VERİ YAPILARI VE ALGORİTMALAR

BLM2512 Gr.2

2020-2021 Bahar Yarıyılı (Uzaktan Eğitim)

Dr.Öğr.Üyesi Göksel Biricik

AĞAÇ (TREE)

Ağaç

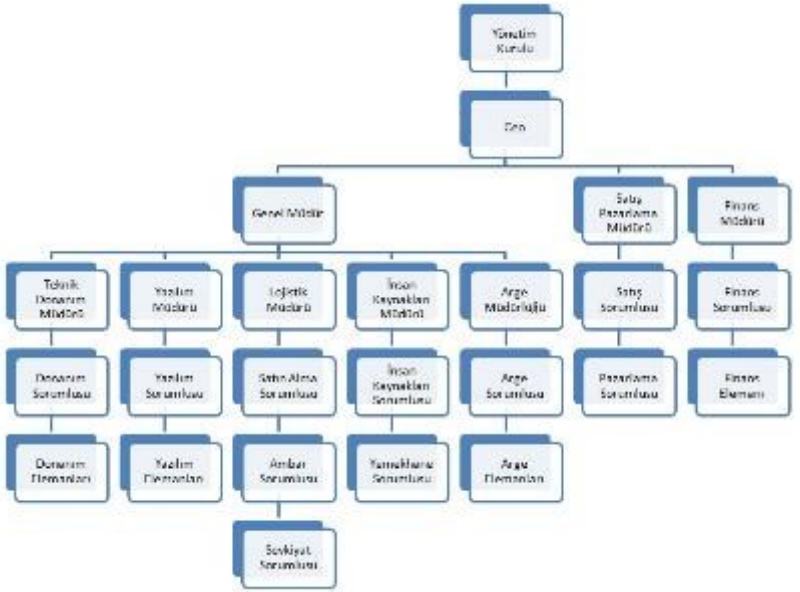
- Hiyerarşik yapıda birbirine bağlı ve kapalı çevrim oluşturmayan sonlu düğümler kümesi

Bir başka tanım:

- Aralarında yönlü bağlantılar bulunan ve aşağıdaki özellikleri içeren düğümler topluluğu:
  - Kendisine doğru kenar içermeyen ve kök (root) olarak adlandırılan özel bir düğüm.
  - Kök haricindeki tüm düğümlere gelen tek bir kenar bulunur.
  - Kök düğümden tüm düğümlere ulaşılan (düğüm ve kenarlardan oluşan) yol tektir.

Ağaç Örnekleri

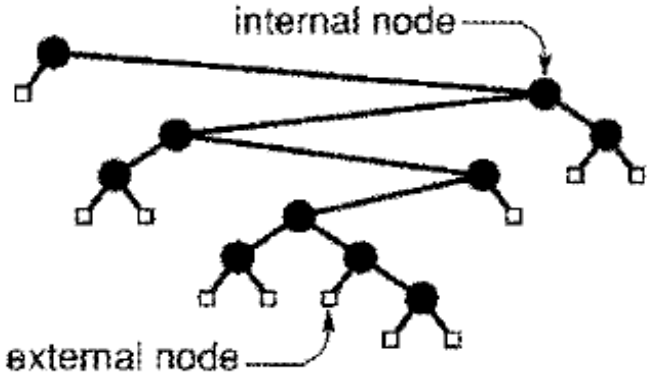
- Örnekler:
  - Soy ağacı
  - Organizasyon şeması
  - Tenis turnuvası
  - İçindekiler tablosu
  - Dosya sistemi
  - Veritabanı indeksleri
  - …



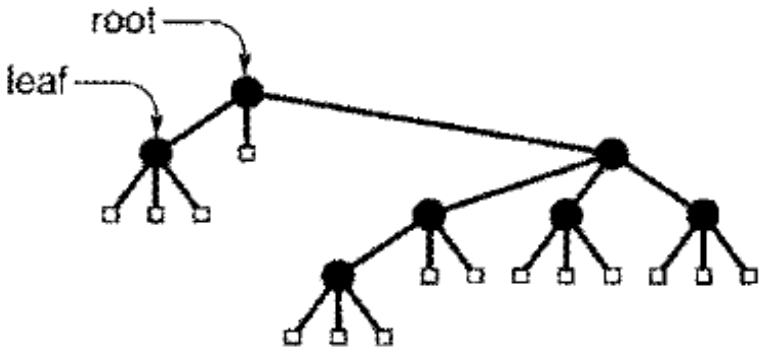
Ağaç Örnekleri

Ağaç Örnekleri

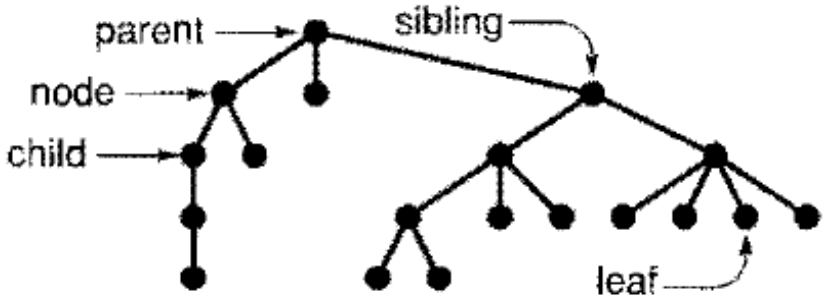
- İkili Ağaç



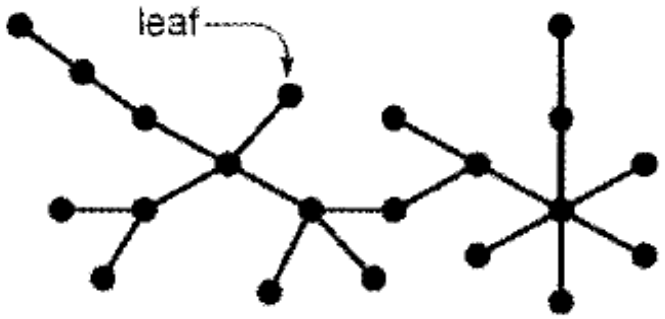
Üçlü Ağaç



- Köklü Ağaç

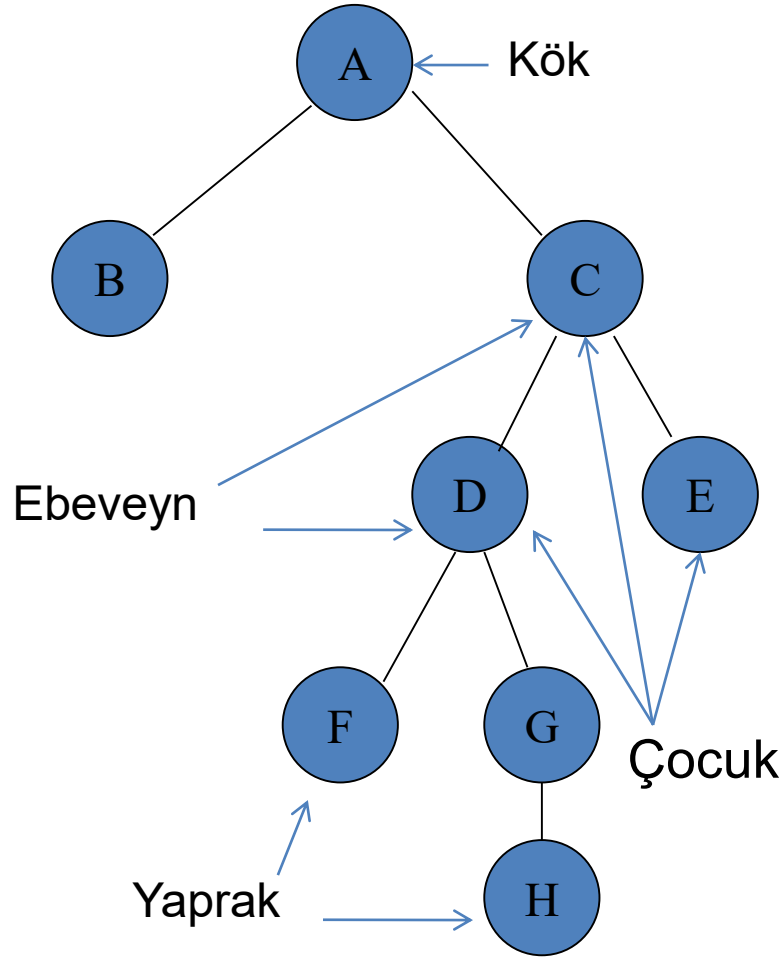


Serbest Ağaç



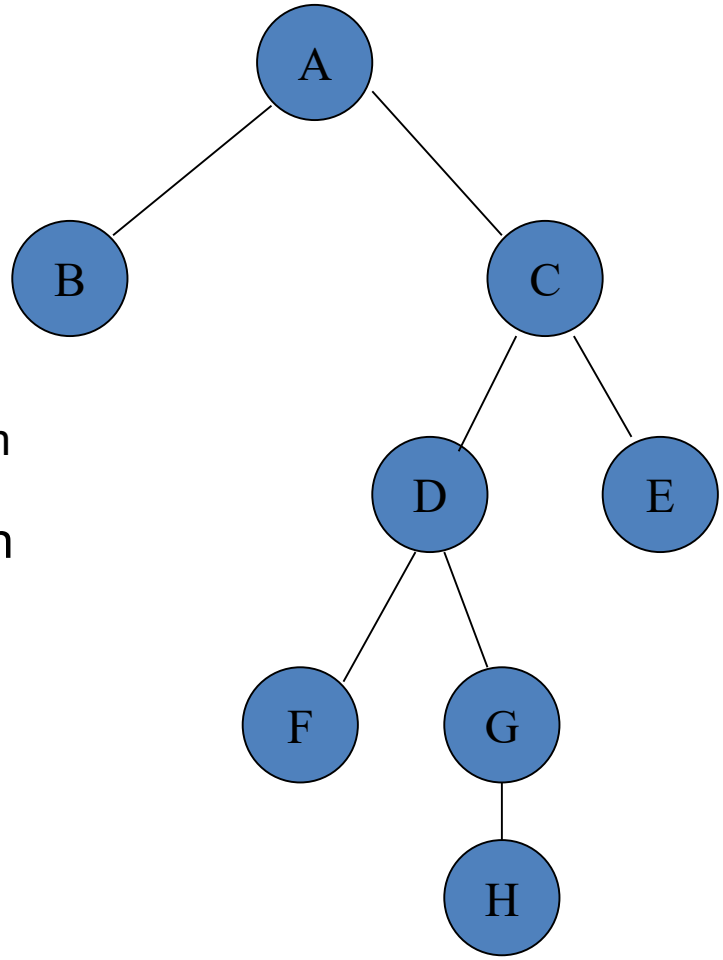
Temel Kavramlar

- Düğüm – veri ve diğer düğümlere bağlantı içeren veri yapısı:
  - **Kök Düğüm (root)** – Tüm düğümlerin atası olan düğüm
  - **Ata / Ebeveyn (Parent)** – Alt ağaçlar olarak düzenlenmiş evlat düğümler içeren düğüm.
  - **Çocuk/ Evlat (Child)** – Ağaç düğümleri 0 ya da daha çok çocuk içerir
  - **Yaprak(Leaf)** – Alt ağacı olmayan düğüm
  - **Kardeş (Sibling)** – Aynı ebeveynin çocukları



Temel Kavramlar

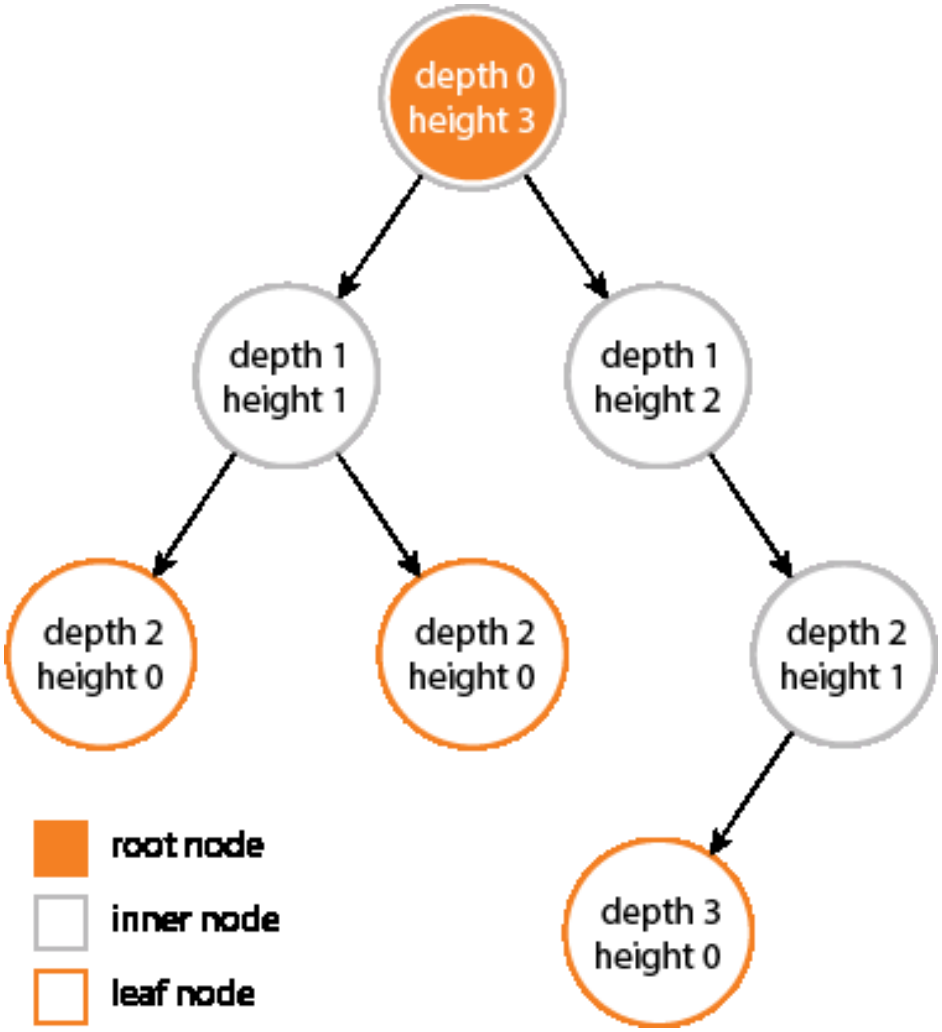
- **Derece(Degree)** – Bir düğümün doğrudan evlat sayısı.
  - Ağacın derecesi: max(degree)
- **Yükseklik (Height)** – Ağaçta kök ile yapraklar arasındaki en uzun yolun kenar sayısı
  - Kökün (Çocuğu varsa) yüksekliği 1'dir.
  - H yüksekliğindeki ikili ağaçta max. kaç düğüm olabilir? 2^h-1
- **Derinlik (Depth)** – Bir düğümden köke olan kenar sayısı
  - Kökün derinliği 0
- **Seviye(Level**)– Bir düğümün kök düğümle arasındaki kenar sayısı. Kök düğümün seviyesi 1, evlatlarının seviyesi 2, …
  - Level = Depth+1



Temel Kavramlar

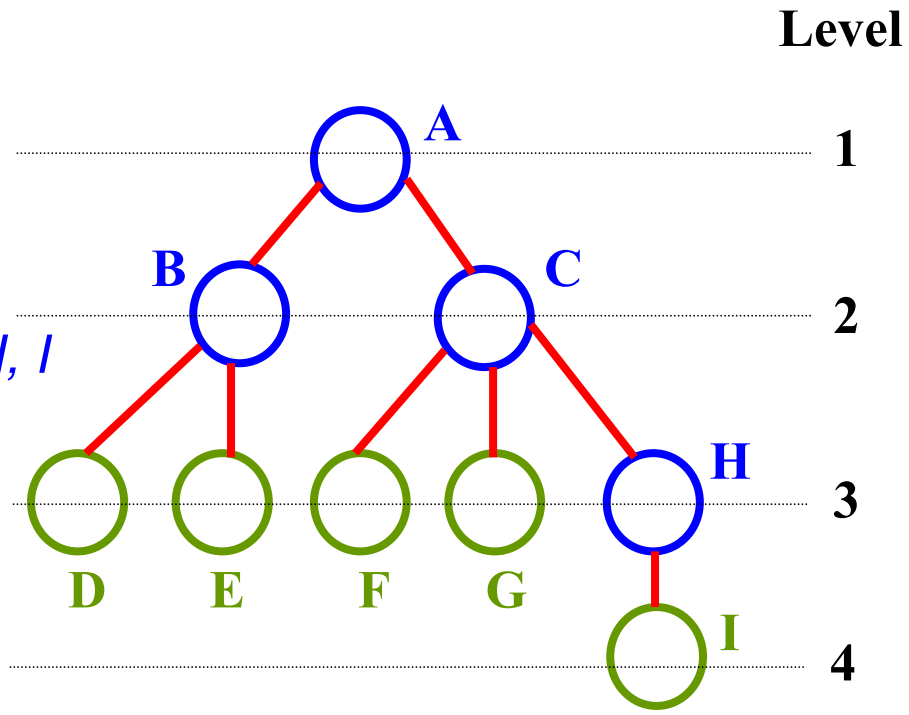
- A düğümünden B düğümüne bir kenar varsa; A, B'nin **ebeveyni**, B, A'nın **evladı** olarak adlandırılır.
- Aynı ebeveynin çocukları bir birine göre **kardeştir**.
- A'dan B'ye yol varsa; A, B'nin **atası**, B, A'nın **torunu** olarak adlandırılır.
- Bir düğüm, tüm torunlarıyla birlikte **alt ağaç** olarak adlandırılır.
- Bir düğümün hiç evladı yoksa, A ağacın **yaprağı** olarak adlandırılır.
- Bir düğümün atası yoksa, düğüm ağacın **köküdür**.

Depth-Height İlişkisi



Örnek

A is the *root* node
B is the *parent* of D and E
C is the *sibling* of B
D and *E* are the *children* of B
D, E, F, G, I are *external nodes*, or *leaves*
A, B, C, H are *internal nodes*
The *level* of *E* is *3*
The *height(depth)* of the tree is *3*
The *degree* of node *B* is *2*
The *degree* of the tree is *3*
The *ancestors* of node *I* is *A, C, H*
The *descendants* of node *C* is *F, G, H, I*



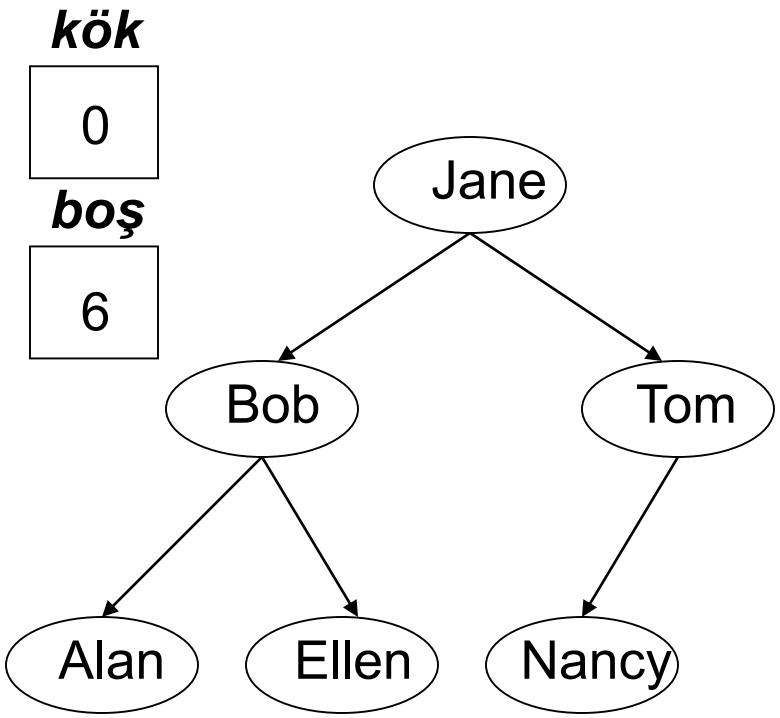
(İkili) Ağaç Oluşturma – Dizi ile

Temel Fikir

- Sol ve sağ evladı göstermek için dizi indisleri kullanılabilir.
- Yeni bir eleman gerektiğinde nereye eklenebileceğini gösteren boş değişkeni kullanılabilir. Sonraki boş alan bu adresin sol ya da sağ evladında yer alabilir.
- Birlikte, dizide kullanılabilir yerlerin listesi *boş listesi* (***free list***) olarak adlandırılır.

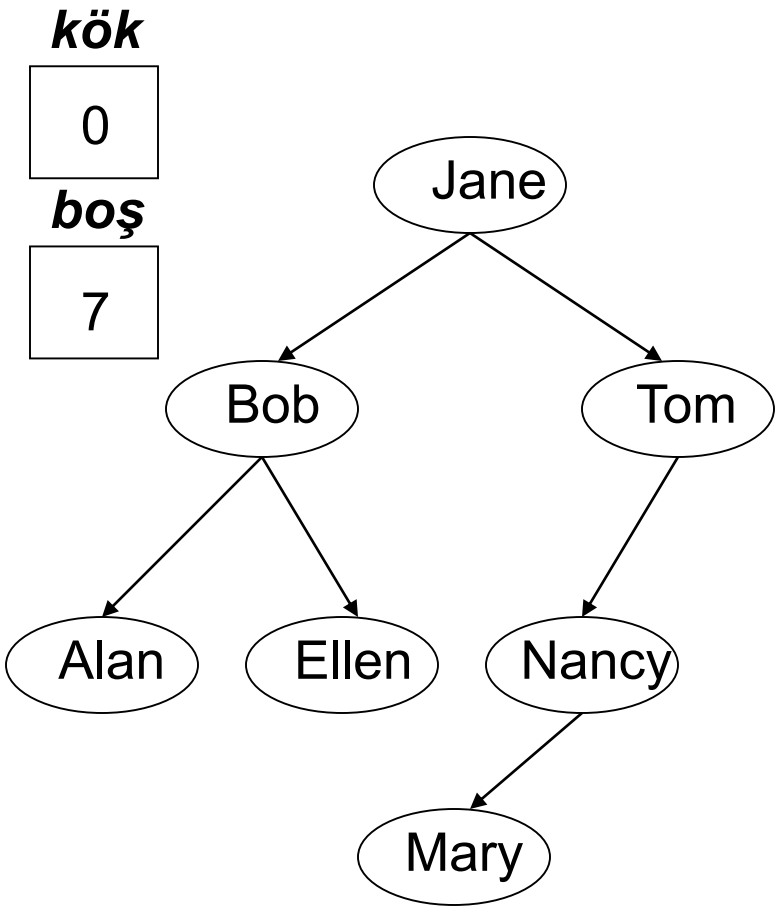
İkili Ağaç: Tablo Gösterimi

İndis	Veri	Sol Evlat	Sağ Evlat
0	Jane	1	2
1	Bob	3	4
2	Tom	5	-1
3	Alan	-1	-1
4	Ellen	-1	-1
5	Nancy	-1	-1
6	?	-1	7
7	?	-1	8
8	?	-1	9
9	. . .	. . .	. . .



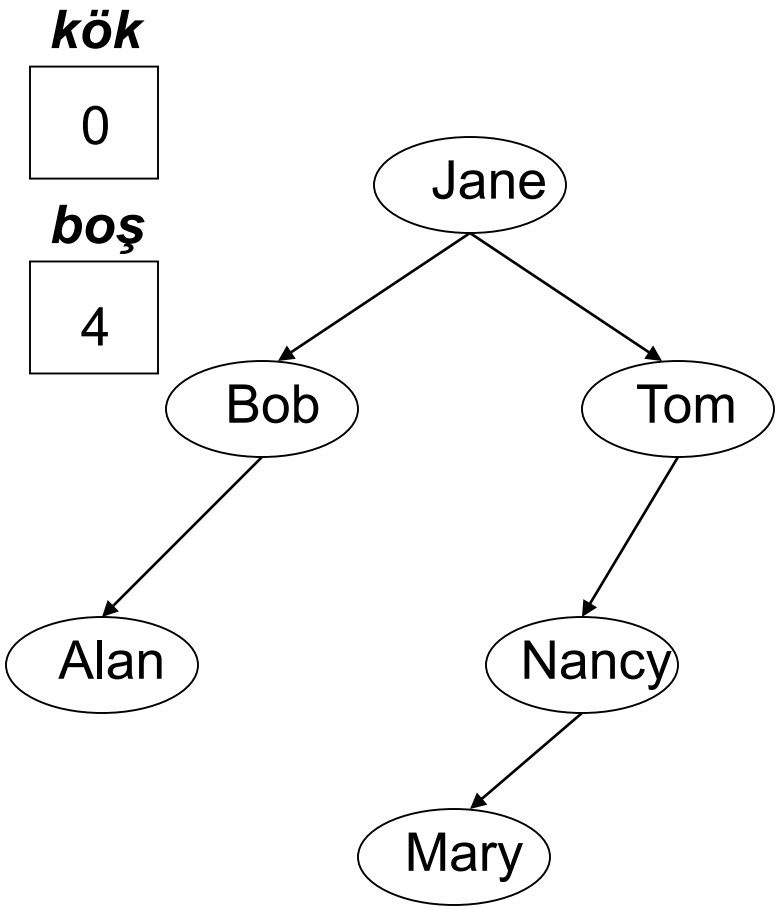
İkili Ağaç: Tablo Gösterimi

İndis	Veri	Sol Evlat	Sağ Evlat
0	Jane	1	2
1	Bob	3	4
2	Tom	5	-1
3	Alan	-1	-1
4	Ellen	-1	-1
5	Nancy	6	-1
6	Mary	-1	-1
7	?	-1	8
8	?	-1	9
9	. . .	. . .	. . .



İkili Ağaç: Tablo Gösterimi

İndis	Veri	Sol Evlat	Sağ Evlat
0	Jane	1	2
1	Bob	3	4
2	Tom	5	-1
3	Alan	-1	-1
4	?	-1	7
5	Nancy	6	-1
6	Mary	-1	-1
7	?	-1	8
8	?	-1	9
9	. . .	. . .	. . .

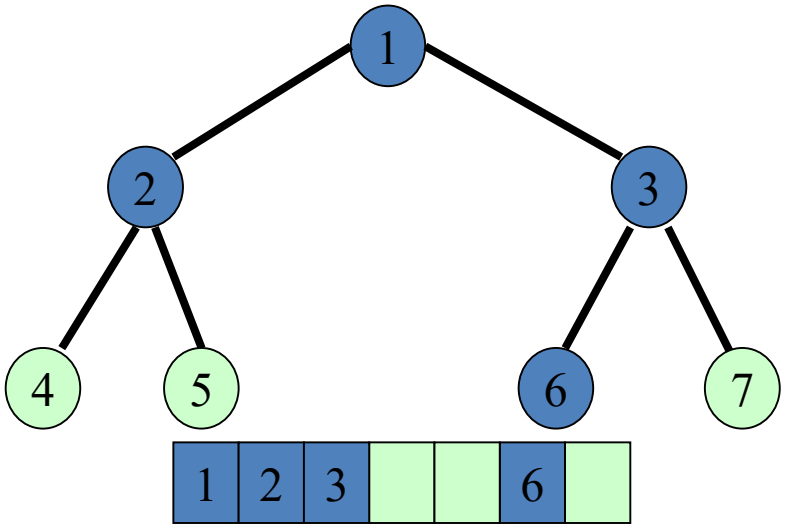


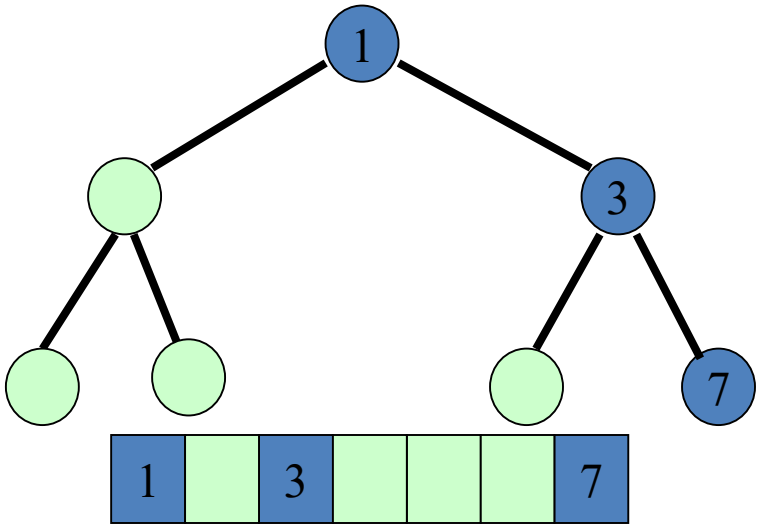
(İkili) Ağaç Oluşturma – Dizi ile

```
#define MAX 100

typedef struct node{
	int data;
	int left;
	int right;
}node;

node tree[MAX];
int root;
int bos;
// left child: 2j+1
// right child: 2j+2
// parent: floor((j-1)/2)
```

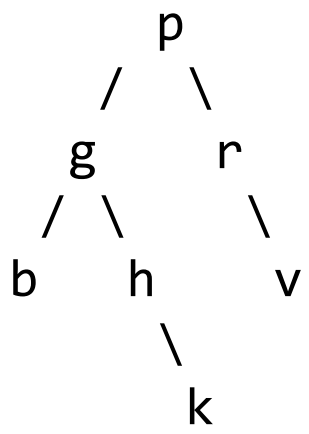
(İkili) Ağaç: Dizi Tabanlı Gösterim

- n eleman için 2^n-1 yere ihtiyaç duyarız.
  - Ağaç tam dolu ise, dizide boşluk olmaz.
  - Tam dolu değilse, pek çok yeri israf ederiz.



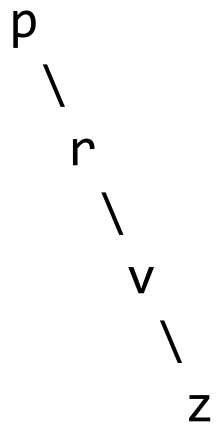


(İkili) Ağaç: Dizi Tabanlı Gösterim



0	1	2	3	4	5	6	7	8	9	10	…
p	g	r	b	h	_	v	_	_	_	k	…

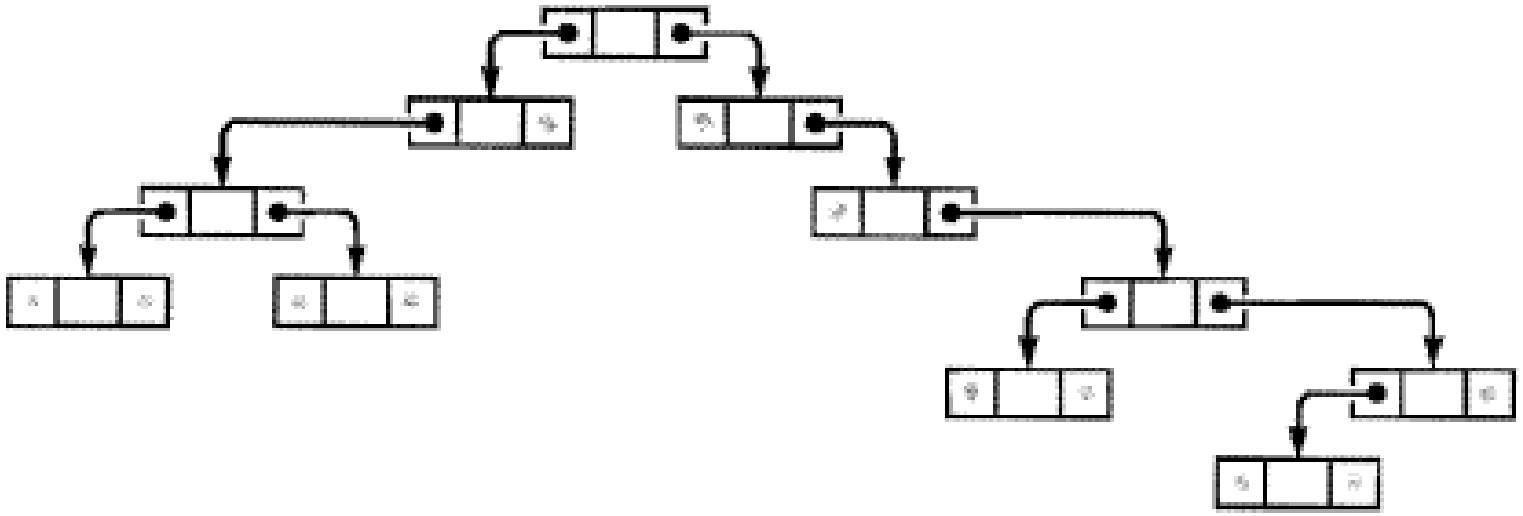
- En kötü durum:



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	…
p	_	r	_	_	_	v	_	_	_	_	_	_	_	z	…

(İkili) Ağaç Oluşturma – İşaretçi ile

- Sol ve sağ evlatları gösterdiğimiz iki işaretçili bir linkli liste ile yer kaybını önleyebiliriz.



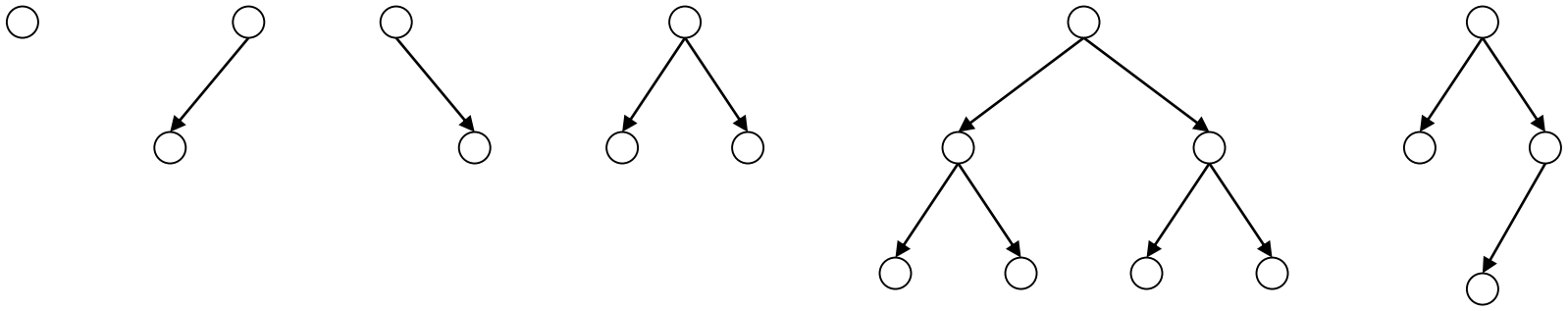
(İkili) Ağaç Oluşturma – İşaretçi ile

```
typedef struct node *tree;
typedef struct node{
	int data;
	tree left, right;
	// tree child, sibling;
}node;

tree createTree(int data){
	tree T = (tree)malloc(sizeof(tree));
	T->data = data;
	T->left = NULL;
	T->right = NULL;
	return T;
}
```

İKİLİ AĞAÇLAR

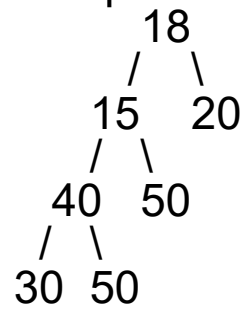
İkili Ağaçlar

- Hiçbir düğümün ikiden fazla çocuğu olamaz.
- (Özyinelemeli) tanım:
  - İkili ağaç, ya boş ya da bir kök düğüm ve buna bağlı olup sol ve sağ alt ağaçlar adı verilen iki bağımsız ikili ağaçtan oluşan sonlu düğümler kümesidir.
  - Tüm ağaçları, önceden de gördüğümüz gibi, sol çocuk / sağ kardeş gösterimi ile ifade edebiliriz.
- İşlemler:
  - BT Create()
  - BT makeBT(bt1, data, bt2) iki alt ağaç ve bir kök ile ağaç oluşturma
  - BT Lchild(bt): if isempty(bt) return error else return left subtree
  - Element Data(bt) if isempty(bt) return error else return data of the root
  - BT Rchild(bt): if isempty(bt) return error else return right subtree

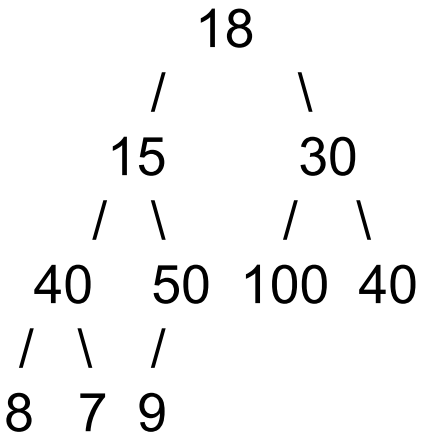


İkili Ağacın Özel Durumları

- **Full Binary Tree:** Her düğümün 0 ya da 2 çocuğu vardır.
  - Yapraklar hariç tüm düğümlerin 2 çocuğu vardır.

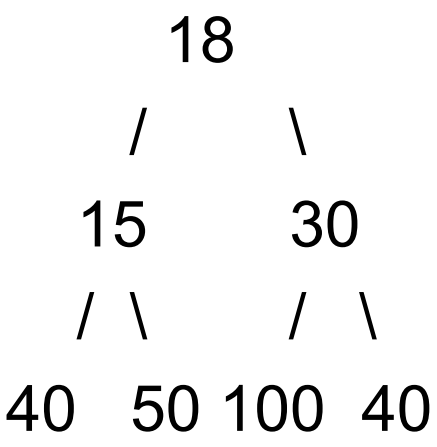


- **Complete Binary Tree:** Son hariç tüm düzeylerin (level) tam dolu olduğu ve son düzeyin de en soldan dolduğu ikili ağaçlar



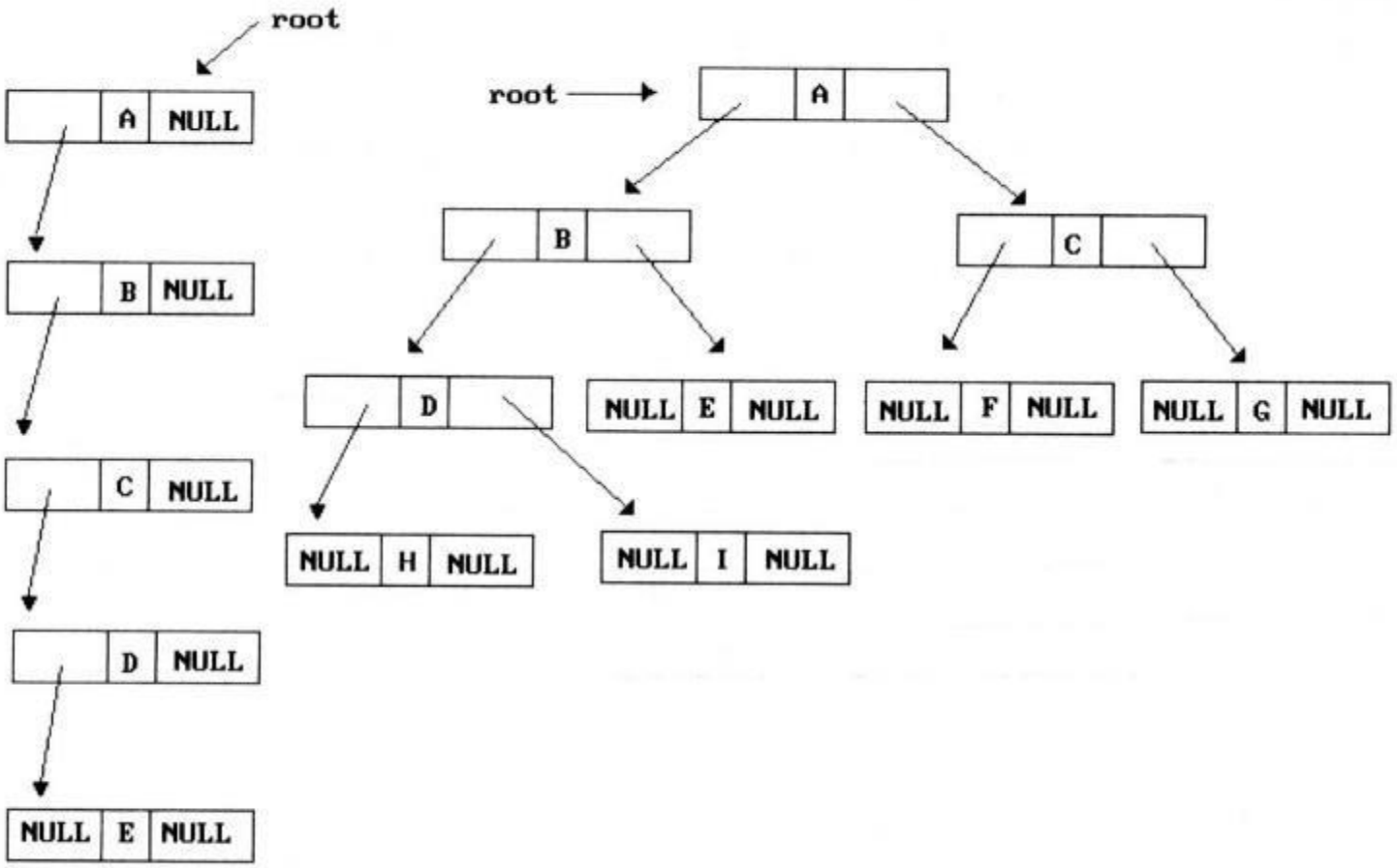
İkili Ağacın Özel Durumları

- **Perfect Binary Tree:** Tüm düğümlerin iki çocuğu vardır ve tüm yapraklar aynı seviyededir.



- **Balanced Binary Tree:** Ağacın yüksekliği n düğüm varken LogN olduğunda dengelidir.
- **Degenerated (Pathological) Binary Tree:** Her i. düğümün tek çocuğu vardır. (Linkli Liste haline gelir)

Link ile İkili Ağaç Gösterimi



İkili Ağaç - Yaratma

```
typedef struct node *tree;
typedef struct node{
        char data;
        tree left, right;
}node;

tree createTree(char data){
        tree T = (tree)malloc(sizeof(tree));
        T->data = data;
        T->left = NULL;
        T->right = NULL;
        return T;
}

void visit(tree t){
        printf("%c\n", t->data);
}
```

İkili Ağaç – Eleman Ekleme

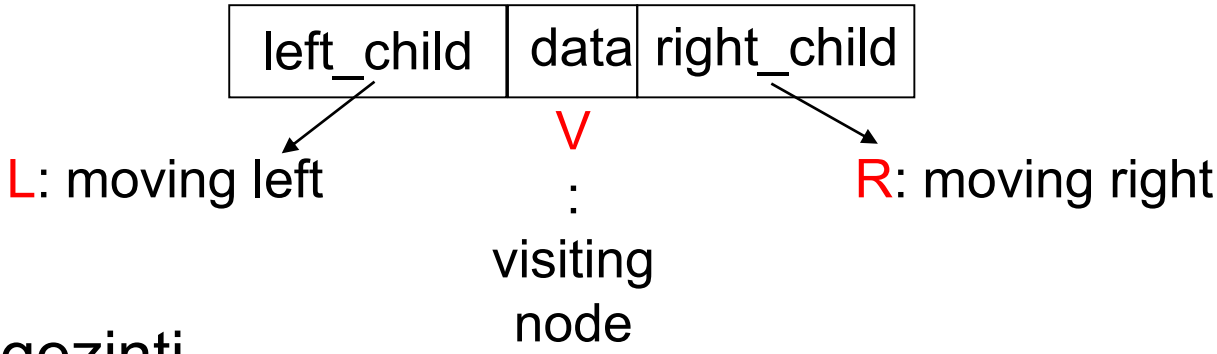
```
tree insert(tree node, char data){
	if (node == NULL) return createTree(data);

	if(data < node->data)
		node->left = insert(node->left, data);
	else if (data > node->data)
		node->right = insert(node->right, data);

	return node;
}
```

İkili Ağaç – Eleman Silme

```
tree del(tree root, char data){
            if (root==NULL) return root;

            // silinecek küçükse soldadır
            if(data < root->data)
                        root->left = del(root->left, data);
            //silinecek büyükse sağdadır
            else if(data > root->data)
                        root->right = del(root->right, data);
            // bu nodu sileceğiz
            else{
                        //node with 1 or 0 child
                        if (root->left == NULL){
                                    tree temp = root->right;
                                    free(root);
                                    return temp;
                        }
                        else if (root->right == NULL){
                                    tree temp = root->left;
                                    free(root);
                                    return temp;
                        }
                        tree temp = minValueNode(root->right);
                        root->data = temp->data;

                        root->right = del(root->right, temp->data);
            }
            return root;
}
```

İkili Ağaçta Gezinti (Binary Tree Traversal)

- İkili ağaçta tüm düğümleri bir kez ziyaret ederek nasıl gezebiliriz?
- Derinlik öncelikli gezinti
  - **L**eft, **V**isit, **R**ight:
  - LVR, LRV, VLR, VRL, RVL, RLV
  - Önce sola, sonra sağa gezebiliriz:
  - **V**LR preorder
  - L**V**R inorder
  - LR**V** postorder

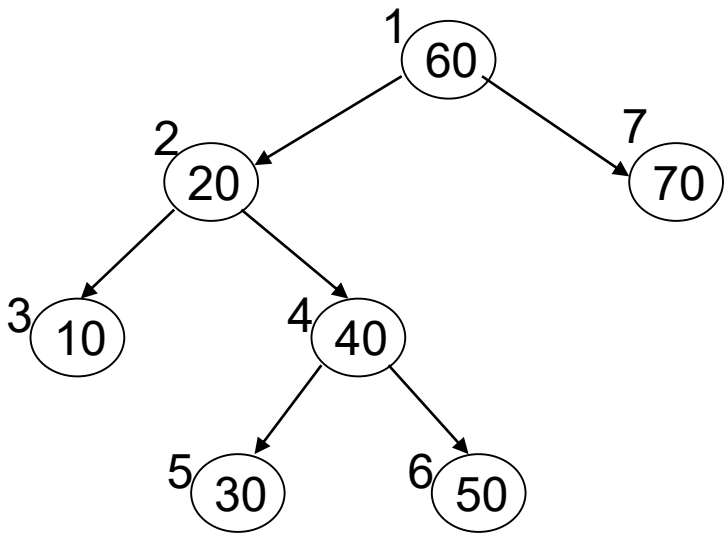


- Genişlik öncelikli gezinti
  - Seviye sırası

Preorder Traverse (Kök-Önde Gezinti)

Temel Fikir:

1) ***Kök***'ü ziyaret et.
2) Özyineli olarak sol alt ağacı kök önde dolaş.
3) Özyineli olarak sağ alt ağacı kök önde dolaş.



60, 20, 10, 40, 30, 50, 70

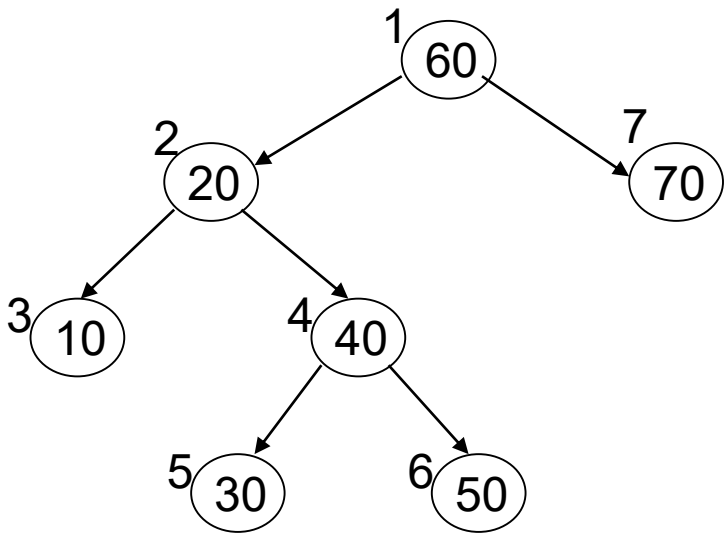
Preorder Traverse (Kök-Önde Gezinti)

```
void preTraverse(tree t){
 if (t==NULL) return;


 visit(t);
 preTraverse(t->left);
 preTraverse(t->right);
}
```

Inorder Traverse (Kök-Ortada Gezinti)

Temel Fikir:

1) Özyineli olarak sol alt ağacı kök ortada dolaş.
2) ***Kök***'ü ziyaret et.
3) Özyineli olarak sağ alt ağacı kök ortada dolaş.



10, 20, 30, 40, 50, 60, 70

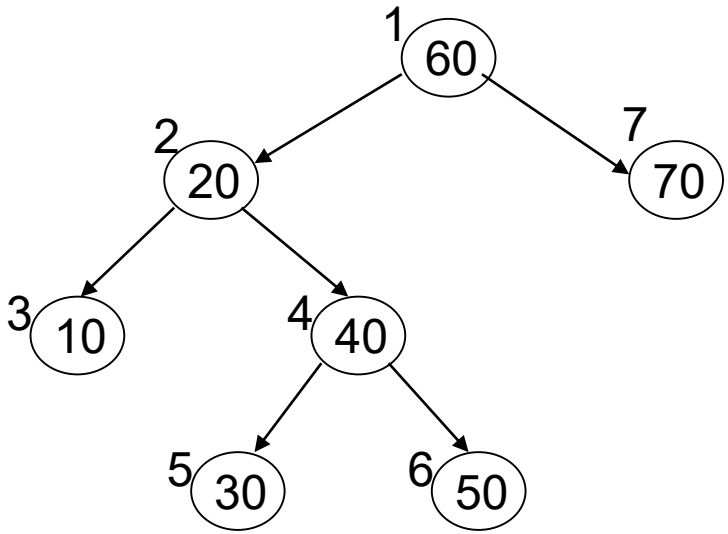
Inorder Traverse (Kök-Ortada Gezinti)

```
void inTraverse(tree t){
      if (t==NULL) return;

      inTraverse(t->left);
      visit(t);
      inTraverse(t->right);
}
```

Postorder Traverse (Kök-Sonda Gezinti)

Temel Fikir:

1) Özyineli olarak sol alt ağacı kök sonda dolaş.
2) Özyineli olarak sağ alt ağacı kök sonda dolaş.
3) ***Kök***'ü ziyaret et.



10, 30, 50, 40, 20, 70, 60

Postorder Traverse (Kök-Sonda Gezinti)

```
void postTraverse(tree t){
 if (t==NULL) return;


 postTraverse(t->left);
 postTraverse(t->right);
 visit(t);
}
```

Alternatif Gezintiler

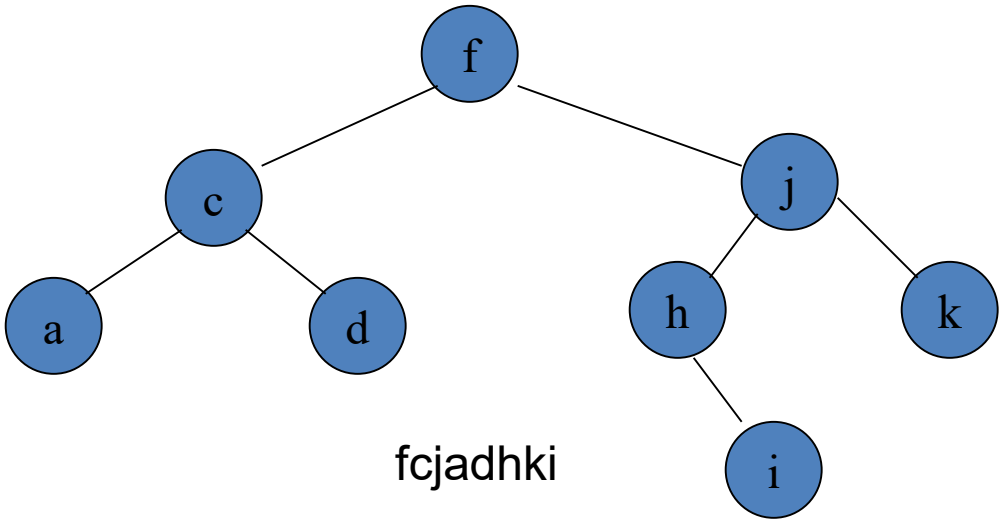
- Özyinelemesiz Inorder: Stack ile yapabiliriz.

1. current = root
2. while (current != NULL)
    1. push(current)
    2. current = current->left
3. while(current == NULL && !isEmpty())
    1. x=pop()
    2. print(x)
    3. current = x->right
4. if (current == NULL && isEmpty()) success

- Genişlik öncelikli gezinti
    - Her seviyede Inorder gezinti yapılır.

Alternatif Gezintiler

- Genişlik öncelikli gezinti (Seviye sıralı gezinti)
  - Her seviyede Inorder gezinti yapılır.
  - Kuyruk ile gerçekleştirebiliriz.
- Kuyruk boşalıncaya kadar yinele
  - Ağacı seviye sırasında soldan sağa dolaş:
    - Kök'ü dolaş ve evlatlarını soldan sağa kuyruğa ekle.
    - Kuyruktan bir düğüm al, dolaş, evlatlarını kuyruğa ekle.



İlave İşlemler – Ağaç kopyalama

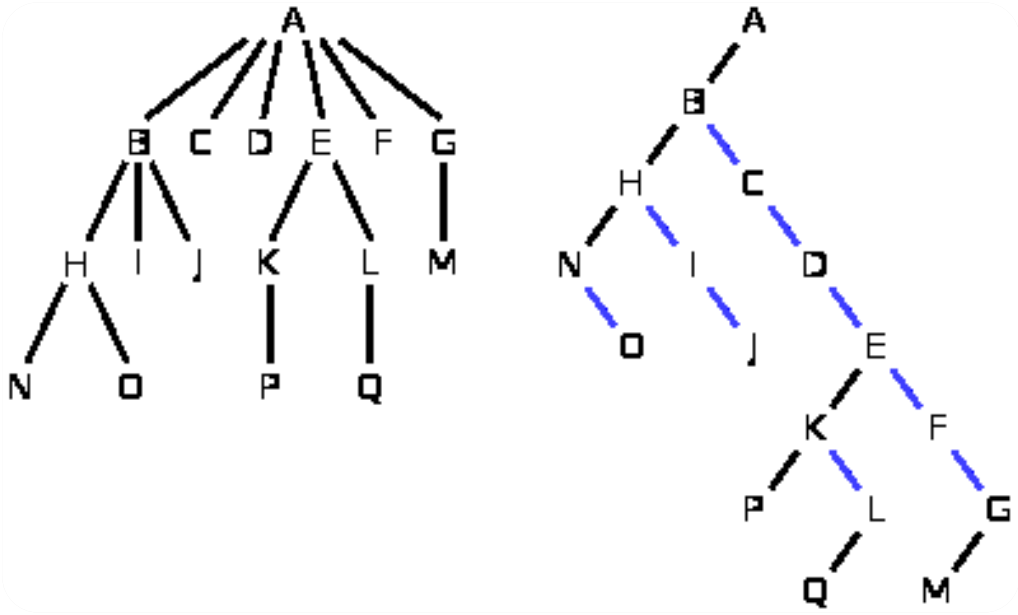
```
tree copyTree(tree orig){
    tree tmp;
    if (orig){
        tmp = (tree)malloc(sizeof(tree));
        // memcheck
        tmp->left = copyTree(orig->left);
        tmp->right = copyTree(orig->right);
        tmp->data = orig->data;
        return tmp;
    }
    return NULL;
}
```

İlave İşlemler – Ağaçlar eşit mi?

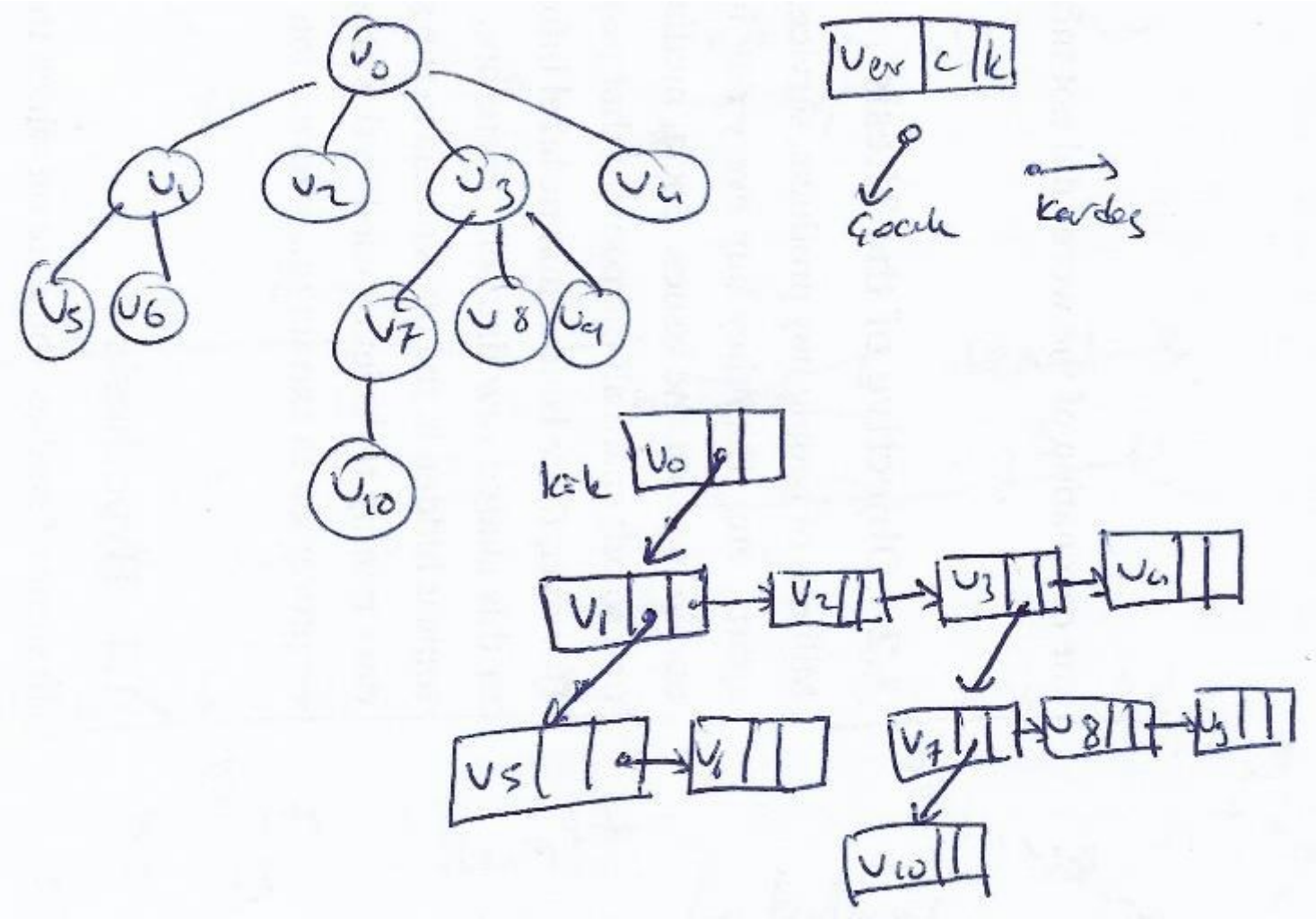
```
int isEqualTree(tree first, tree second){
   return( (!first && !second) ||
           (
             (first) &&
             (second) &&
             (first->data == second->data) &&
             isEqualTree(first->left, second->left) &&
             isEqualTree(first->right, second->right)
           )
   );
}
```

Çoklu Ağaçlar

- Çok evlatlı ağaçlar, ikili ağaç olarak ifade edilebilir. İlk oğul için bir işaretçi, sağdaki ilk kardeş için bir işaretçi yeterlidir.



İki İşaretçi ile (Genel) Ağaç Oluşturma

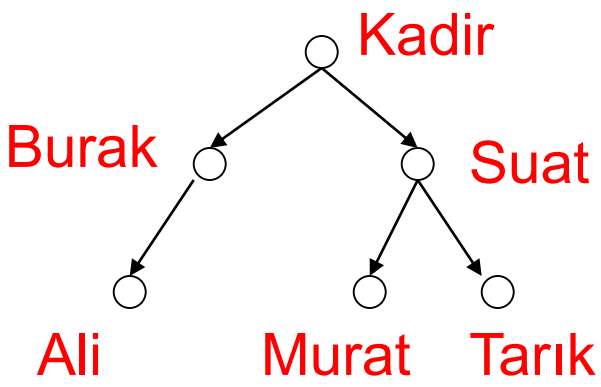


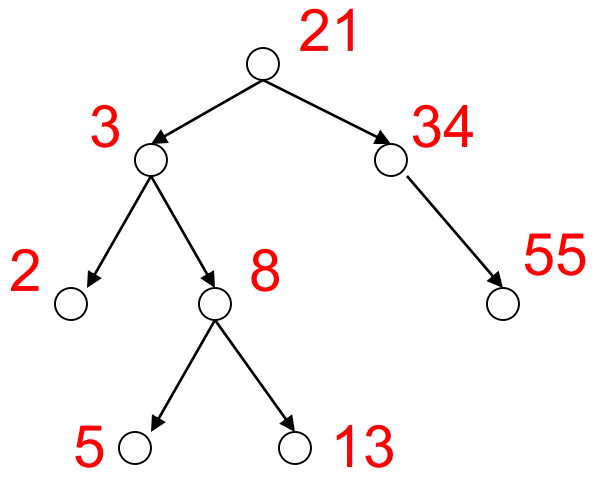
İKİLİ ARAMA AĞACI

Binary Search Tree

İkili Arama Ağacı (Binary Search Tree)

- İkili Arama Ağacı, her bir A düğümümün aşağıdaki özellikleri taşıdığı ağaçtır:
  - *A'nın değeri sol alt ağacındaki, T_L, düğümlerin değerlerinden büyüktür.*
  - *A'nın değeri sağ alt ağacındaki, T_R, düğümlerin değerlerinden küçüktür.*
  - T_L ve T_R birer ikili arama ağacıdır.





İkili Arama Ağacı

- Örnek: b ve c ikili arama ağacıdır.

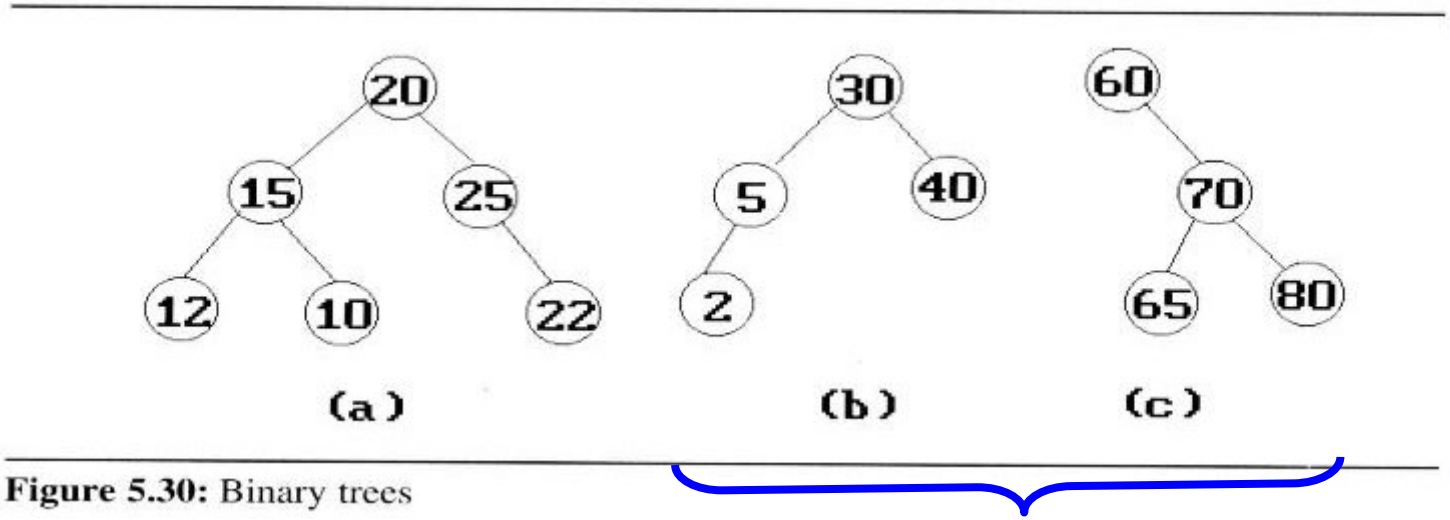
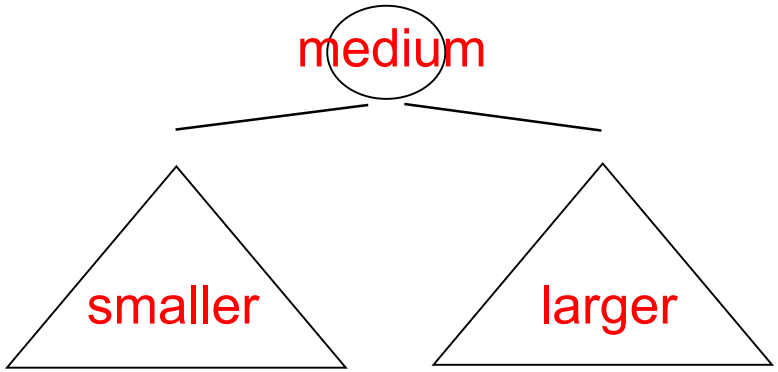


Figure 5.30: Binary trees

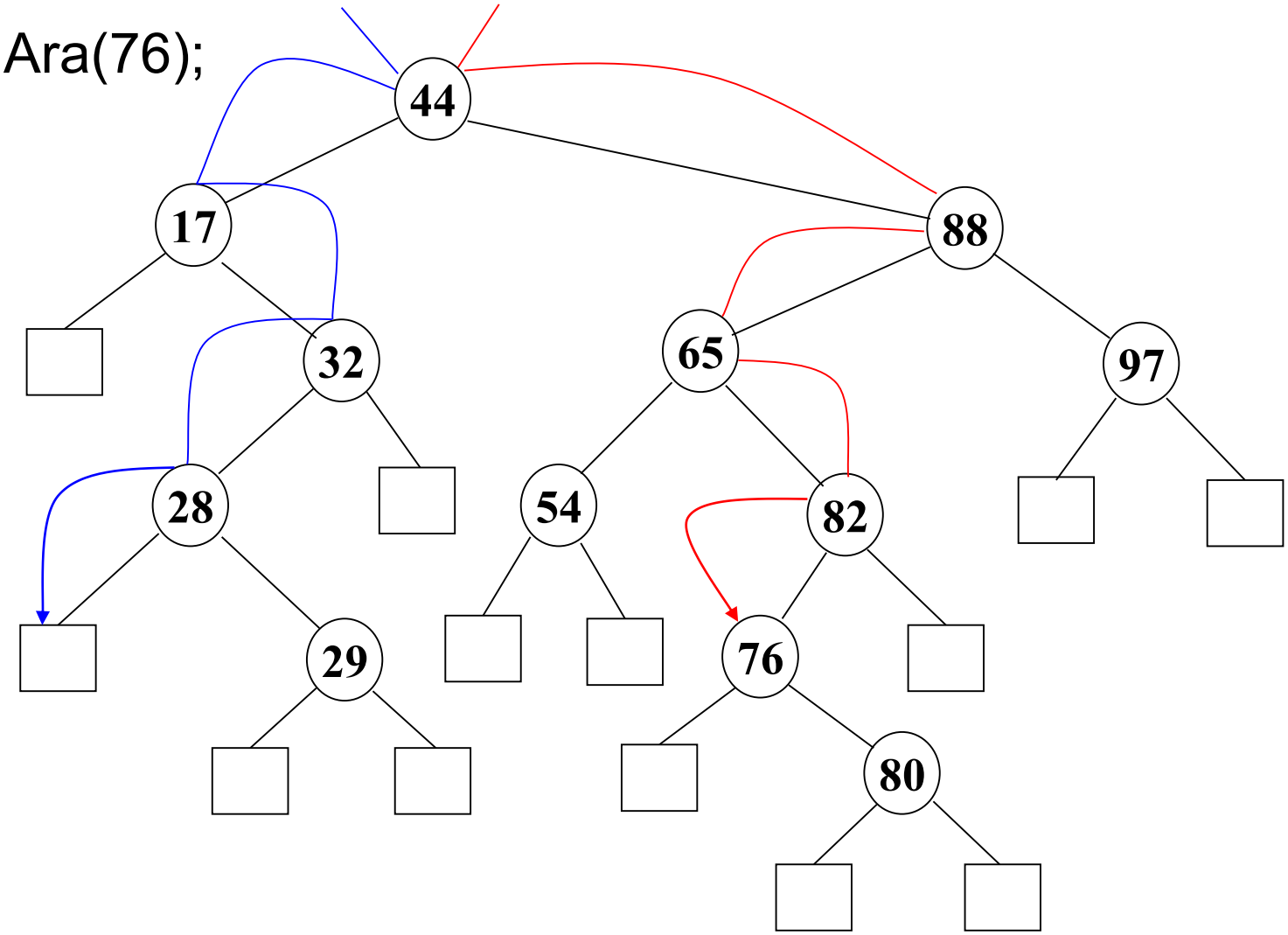


İkili Arama Ağacı – Eleman Arama

```
tree find(tree root, char data){
    if (root->data > data)
        return find(root->left, data);
    else if (root->data < data)
        return find(root->right, data);
    else
        return root; //ya köke eşit ya da null
}
```

İkili Arama Ağacı

- Ara(25); Ara(76);



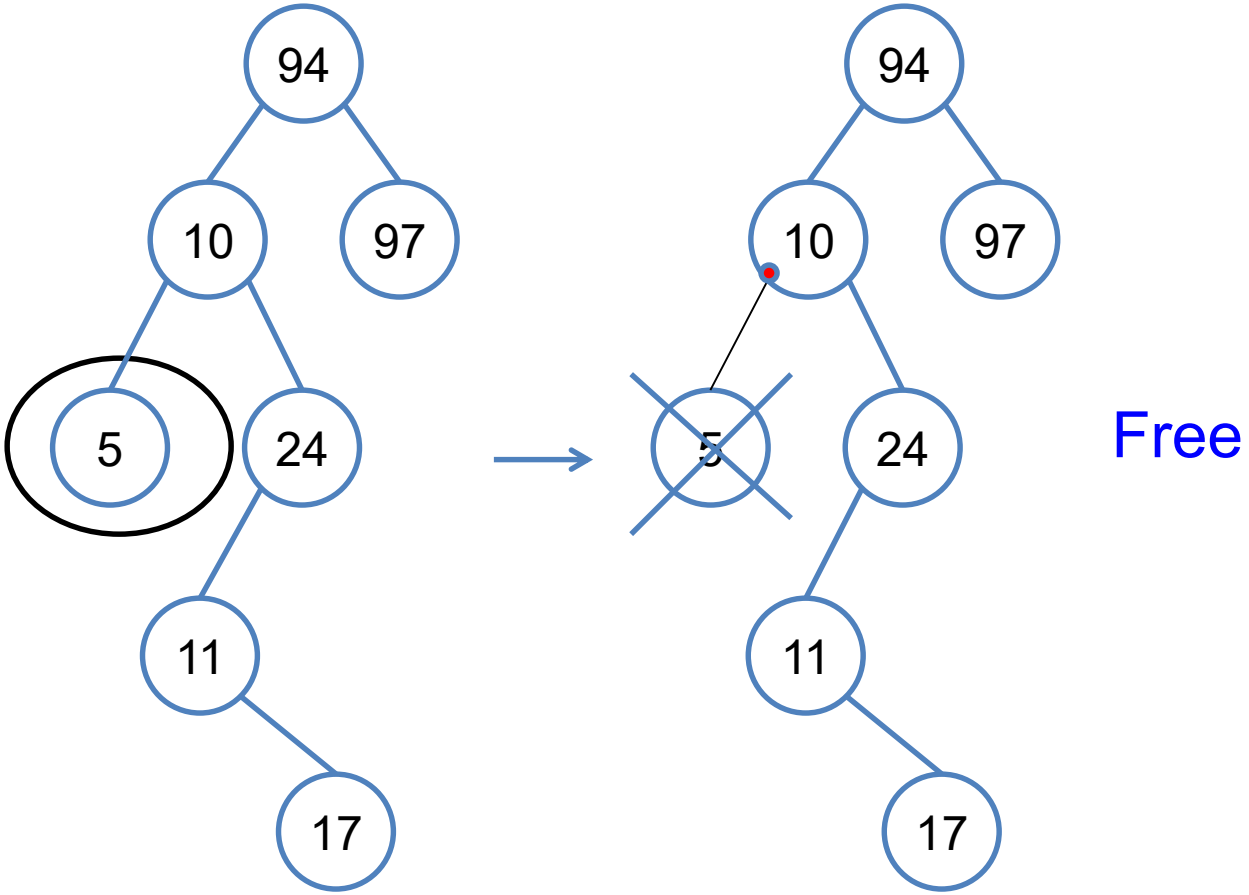
İkili Arama Ağacı – Eleman Ekleme

```
tree insert(tree node, char data){
    if (node==NULL)
        return createTree(data);
    if (data < node->data)
        node->left = insert(node->left, data);
    else if (data > node->data)
        node->right = insert(node->right, data);

    return node;
}
```

İkili Arama Ağacı – Eleman Silme – Yaprak (Çocuk yok)

- Direk Sileriz

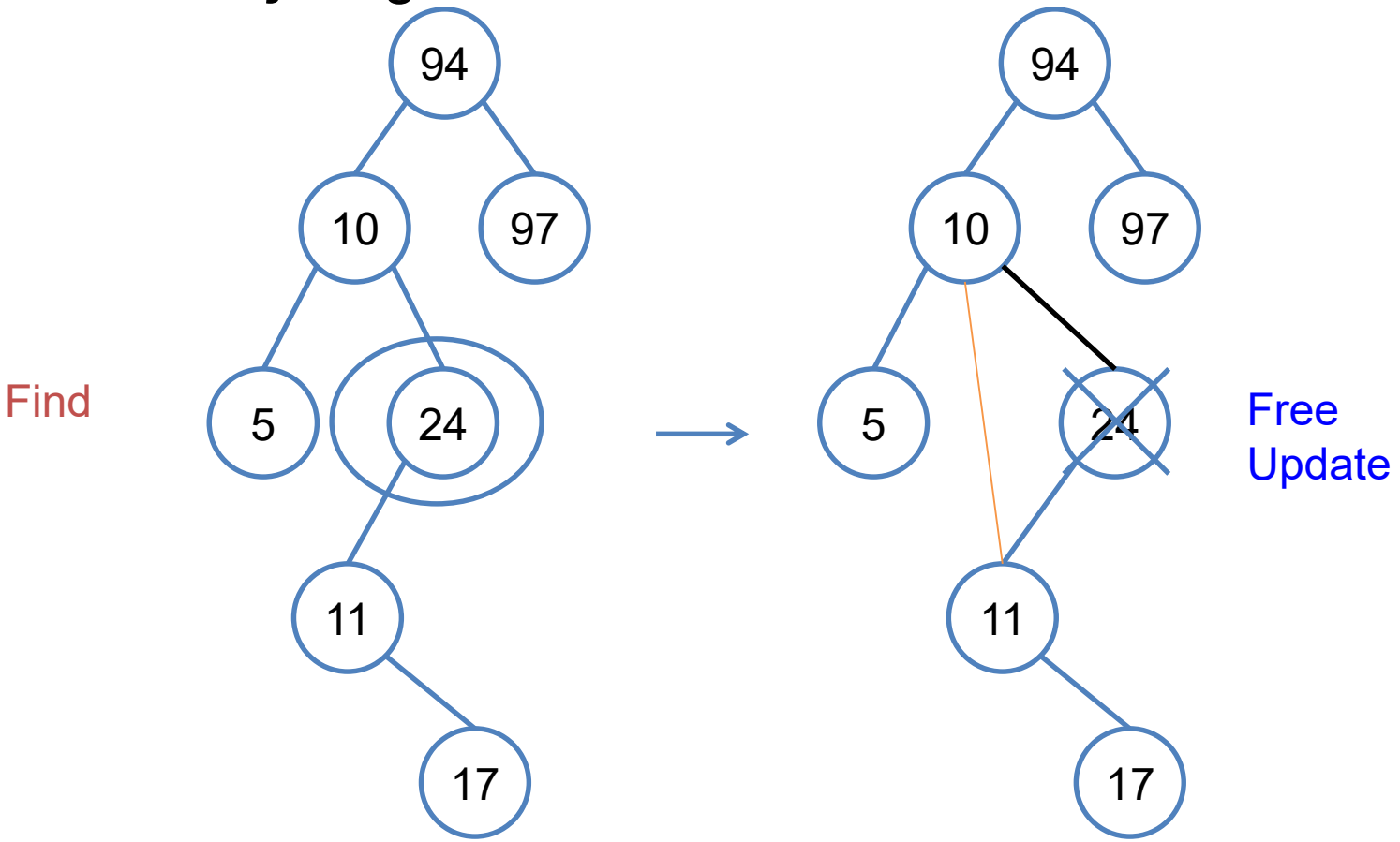
Find



Free

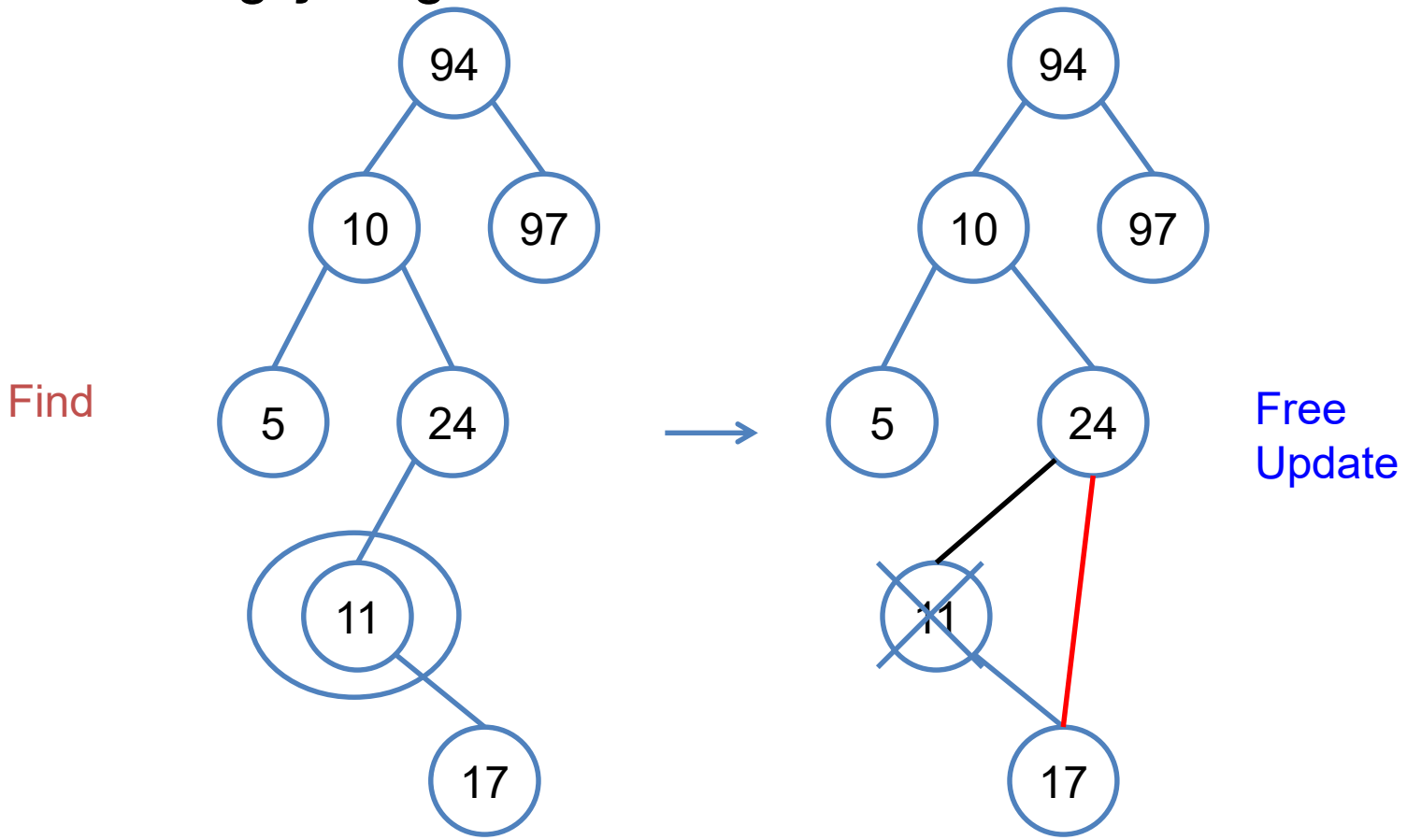
İkili Arama Ağacı – Eleman Silme – Tek Çocuk

- Çocuğu düğüme kopyala ve çocuğu sil.
- Örnek: Sol çocuğu var.



İkili Arama Ağacı – Eleman Silme – Tek Çocuk

- Çocuğu düğüme kopyala ve çocuğu sil.
- Örnek: Sağ çocuğu var.



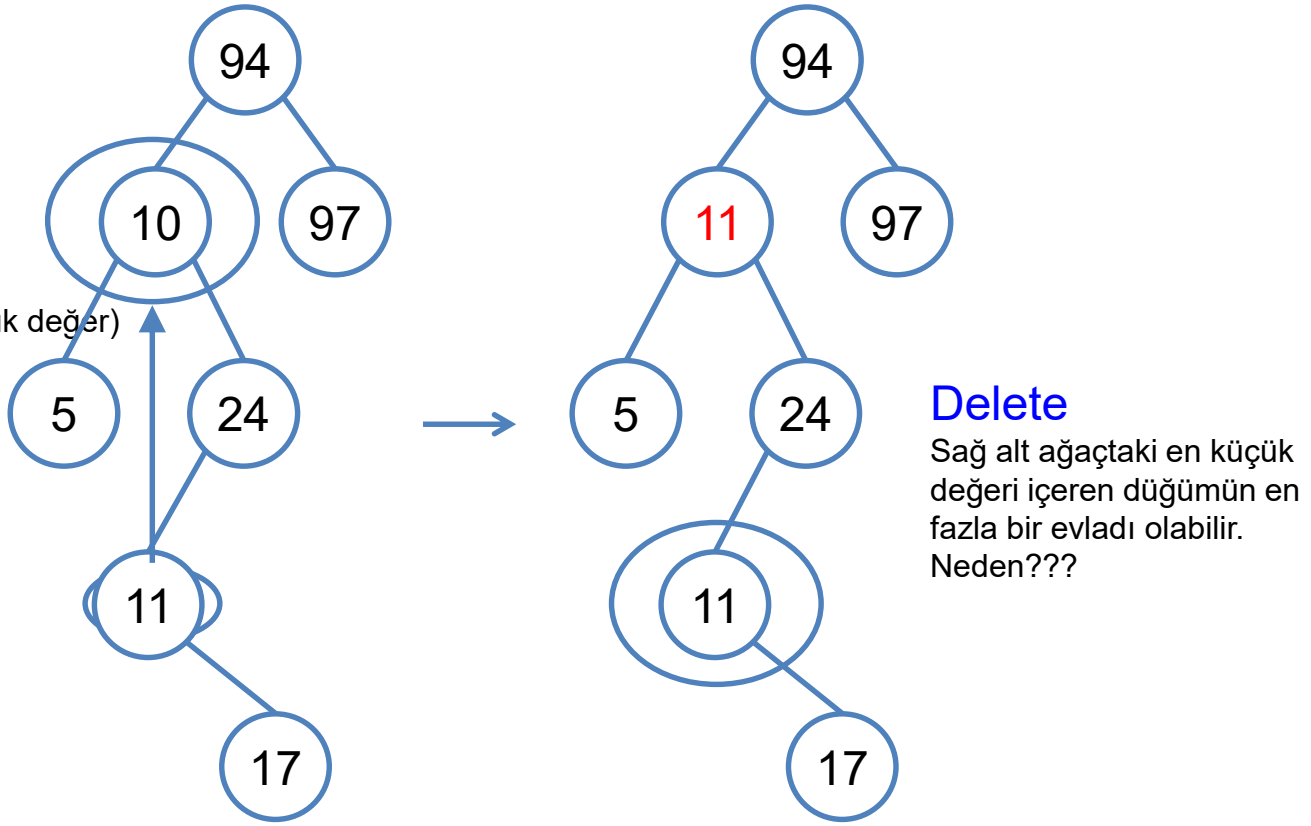
İkili Arama Ağacı – Eleman Silme – İki Çocuk

- Sağ alttaki en küçük ya da sol alttaki en büyük değere sahip çocuğu düğüme kopyala ve çocuğu sil.

Find

Copy

(sağ alt ağaçtan en küçük değer)



Delete

Sağ alt ağaçtaki en küçük değeri içeren düğümün en fazla bir evladı olabilir. Neden???

İhtiyacımız olacak: findMin - findMax

```
tree findMin(tree T){
    if(T && T->left)
        return findMin(T->left);
    else
        return T;
}
tree findMax(tree T){
    if(T && T->right)
        return findMax(T->right);
    else
        return T;
}
```

İkili Arama Ağacı – Eleman Silme

```
tree delete(tree T, char data){
   if (T==NULL)
      return NULL;
   if (data < T->data)
      T->left = delete(T->left, data);
   else if (data > T->data)
      T->right = delete(T->right, data);
   else if (T->left != NULL && T->right !=NULL){
      T->data = findMin(T->right)->data;
      t->right = delete(T->right, T->data);
   }
   else
      T = (T->left != NULL) ? T->left : T->right;

   return T;
}
```

İkili Arama Ağacı

- Ağacın Yüksekliği:

- N elemanlı bir ağaç, N yüksekliğe erişebilir.
- Ekleme ve silmeler rasgele olursa, ortalamada $O(\log_2 N)$ yüksekliğe sahip olur.
- Bunlara dengeli arama ağacı da denir.

- Arama, ekleme, silme: O(h) (yükseklik)
  - En kötü durum: O(n)
  - Nasıl önlenir? Dengeleme politikaları
    - AVL, Splay, B-Tree (2-3, 2-3-4), red-black tree