Lesson 23-24

Objectives

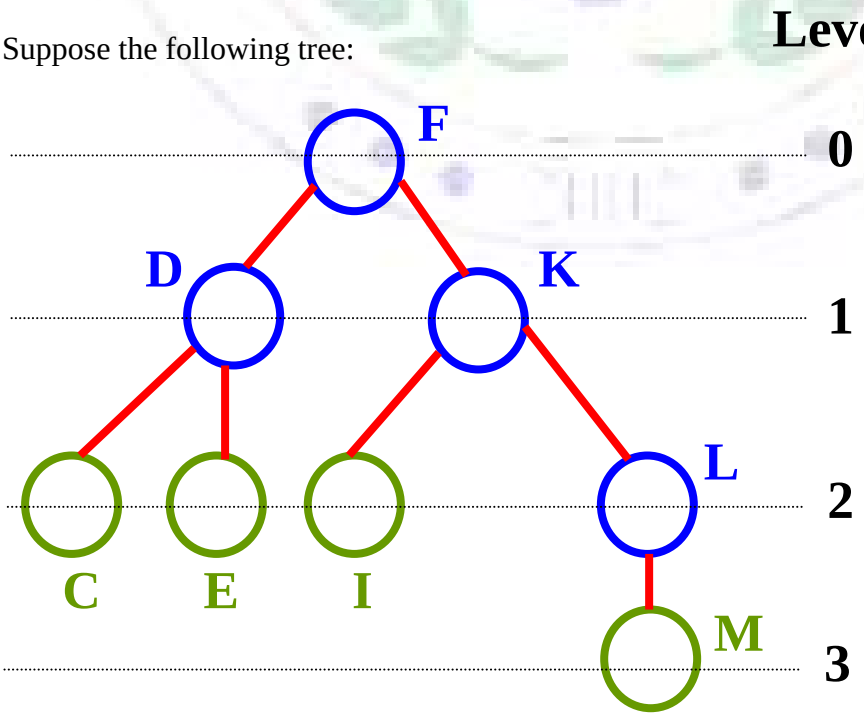
- Tree with Array
  - Introduction of Tree
    - Binary Search Tree Concept in Array
  - Binary Search Tree
  - Binary Search Tree via Array
    - Insertion
      - How to insert in Binary Tree?
      - Insertion
    - Deletion
      - How to delete in Binary Search Tree?
        - Deleting a leaf
        - Deleting a node with one (Right) children
        - Deleting a node with two children
      - Deletion

Tree

- A *tree* is a finite set of one or more nodes such that
  1.There is a specially designated node called *root*.
  2. The remaining nodes *are* called the *sub trees* of the root.
- Tree is a subset of graph but it is never circular.
- Each node must be directly or indirectly connected to the root of the tree.
- Trees have a strict hierarchical structure where one node is root node and all others nodes are its children.

Introduction of Tree:

Suppose the following tree:



F is the *root* node

D is the *parent* of C and E

C is the *sibling* of B

I and *L are* the *children* of K

C, ***E***, ***M***, *I* are *external nodes*, or *leaves*

The ***level*** of *E* is *2*

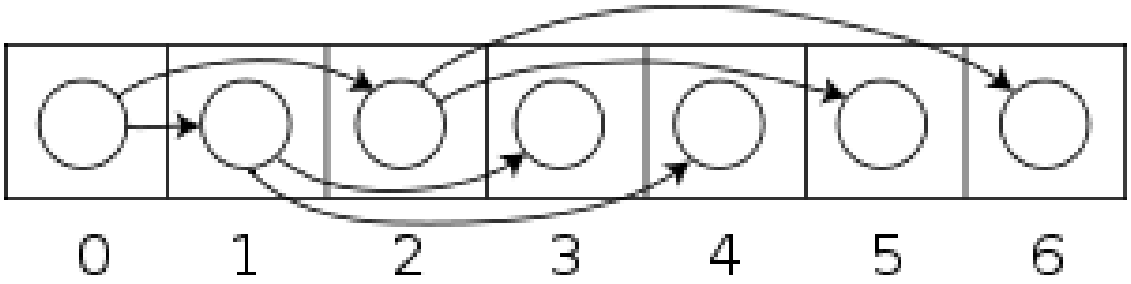
The ***height*** *(depth)* of the tree is *3*

The ***ancestors*** of node *I* is *A, C, H*

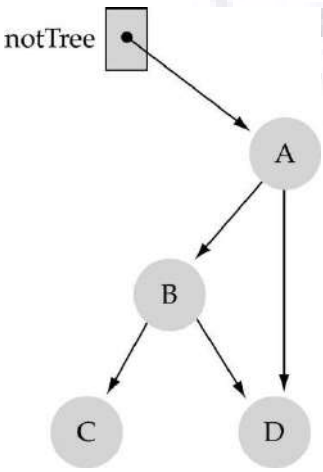
The ***descendants*** of node *C* is *F, G, H, I*

Binary Search Tree Concept in Array:

If a node has an index *i*, its children are found at indices (for the left child) 2i+1 and (for the right) 2i+2.



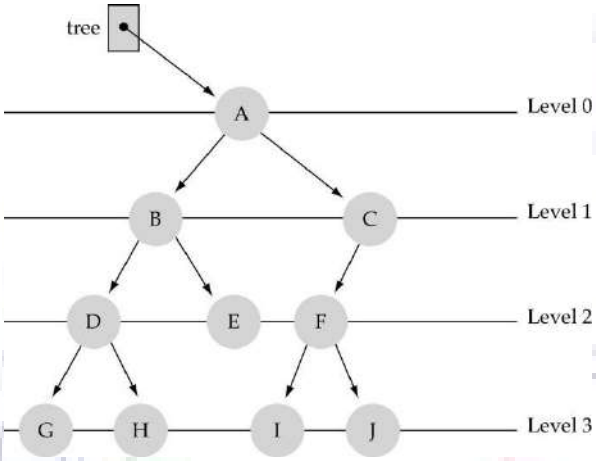
- A unique path exists from the root to every other node.



Not a Valid Binary Tree

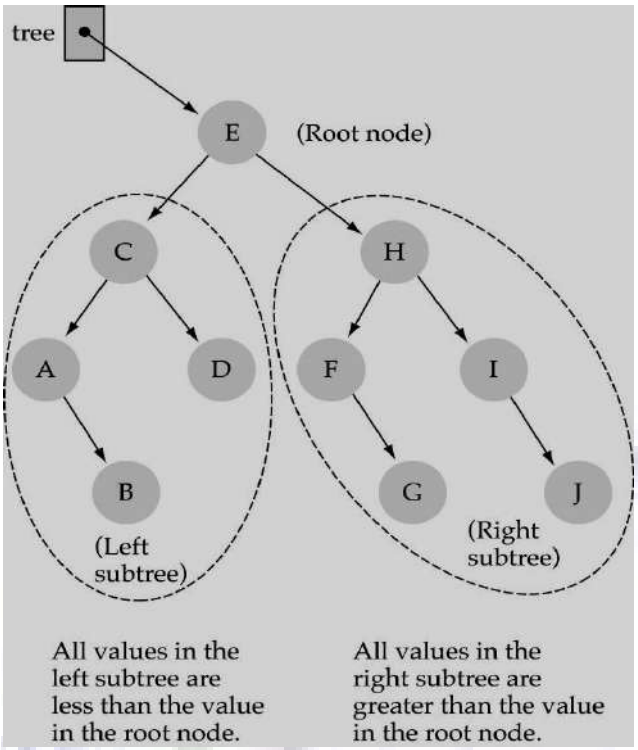
- Nodes are organize in levels (indexed from 0).
- **Level (or depth) of a node**: number of edges in the path from the root to that node.
- **Height of a tree h**: The **height** of a node is the number of edges from the node to the deepest leaf

- **Full tree:** every node has exactly two children ***and*** all the leaves are on the same level.



Binary Search Tree:

- It is a tree in which each node has maximum of two child or sub nodes.
- A Binary Tree has two child at most and BST has smaller values on the left side and larger on the right.
- It is implemented in two ways;
  1. Put the values smaller or equal to root node on the left side of the tree and larger values to the right side of the tree.
  2. Put the values smaller or equal to root node on the right side of the tree and larger values to the left side of the tree.



Binary Search Tree via Array:

How to Insert in BST?

Step1: if the tree is empty,then Root (T) =z.

Step2: Pretending we are searching for zin BST T, until we meet a null node.

Insertion:

For insert we apply the above two steps. The code of insertion in BST is below.

```
void insert(int x, int i)
{
        if (A[i]==-1)
                A[i]=x;
        else
        {
                if(A[i]>x)
                        insert(x,2*1+2)
                else if(A[i]<x)
                        insert(x,2*1+2)
                else
        System.out.println("already inserted");
        }
}
```

How to delete in BST?

First, find the item; then, delete it

Binary search tree property must be preserved!!

We need to consider three different cases:

(1) Deleting a leaf

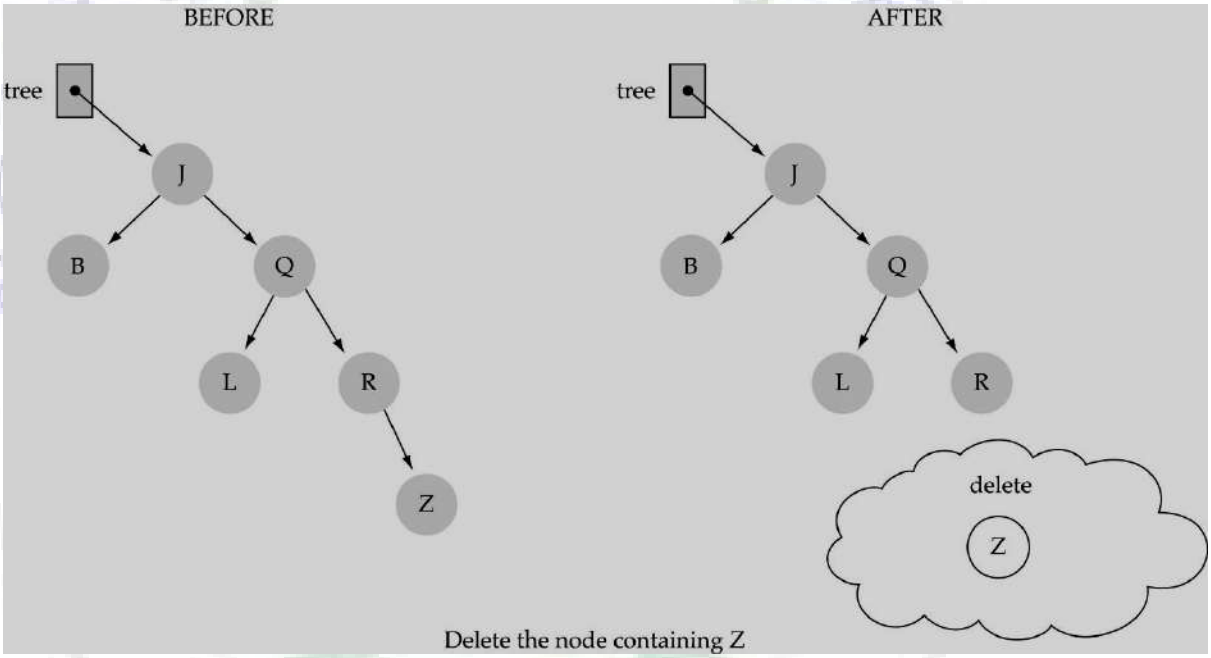
(2) Deleting a node with only one child

(3) Deleting a node with two children

1. **Deleting a node with two children**

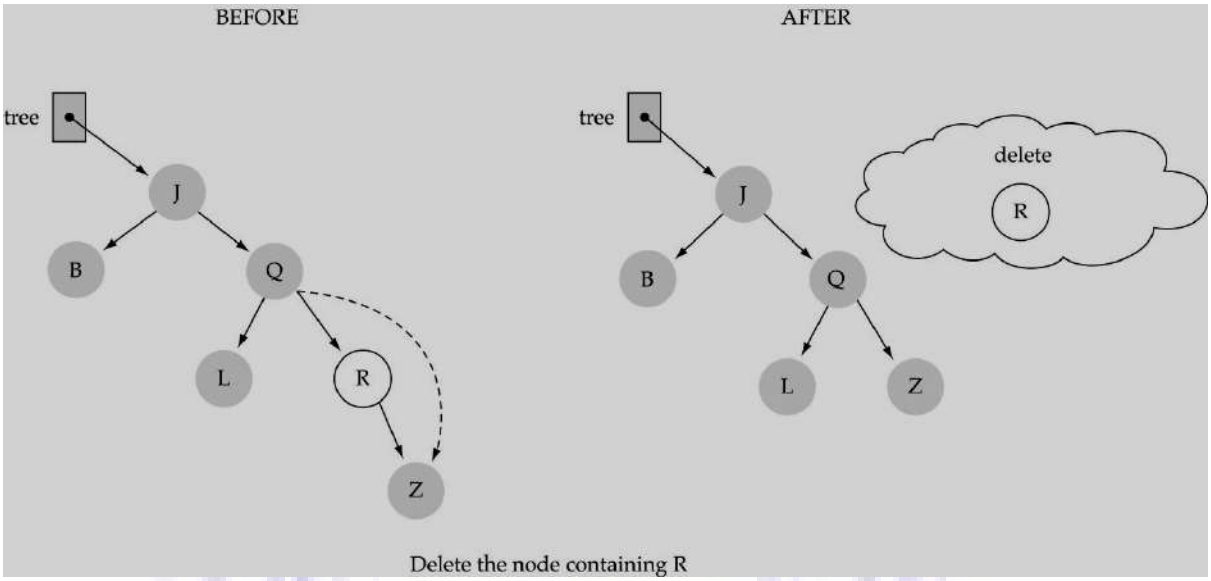
Find and remove it.

For example: We delete Z from the below picture which doesn't have further sub nodes. We'll just set Null to it.

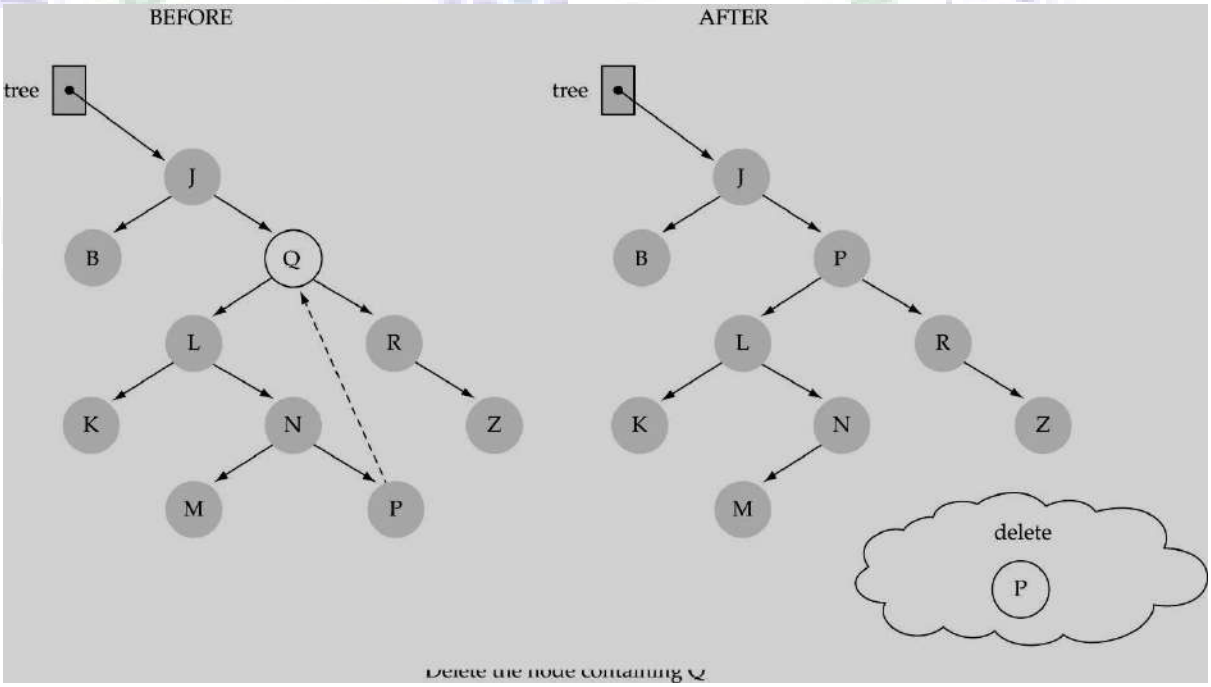


2. **Deleting a node with two children**

- Find predecessor (i.e., leftmost node in the right sub tree)
- Replace the data of the node to be deleted with predecessor's data.
- Delete predecessor node,



3. **Deleting a node with two children**
   - Find predecessor (i.e., rightmost node in the left subtree)
   - Replace the data of the node to be deleted with predecessor's data
   - Delete predecessor node



Deletion:

For deletion we apply the above three conditions to delete anything that we have to delete from the BST via Array. The code of insertion in BST is below.

```
void remove(int x, int i)
```

```
{
        while(a[i]!=-1)
        {
                if(a[i]==x || i>=size)
                break;
                else
                {
                if(a[i]>x)
                                i=i*2+1;
                else if(a[i]>x)
                                i=i*2+1;
                        else
                System.out.println("not found");
                }
        }
        if(i>=size)
        System.out.println("not found");
        else if(i<size)
        {
                if(a[i]==-1)
                System.out.println("not found");
                else
                {
                        if(i*2+1>=size && i*2+2>=size)
                                a[i]=-1;
                        else if(a[i*2+1]=-1 && a[i*2+2]=-1)
                                a[i]=-1;
                        else if(a[i*2+1]=-1 && a[i*2+2]!=-1)
                        {
                                int y=2*i+2;
                                while(a[y]!=-1)
                                        y=y*2+1;
                                y=(y-1)/2;
                                a[i]=a[y];
                                remove(a[y],y);
                        }
```

```
                else if(a[i*2+1]!=-1 && a[i*2+2]==-1)
                {
                        int y=2*i+2;
                        while(a[y]!=-1)
                                y=y*2+1;
                        y=(y-1)/2;
                        a[i]=a[y];
                        remove(a[y],y);
                }
                else if(a[i*2+1]!=-1&& a[i*2+2]!=-1)
                {
                        int y=2*i+1;
                        while(a[z!=-1)
                                z=z*2+2;
                        z=(z-1)/2;
                        a[i]=a[z];
                        remove(a[z],z);
                }
        }
    }

}
```