(agacptr, dugum);
    }  printf("\n");

     printf("Ağacın preorder dolaşılması  :\n");
      preorder(agacptr); printf("\n");

      printf("Ağacın inorder dolaşılması   :\n");
      inorder(agacptr); printf("\n");

      printf("Ağacın postorder dolaşılması :\n");
      postorder(agacptr); printf("\n");
    }
```

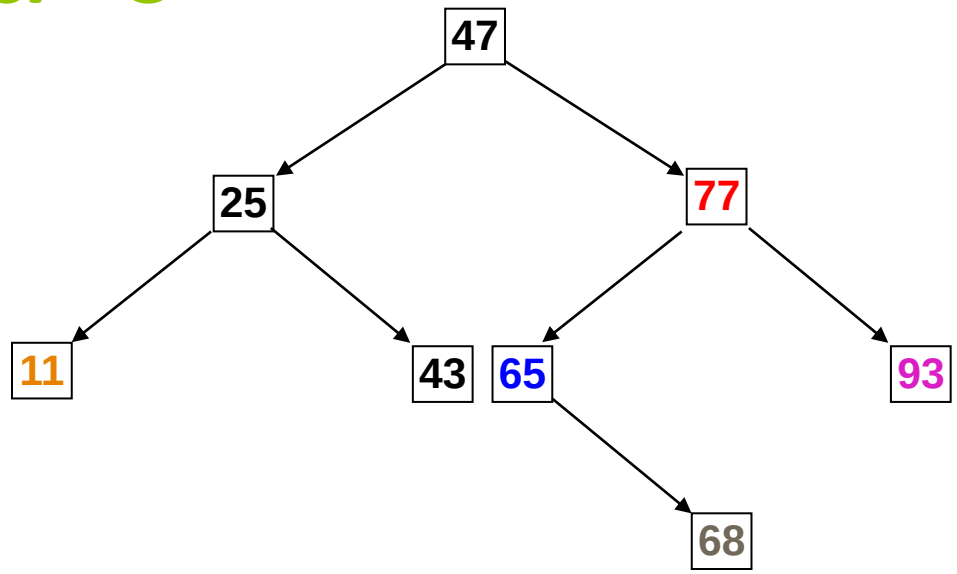
İkili arama Ağacı– C++

- 47
- 25
- Ağaçın soluna veri eklendi
- 43
- Ağaçın soluna veri eklendi
- Ağaçın sağına veri eklendi



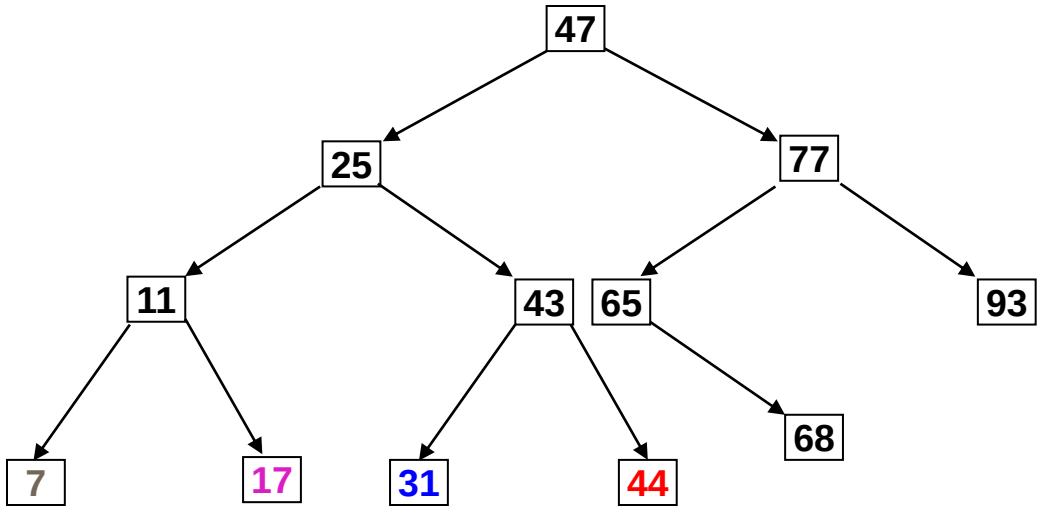
İkili arama Ağacı– C++

- 77
- Ağaçın sağına veri eklendi
- 65
- Ağaçın sağına veri eklendi
- Ağaçın soluna veri eklendi
- 68
- Ağaçın sağına veri eklendi
- Ağaçın soluna veri eklendi
- Ağaçın sağına veri eklendi
- 93
- Ağaçın sağına veri eklendi
- Ağaçın sağına veri eklendi
- 11
- Ağaçın soluna veri eklendi
- Ağaçın soluna veri eklendi



İkili arama Ağacı– C++

- 17
- Ağaçın soluna veri eklendi
- Ağaçın soluna veri eklendi
- Ağaçın sağına veri eklendi
- 44
- Ağaçın soluna veri eklendi
- Ağaçın sağına veri eklendi
- Ağaçın sağına veri eklendi
- 31
- Ağaçın soluna veri eklendi
- Ağaçın sağına veri eklendi
- Ağaçın soluna veri eklendi
- 7



İkili arama Ağacı– C++, Ekleme, Silme, Dolaşma

```
#include <stdio.h>
#include <stdlib.h>
struct tnode {
 int data;
 struct tnode *lchild, *rchild; };
/* Dugum degeri verilen dugumun kok pointer degerini elde eden bir foksiyondur*/
struct tnode *getptr(struct tnode *p, int key, struct tnode **y) {
printf("kok pointer\n");//silme anında devreye giriyor
 struct tnode *temp;
   if( p == NULL)      return(NULL);
   temp = p;    *y = NULL;
    while( temp != NULL)     {
      if(temp->data == key)   return(temp);
      else     {
         *y = temp; /* bu pointer root (kok) olarak depolanir */
   if(temp->data > key)
     temp = temp->lchild;
   else
     temp = temp->rchild;      }     }
    return(NULL);  }
```

İkili arama Ağacı– C++, Ekleme, Silme, Dolaşma

```
/* Veri degeri verilen dugumu silmek icin bir fonksiyon */
struct tnode *delete1(struct tnode *p,int val)  {
   struct tnode *x, *y, *temp;
   x = getptr(p,val,&y);
   if( x == NULL)   {     printf("Dugum mevcut degil\n");     return(p);    }
   else
   {
   /* bu kod kok (root) dugumu silmek icindir*/
   if( x == p)     {
     printf("kok dugum siliniyor\n");
      temp = x->lchild;
      y = x->rchild;
      p = temp;
      while(temp->rchild != NULL)     temp = temp->rchild;
    temp->rchild=y;
    free(x);
    return(p);
   }
```

İkili arama Ağacı– C++, Ekleme, Silme, Dolaşma

```
/* bu kod dugumun sahip oldugu cocukları silmek icindir.*/
 if( x->lchild != NULL && x->rchild != NULL)    {
      if(y->lchild == x)      {
temp = x->lchild;
y->lchild = x->lchild;
while(temp->rchild != NULL)    temp = temp->rchild;

temp->rchild=x->rchild;
x->lchild=NULL;
x->rchild=NULL;
      }
      else       {
        temp = x->rchild;
        y->rchild = x->rchild;
         while(temp->lchild != NULL)
     temp = temp->lchild;
   temp->lchild=x->lchild;
x->lchild=NULL;
 x->rchild=NULL;
       }
free(x);
   return(p);   }
```

İkili arama Ağacı– C++, Ekleme, Silme, Dolaşma

```
/* bu kod cocukları ile birlikte bir dugum siliyor*/
  if(x->lchild == NULL && x->rchild !=NULL)
    {
      if(y->lchild == x)
   y->lchild = x->rchild;
        else
          y->rchild = x->rchild;
        x->rchild=NULL;
        free(x);
        return(p);
      }
      if( x->lchild != NULL && x->rchild == NULL)
       {
        if(y->lchild == x)
          y->lchild = x->lchild ;
        else
          y->rchild = x->lchild;
        x->lchild = NULL;
        free(x);
        return(p);
            }
```

İkili arama Ağacı– C++, Ekleme, Silme, Dolaşma

```
/* bu kod cocukları olmadan bir dugumu siliyor*/
      if(x->lchild == NULL && x->rchild == NULL)
       {
         if(y->lchild == x)
           y->lchild = NULL ;
         else
           y->rchild = NULL;
         free(x);
         return(p);
       }
   }
}
```

İkili arama Ağacı– C++, Ekleme, Silme, Dolaşma

```
/*inorder binary agacını tekrarlamalı olarak yazdıran bir fonksiyon*/
void inorder1(struct tnode *p) {
 struct tnode *stack[100]; //yigin
 int top;
 top = -1;
if(p != NULL)   {
   top++;
   stack[top] = p;
   p = p->lchild;
   while(top >= 0)       {
        while ( p!= NULL)/* sol cocuk yigindan (stack dizisinden) cikariliyor*/
         {     top++;
              stack[top] =p;
              p = p->lchild;   }
    p = stack[top];
    top--;
    printf("%d\t",p->data);
    p = p->rchild;
    if ( p != NULL) /* sag cocuk cikariliyor*/
      {     top++;
      stack[top] = p;
         p = p->lchild;    }   }    }   }
```

İkili arama Ağacı– C++, Ekleme, Silme, Dolaşma

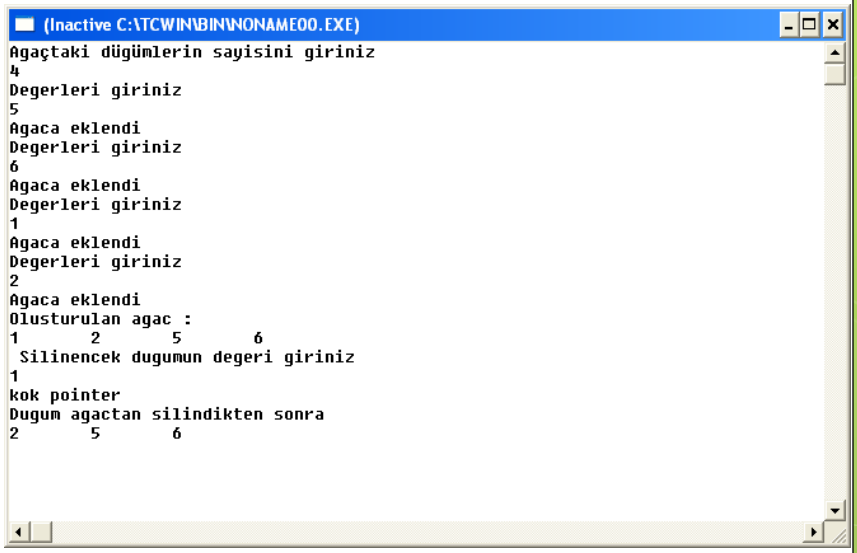
```
/* Olusturulan agaca yeni bir dugum ilave etmek icin olusturulan bir fonksiyon*/

 struct tnode *insert(struct tnode *p,int val)  {
  printf("Agaca eklendi\n");
    struct tnode *temp1,*temp2;
    if(p == NULL)     {
      p = (struct tnode *) malloc(sizeof(struct tnode)); /* koke bir dugum ilave ediliyor*/
      if(p == NULL)         {
    printf("Erisim izni yok\n");
    exit(0);         }
      p->data = val;
      p->lchild=p->rchild=NULL;  }
  else
 {
    temp1 = p;
  /* child (cocuk) olacak dugumun pointer(gostergesini) almak icin agacta dolasma*/
 while(temp1 != NULL)
  {
   temp2 = temp1;
   if( temp1 ->data > val)      temp1 = temp1->lchild;
   else     temp1 = temp1->rchild;
  }
```

İkili arama Ağacı– C++, Ekleme, Silme, Dolaşma

```
if( temp2->data > val)   {
    temp2->lchild = (struct tnode*)malloc(sizeof(struct tnode));/*Sol cocuk (left child) olarak yeni olusturulan dugumu ekler*/
    temp2 = temp2->lchild;
    if(temp2 == NULL)         {
    printf("Erisim izni yok\n");
    exit(0);         }
   temp2->data = val;
   temp2->lchild=temp2->rchild = NULL;   }
  else  {
     temp2->rchild = (struct tnode*)malloc(sizeof(struct tnode));/*Sag cocuk (right child) olarak yeni olusturulan dugumu ekler*/
    temp2 = temp2->rchild;
    if(temp2 == NULL)         {
    printf("Erisim izni yok\n");
    exit(0);        }
   temp2->data = val;
   temp2->lchild=temp2->rchild = NULL;   }   }
  return(p);
  }
```

İkili arama Ağacı– C++, Ekleme, Silme, Dolaşma

```
void main()
 {
   struct tnode *root = NULL;
   int n,x;
   printf("Agaçtaki dügümlerin sayisini giriniz\n");
   scanf("%d",&n);
   while( n-->0)
     {
       printf("Degerleri giriniz\n");
       scanf("%d",&x);
       root = insert(root,x);
     }
    printf("Olusturulan agac :\n");
    inorder1(root);
    printf("\n Silinencek dugumun degeri giriniz\n");
    scanf("%d",&n);
    root=delete1(root,n);
    printf("Dugum agactan silindikten sonra \n");
    inorder1(root);
}
```



İkili arama Ağacı– C# (1. Örnek)

- İkilik Arama Ağacı Oluşturmak
- class bstNodeC
- {
- public int sayi;
- public bstNodeC leftNode, rightNode;
- public bstNodeC(int sayi, bstNodeC leftNode, bstNodeC rightNode)
- {
- this.sayi = sayi;
- this.leftNode = leftNode;
- this.rightNode = rightNode;
- }
- }

İkili arama Ağacı– C# (1. Örnek)

```
private void preOrder(bstNodeC node)
{
listBox1.Items.Add(node.sayi);
if (node.leftNode != null) preOrder(node.leftNode);
if (node.rightNode != null) preOrder(node.rightNode);
}
private void inOrder(bstNodeC node)
{
if (node.leftNode != null) inOrder(node.leftNode);
listBox1.Items.Add(node.sayi);
if (node.rightNode != null) inOrder(node.rightNode);
}
private void postOrder(bstNodeC node)
{
if (node.leftNode != null) postOrder(node.leftNode);
if (node.rightNode != null) postOrder(node.rightNode);
listBox1.Items.Add(node.sayi);
}
```

İkili arama Ağacı– C# (1. Örnek)

```
//Boşaltma (makeEmpty)
private void makeEmpty(bstNodeC node)
{
if (node.leftNode node.leftNode != null)  makeEmpty(node.leftNode );
if (node.rightNode node.rightNode != null) makeEmpty(node.rightNode);
node = null;
}
//Arama (find)
private void find(int sayi, bstNodeC node)
{
if (node.sayi == 0) textBox2.Text = "Boş ağaç";
else if ((sayi < node.sayi)&&(node.leftNode!=null))
find(sayi, node.leftNode);
else if ((sayi > node.sayi)&&(node.rightNode!=null))
find(sayi, node.rightNode);
else if (sayi == node.sayi) textBox2.Text = “Sayı bulundu";
else textBox2.Text = “Sayı yok";
}
```

İkili arama Ağacı– C# (1. Örnek)

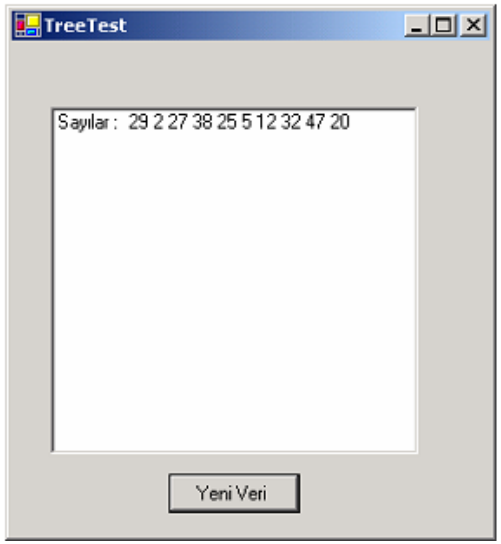
```
//Ekleme (append)
private void append(int sayi, bstNodeC node) {
bstNodeC yeniNode = new bstNodeC(sayi, null, null);
if (node.sayi == 0) node.sayi = sayi;
else  {
bstNodeC current = node; bstNodeC parent;
while(true)
{   parent = current;
if(sayi < current.sayi)
{ current = current.leftNode;
if(current == null)  {parent.leftNode = yeniNode;break; }
}
else
{ current = current.rightNode;
if(current==null) { parent.rightNode = yeniNode; return; }
}
}
}
}
```

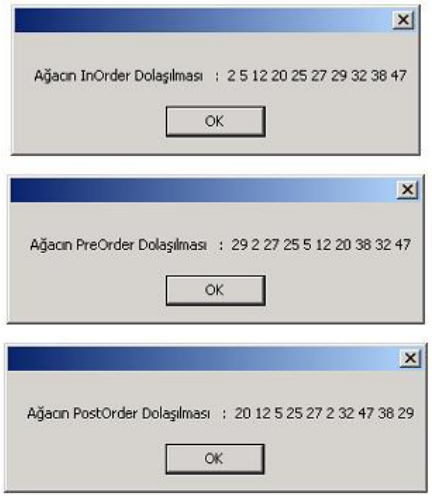
İkili arama Ağacı– C# (1. Örnek)

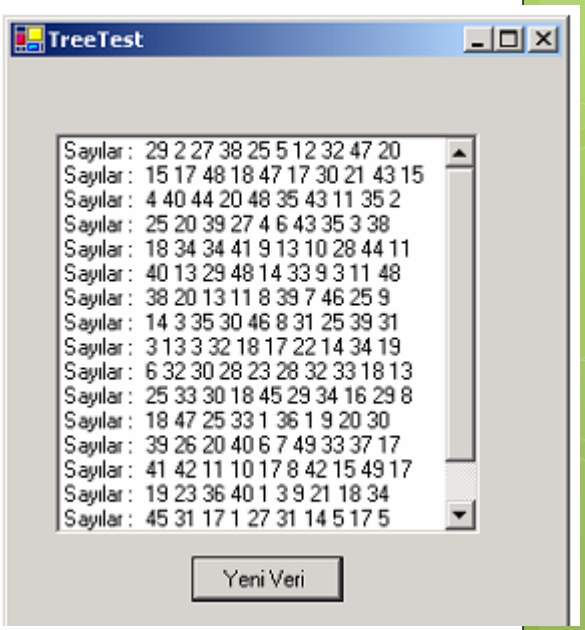
```
//Ekleme (append)
private void append(int sayi, bstNodeC node) {
bstNodeC yeniNode = new bstNodeC(sayi, null, null);
if (node.sayi == 0) node.sayi = sayi;
else  {
bstNodeC current = node; bstNodeC parent;
while(true)
{   parent = current;
if(sayi < current.sayi)
{ current = current.leftNode;
if(current == null)  {parent.leftNode = yeniNode;break; }
}
else
{ current = current.rightNode;
if(current==null) { parent.rightNode = yeniNode; return; }
}
}
}
}
```

İkili arama Ağacı– C# (2. Örnek)

- “TreeTest” adlı programın formu üzerinde 1 adet liste kutusu ve 1 adet düğme bulunmaktadır. “Yeni Veri” düğmesine basıldıkça, 1 ile 50 arasında 10 tane rastgele sayı üretmektedir ve arka arkaya gelen üç mesaj penceresi ile, InOrder, PreOrder ve PostOrder dolaşmalarda ikili ağaç üzerinde hangi düğüm sırasının izleneceğini göstermektedir.







İkili arama Ağacı– C# (2. Örnek)

```
public class GLOBAL   {      public static string tempStr;   }
class TreeNode
{   public int data;     public TreeNode leftChild;     public TreeNode rightChild;
  public void displayNode()     { GLOBAL.tempStr += (" "+data); }
 }
 // Ağaç Sınıfı
class Tree
{  private TreeNode root;
   public Tree()     { root = null; }
   public TreeNode getRoot()     { return root; }
// Ağacın preOrder Dola□ılması
public void preOrder(TreeNode localRoot)
{
  if(localRoot!=null)
  {
    localRoot.displayNode();
    preOrder(localRoot.leftChild);
    preOrder(localRoot.rightChild);
  }
 }
```

İkili arama Ağacı– C# (2. Örnek)

```
// Ağacın inOrder Dolaşılması
  public void inOrder(TreeNode localRoot)
  {
   if(localRoot!=null)
   {
    inOrder(localRoot.leftChild);
    localRoot.displayNode();
    inOrder(localRoot.rightChild);
   }
  }
  // Ağacın postOrder Dolaşılması
  public void postOrder(TreeNode localRoot)
  {
   if(localRoot!=null)
   {
    postOrder(localRoot.leftChild);
    postOrder(localRoot.rightChild);
    localRoot.displayNode();
   }
  }
```

İkili arama Ağacı– C# (2. Örnek)

```
// Ağaca bir düğüm eklemeyi sağlayan metot
public void insert (int newdata)
{
 TreeNode newNode = new TreeNode();
 newNode.data = newdata;
 if(root==null)     root = newNode;
 else
 {
  TreeNode current = root;
  TreeNode parent;
  while(true)
  {
   parent = current;
   if(newdata<current.data)
   {
    current = current.leftChild;
    if(current==null)
```

İkili arama Ağacı– C# (2. Örnek)

```
{         parent.leftChild=newNode;
          return;
         }
        }
        else
        {
         current = current.rightChild;
         if(current==null)
         {
          parent.rightChild=newNode;
          return;
         }
        }
      }  // end while
     } // end else not root
   } // end insert()
} // class Tree
```

İkili arama Ağacı– C# (2. Örnek)

```
private void dugme1_Click(object sender, System.EventArgs e)     {
    Random r = new Random();            Tree theTree = new Tree();
    // Ağaca 10 tane sayı yerleştirilmesi
    string str = "";        str += "Sayılar : ";
    for (int i=0;i<10;++i)
    {
      int sayi = (int) (r.Next(1,50));        str += (" "+sayi);        theTree.insert(sayi);
    }
     listBox1.Items.Add(str);      GLOBAL.tempStr = "";
    theTree.inOrder(theTree.getRoot());
    MessageBox.Show("\nAğacın InOrder Dolaşılması   : "+GLOBAL.tempStr);

  GLOBAL.tempStr = "";       theTree.preOrder(theTree.getRoot());
    MessageBox.Show("\nAğacın PreOrder Dolaşılması   : "+GLOBAL.tempStr);

    GLOBAL.tempStr = "";       theTree.postOrder(theTree.getRoot());
    MessageBox.Show("\nAğacın PostOrder Dolaşılması   : "+GLOBAL.tempStr);
  }
```