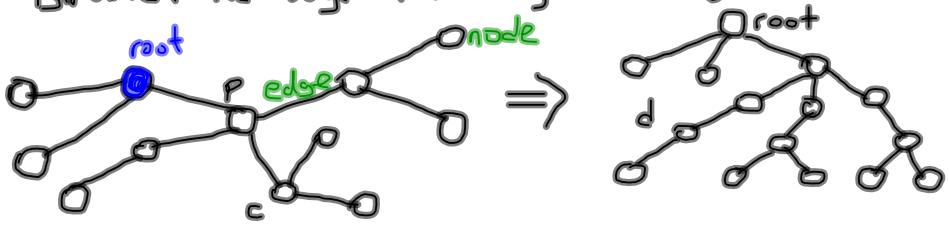
AĞAÇLAR (Trees)

Ağaç: Düğümler ve bunları birbirlerine bağlayan ayrıtlar kümesi,
her iki düğüm (node) arasında sadece bir yol olmalıdır.

Yol : Birbirleri ile bağlantılı ayrıtlar (edge) dizisidir.



Rooted Tree: Bir düğüm root olarak adlandırılır.
Kök hariç, her c düğümün bir p ebeveyni (parent) vardır.
parent: c den köke olan yol üzerindeki ilk düğüm.
c ise p'nin çocuğu (child) dur.

Leaf (Yaprak) : Çocuğu olmayan düğümdür.

Sibling (Kardeş) : Parent'ı aynı olan düğümdür.

Ancestor (Ata): Bir d düğümünün ataları d den köke olan yol üzerindeki tüm düğümlerdir.

Descendant : Çocukları, torunları, ...

Yol uzunluğu : Yol üzerindeki ayrıt sayısıdır.

n düğümün derinliği: n den köke olan yolun uzunluğu
kökün derinliği sıfırdır.

n düğümünün yüksekliği: n den en alttaki descendant düğümüne olan yolun uzunluğudur.

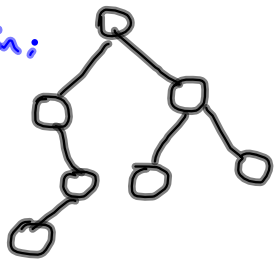
Bir ağacın yüksekliği = kökün yüksekliği

n düğümünde köklenen altağaç (subtree rooted at n):
n ve n'in soyu (descendant) tarafından oluşan ağaç

Binary tree (ikili ağaç): Bir düğümün en fazla 2 çocuğu olabilir, sol çocuk, sağ çocuk.

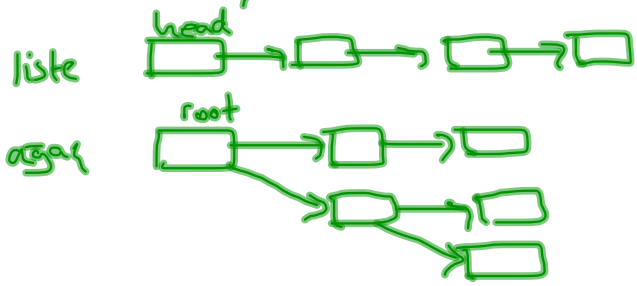
Ağaçların Temsili

1) İkili Ağaçlar için:



```
struct node {
    int data;
    struct node *left;
    struct node *right;
}
```

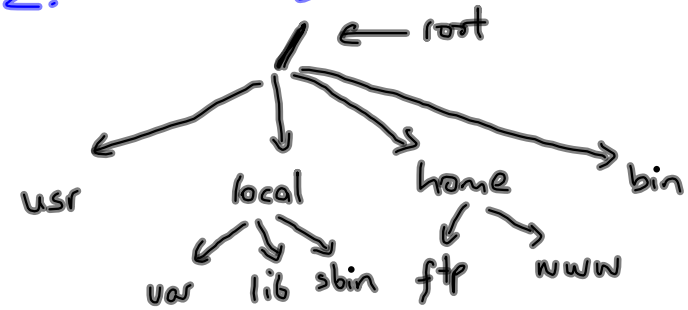
Soru: Ağaçlarda bağlı liste olduğuna göre klasik bağlı listelerden farkı nedir?



linear (doğrusal)

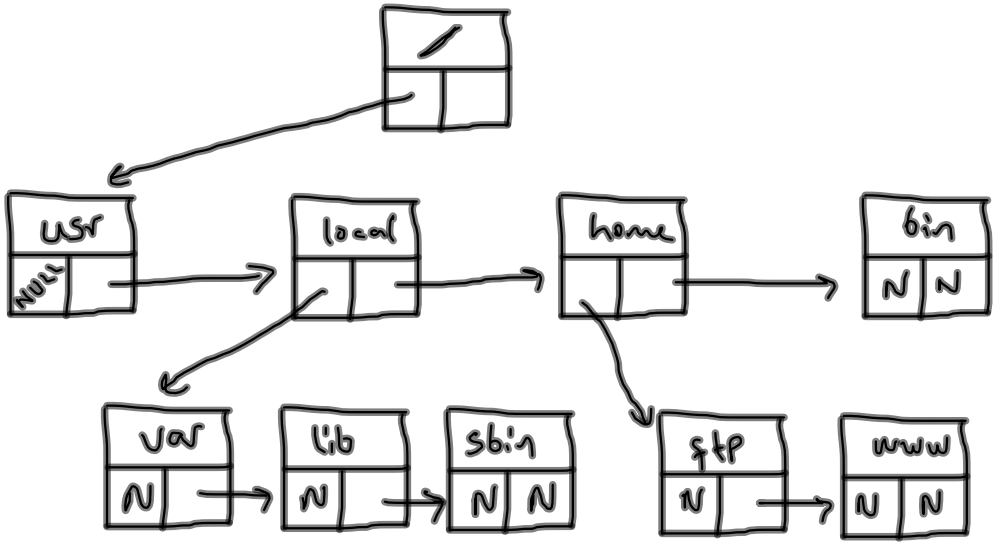
nonlinear (doğrusal olmayan)

2. Tüm Ağaçlar için:




```
struct node {
    int data;
    struct node *firstchild;
    struct node *nextSibling;
}
```

Kardeşler birbirlerine bağlanır.



İkili Ağaçlar Üzerinde Dolaşma

1.) Preorder (Önce kök) Dolaşma: root-left-right

```
typedef  struct node *BTREE;

void  preorder ( BTREE  root){
      if ( root!=NULL) {
          printf ("%d", root->data);
          preorder ( root->left);
          preorder ( root->right);
      }
}
```

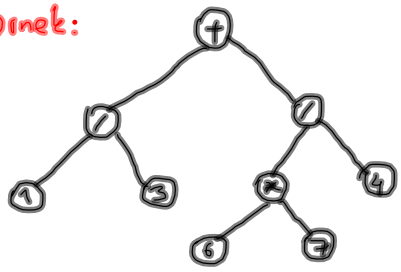
2.) Inorder (Kök arada) Dolaşma: left-root-right

```
void  inorder ( BTREE  root){
      if ( root!=NULL) {
          inorder ( root->left);
          printf ("%d", root->data);
          inorder ( root->right);
      }
}
```

3.) Postorder (Kök sonda) Dolaşma: left-right-root

```
void  postorder ( BTREE  root){
      if ( root!=NULL) {
          postorder ( root->left);
          postorder ( root->right);
          printf ("%d", root->data);
      }
}
```

Örnek:



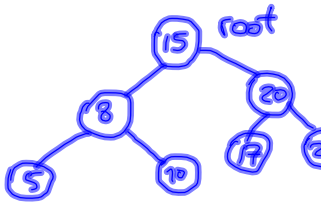
Inorder: left-root-right : 1/3+6*7/4
Preorder: root-left-right : +/13/*674
Postorder: left-right-root : 13/67*4/+

İkili Ağaç Oluşturma

```
/* Yeni bir düğüm oluşturma */
BTREE new_node(int data){
    BTREE p;
    p = (BTREE)malloc(sizeof(struct node));
    p->data = data;
    p->left = NULL;
    p->right = NULL;
    return p;
}
```

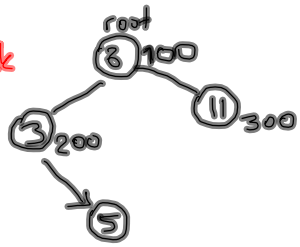
x = new_node(7);


```
/* Ağaca Veri Ekleme */
/* Sol çocuğun verisi ebeveynin datasından küçük olmalı */
/* Aksi halde sağa eklenir */
BTREE insert(BTREE root, int data){
    if(root != NULL){
        if(data < root->data)
            root->left = insert(root->left, data);
        else
            root->right = insert(root->right, data);
    }else
        root = new_node(data);
    return root;
}
```



Binary Search Tree
İkili Arama Ağacı
inorder: 5 8 10 15 17 20

Örnek



insert(100, 5)
↓
100->left = insert(200, 5)
↓
200->right = insert(NULL, 5)
↓
5

Klavyeden -1 girinceye kadar agaca ekle

```
main ( ){
    BTREE myroot=NULL;
    int i=0;
    scanf ("%d", &i);
    while (i!=-1){
        myroot= insert(myroot, i);
        scanf ("%d", &i);
    }
    inorder (myroot);
}
```

10 5 7 3 12 17 9 -1