UNIT - IV: TREES

Syllabus:

Trees: Terminology **Binary Trees:** definition, types of binary trees, Representation, Implementation (linked list), **Tree traversals**: Recursive techniques, Expression Tress, **Search Tree:** Binary Search Tree-search, insert, Delete, **Balanced Tree** –Introduction to AVL tree and Rotations.

Learning Material

Tree is mainly used to represent information level by level.

Tree is a nonlinear data structure.

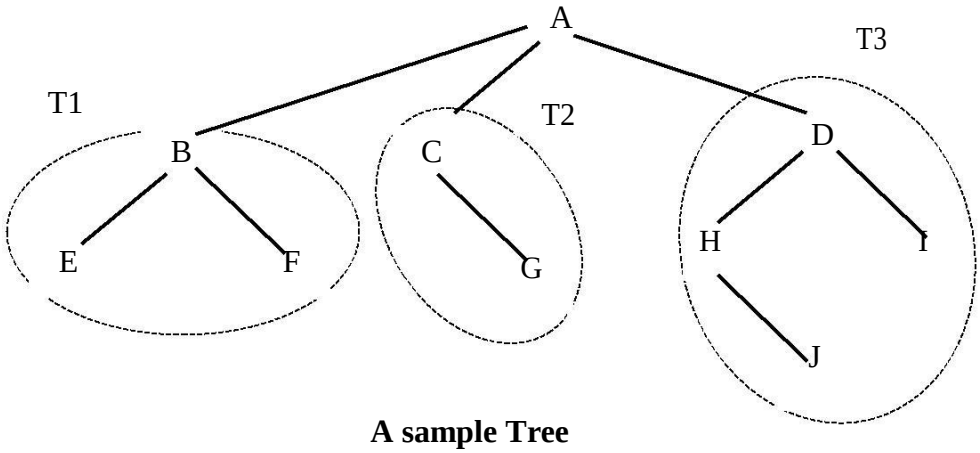
Definition

A tree T is a finite set of one or more nodes such that:

(i) There is a special node called as root node.

(ii) The remaining nodes are partitioned into n disjoint sets T_1, T_2, T_3,. . . T_n. where n>0 and each disjoint set is a tree.

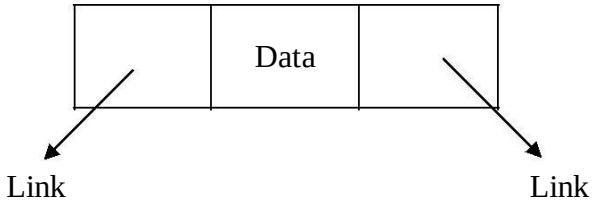
T_1, T_2, T_3. . .T_n are called as sub trees.



A sample Tree

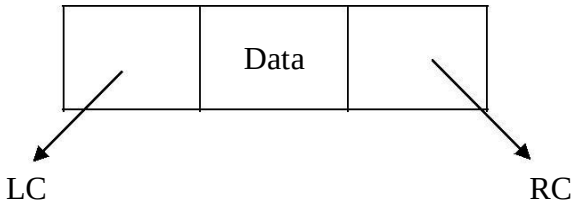
Basic Terminology:

1. Node: Every element of tree is called as a node. It stores the actual data and links to other nodes.



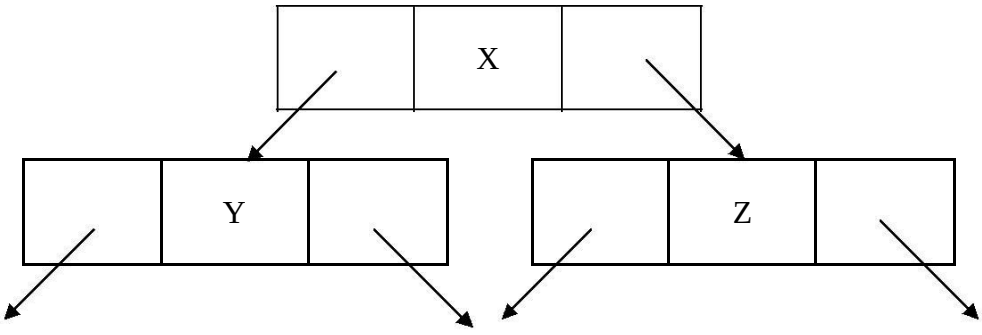
Structure of a node in Tree

2. Link / Edge / Branch: It is a connection between 2 nodes.



Here LC Points To Left Child and RC Points To Right Child.

3. Parent Node: The Immediate Predecessor of a Node is called as Parent Node.

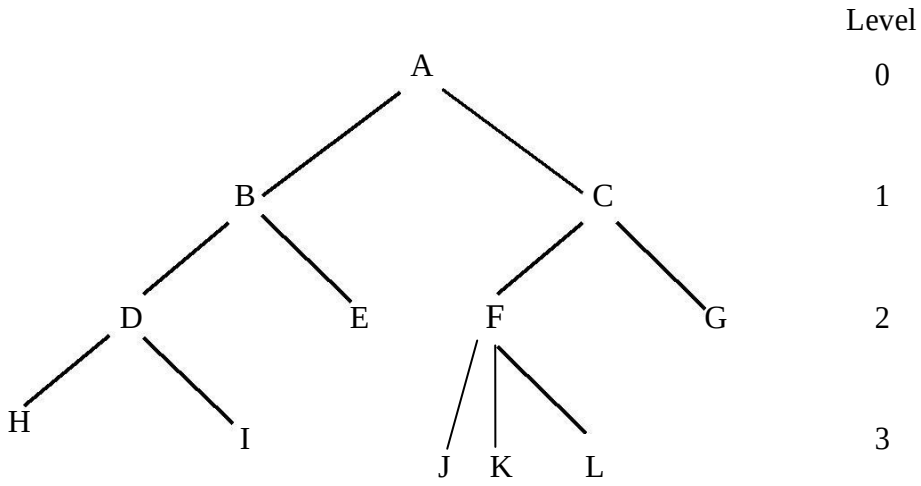


Here X is Parent Node to Node Y and Z.

4.Child Node: The Immediate Successor of a Node is called as Child Node.

In the above diagram Node Y and Z are child nodes to node X.

5. Root Node: Which is a specially designated node, and does not have any parent node.



In the above diagram node A is a Root Node.

6. Leaf node or terminal node: The node which does not have any child nodes is called leaf node. In the above diagram node H, I, E, J, K, L and G are Leaf nodes.

7. Level: It is the rank of the hierarchy and the Root node is present at level **0**. If a node is present at level ***l*** then its parent node will be at the level ***l*-1**and child nodes will present at level ***l*+1**.

8. Siblings: The nodes which have same parent node are called as siblings.

In the above example nodes B and C are siblings, nodes D and E are siblings, nodes F and G are siblings, nodes H and I are siblings.

9. Degree of a node: The number of nodes attached to a particular node.

In the above figure degree of A is 2, degree of F is 3.

10. Degree of a tree: The maximum degree of a node is called as degree of tree.

In the above figure degree of tree is 3.

11.Non terminal node or Internal node: The nodes with child nodes are called as non terminal nodes.

In the above example A,B,C,D,F are internal nodes.

12.Path: It is a sequence of consecutive edges from source to destination nodes.

In the above figure path from A to E is A-B-E.

13.Path length: The number of edges between source and destination nodes.

Ex: A-B-E (path length is 2)
A-C-F-J (path length is 3)

14. Height of a node: It is the length of longest path from the node to leaf. Height of leaf node is zero.

In the above figure, height of A is 3, height of B is 2.

15. Height of a tree:- : It is the length of longest path from the root node to leaf.

In the above figure, height of tree is 3.

16. Depth of a node:- : It is the length of longest path from the root to that node. Depth of root node is 0.

In the above figure, depth of A is 0, depth of B is 1.

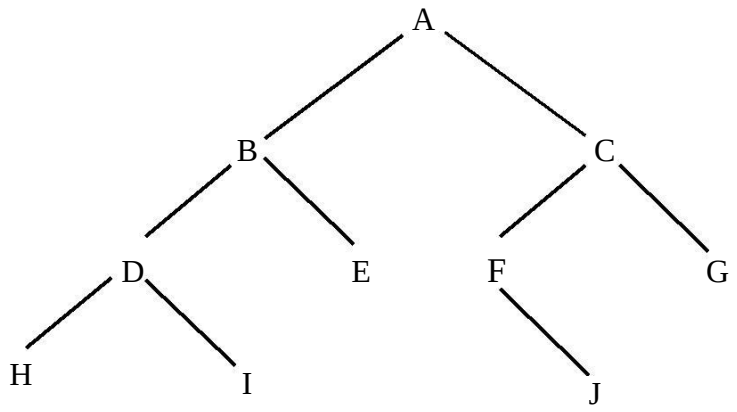
17. Depth of a tree:- : It is the length of longest path from the root node to leaf.

In the above figure, depth of tree is 3.

18. Ancestor and Descendent node:- If there is a path from node A to node B then A is called as Ancestor of B and B is called as descendent of A.

BINARY TREE: Is a special form of a tree.

Definition: A tree in which every node can have a maximum of two children is called as Binary Tree.



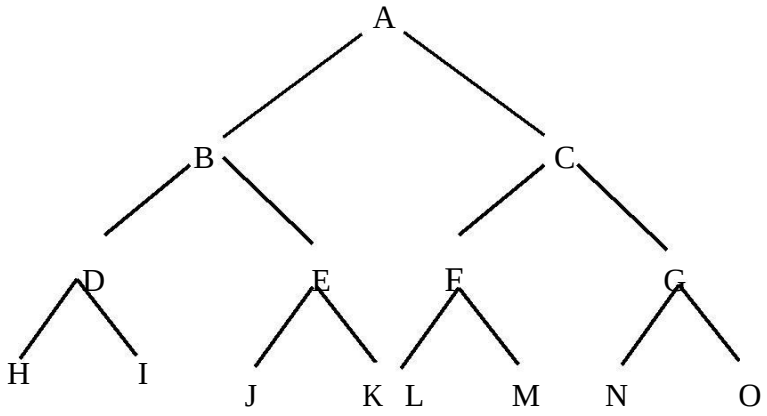
A sample Binary Tree

TYPES OF BINAERY TREE

1. Full binary tree **or** Strictly binary tree
2. Complete binary tree
3. Left Skewed binary tree
4. Right Skewed binary tree
5. Balanced binary tree

1. Full Binary Tree: A binary tree is said to be full binary tree, if every node should have exactly two children or none.

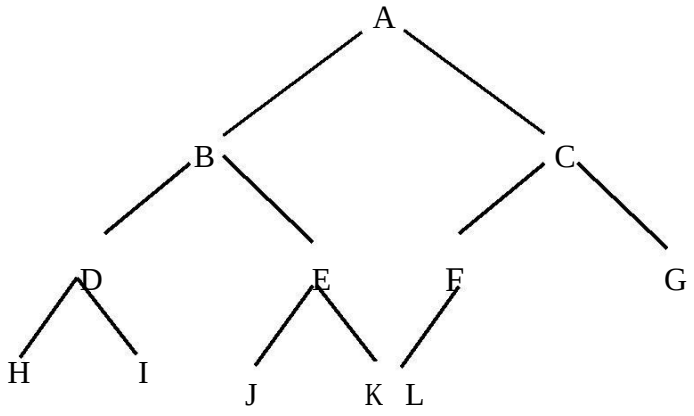
Eg:



A Full Binary Tree

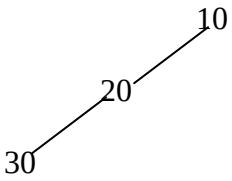
2. Complete Binary Tree: A complete binary tree is a binary tree in which every level except possibly the last level is completely filled and all nodes are from left to right.

Eg:

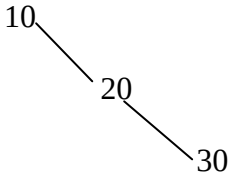


A Complete Binary Tree

3. Left Skewed binary tree: A binary tree in which each node is having only left sub trees is called as left skewed binary tree.



4.Right Skewed binary tree: A binary tree in which each node is having only right sub trees is called as right skewed binary tree.



6. Balanced Binary Tree:
A binary tree is balanced if height of the tree is O(Log n) where n is number of nodes. For Example, AVL tree maintain O(Log n) height by making sure that the difference between heights of left and right sub trees is 1.

Representation of Binary Tree

Binary tree can be represented in two ways.

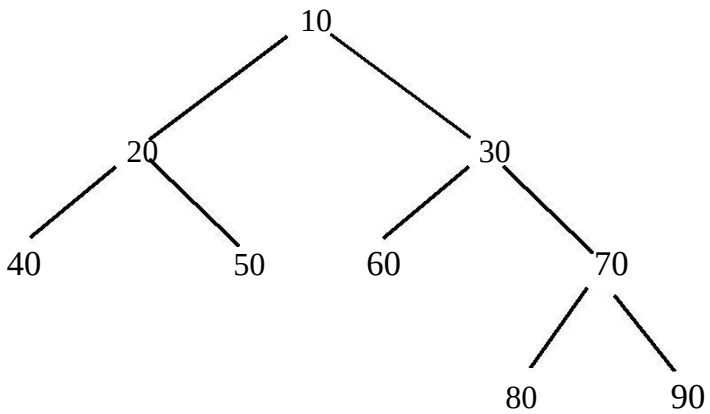
1. Linear (or) Sequential representation using arrays.
2. Linked List representation using pointers.

1. <u>Linear (or) Sequential representation using arrays :-</u>

1. The ROOT node is at index **0**.
2. The left child of tree is stored at 2i +1 where i is the parent index value
3. The right child is stored at 2i+2 location.

.

Eg.



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
10	20	30	40	50	60	70							80	90

Array representation of above binary tree.

Advantages of Linear representation of Binary Tree

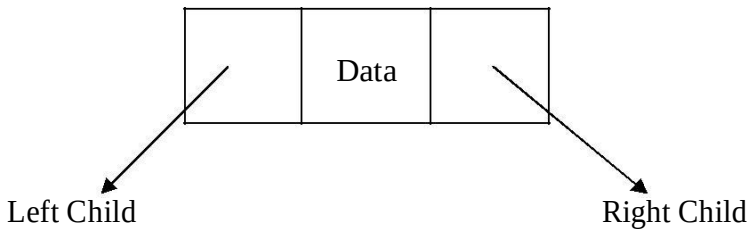
1. Any node can be accessed from any other node by calculating the index and this is efficient from execution point of view.
2. Here only data is stored without any pointers to their successor (or) predecessor.

Disadvantages of Linear representation of Binary Tree

1. Other than full binary tree, majority of entries may be empty.
2. It allows only static memory allocation.

2. Linked List representation of Binary Tree using pointers :-

Linked list representation of assumes structure of a node as shown in the following figure.



With linked list representation, if one knows the address of ROOT node, then any other node can be accessed.

Linked List representation of Binary Tree

Advantages of Linked List representation of Binary Tree

1. It allows dynamic memory allocation.
2. We can overcome the drawbacks of linear representation.

Disadvantages of Linked List representation of Binary Tree

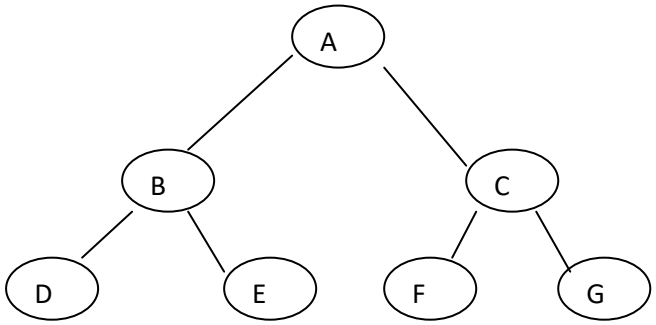
1. It requires more memory than linear representation. i.e. Linked list representation requires extra memory to maintain pointers**.**

Tree Traversals

It is the process of visiting the nodes of a tree exactly once.
A Binary tree can be traversed in 3 ways.

1. Preorder traversal
2. Inorder traversal
3. Postorder traversal

Ex:-



1. Preorder traversal :-

Here first **ROOT** node is visited, then **LEFT** sub tree is visited in Preorder fashion and then **RIGHT** sub tree is visited in Preorder fashion.

i.e. ROOT, LEFT, RIGHT

Function:-

```
void preorder(struct node *temp)
{
        if(temp!=NULL)
        {
                printf("%d \t",temp->key);
                preorder(temp->left);
                preorder(temp->right);
        }
}
```

Ex:- Preorder traversal for the above binary tree is : A B D E C F G

2. Inorder traversal:-

Here first **LEFT** sub tree is visited in Inorder fashion, then **ROOT** node is visited and then **RIGHT** sub tree is visited in Inorder fashion

i.e. LEFT, ROOT, RIGHT

Function:-

```
void inorder(struct node *temp)
{
   if(temp!=NULL)
   {
      inorder(temp->left);
      printf("%d \t",temp->key);
      inorder(temp->right);
   }
}
```

3. Postorder traversal:-

Here first **LEFT** sub tree is visited in postorder fashion, then **RIGHT** sub tree is visited in postorder fashion and then **ROOT** node is visited.

i.e. LEFT, RIGHT, ROOT

Eg:

Function:-

```
void postorder(struct node *temp)
 {
        if(temp!=NULL)
       {
              postorder(temp->left);
              postorder(temp->right);
              printf("%d \t",temp->key);
       }
 }
```

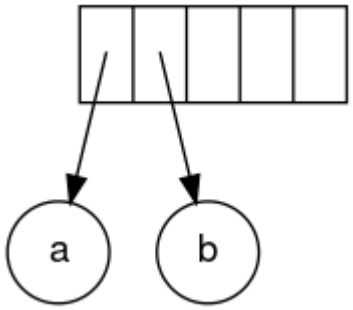
Expression trees:- Expression tree is a binary tree in which each internal node represents an operator and each leaf node represents operand.
Expression trees are constructed based on postfix expression.

Procedure to construct expression tree:-

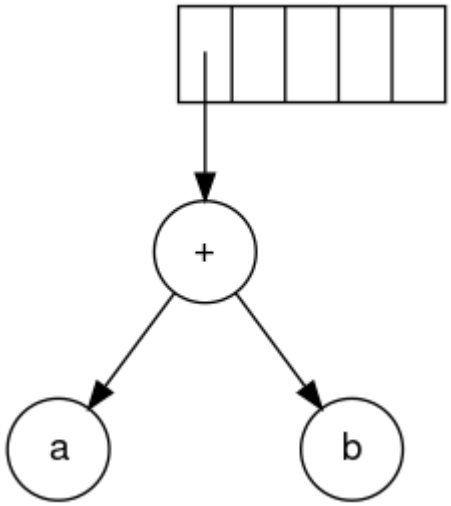
1. Read postfix expression from left to right.
2. If the reading symbol is an operand then create a one node tree and push a pointer (→) to it on to the stack.
3. If the reading symbol is an operator then pop top 2 elements say T_1,T_2 and create a new tree whose parent is an operator, where Left, right children are T_1,T_2 and push a pointer(→) to it on to a stack.

Construct expression tree for a b + c d e + * *:-

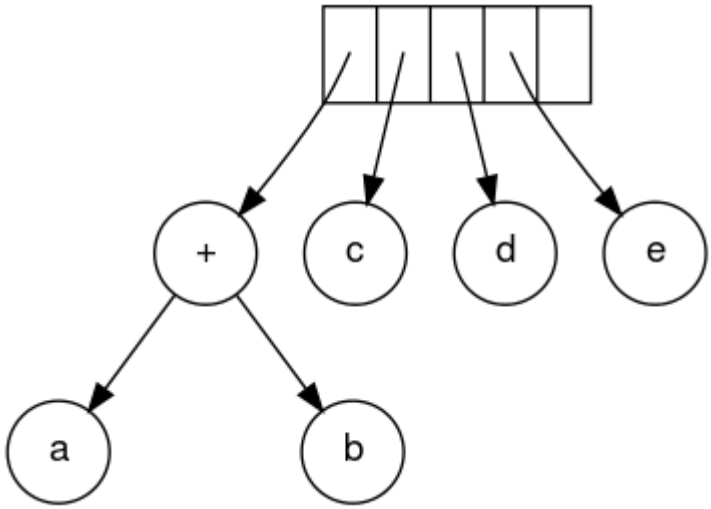
Since the first two symbols are operands, one-node trees are created and pointers are pushed to them onto a stack.



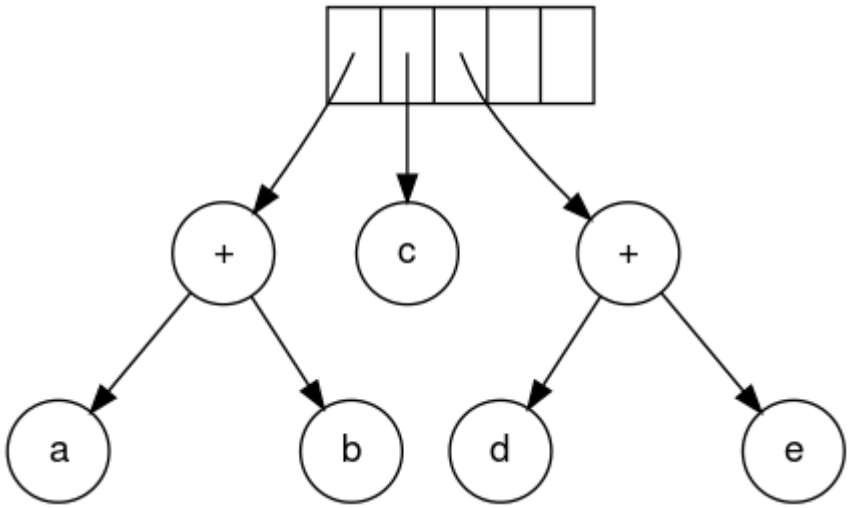
The next symbol is a '+'. It pops the top two pointers to the trees, a new tree is formed, and a pointer to it is pushed onto to the stack.



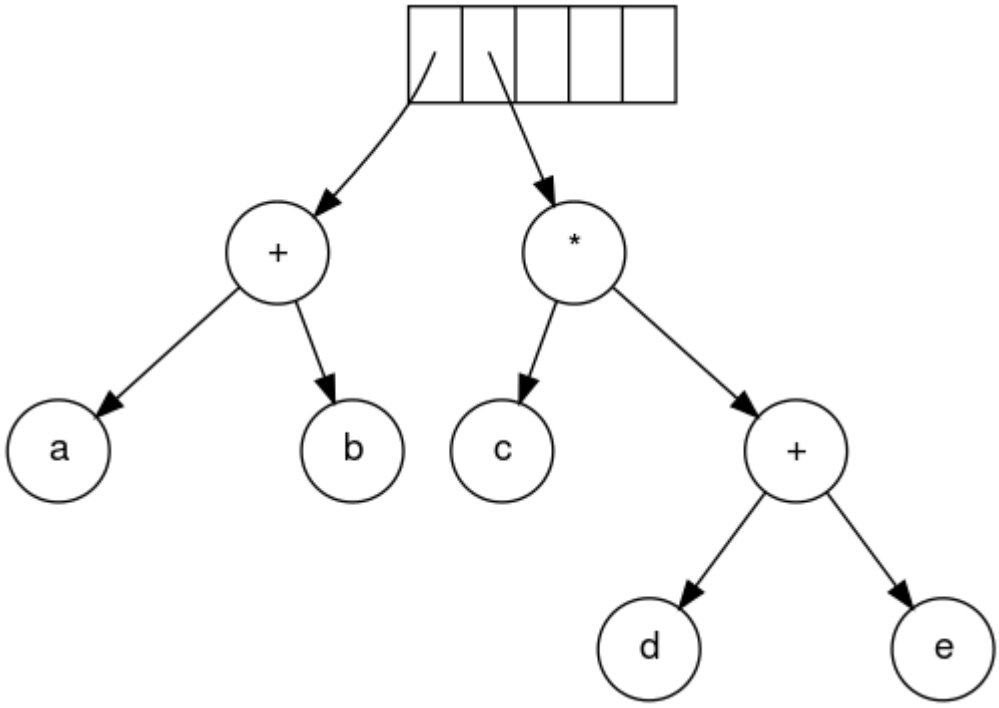
Next, c, d, and e are read. A one-node tree is created for each and a pointer to the corresponding tree is pushed onto the stack.



Next '+' is read, pops the top 2 elements and a new tree is formed with + as parent node.

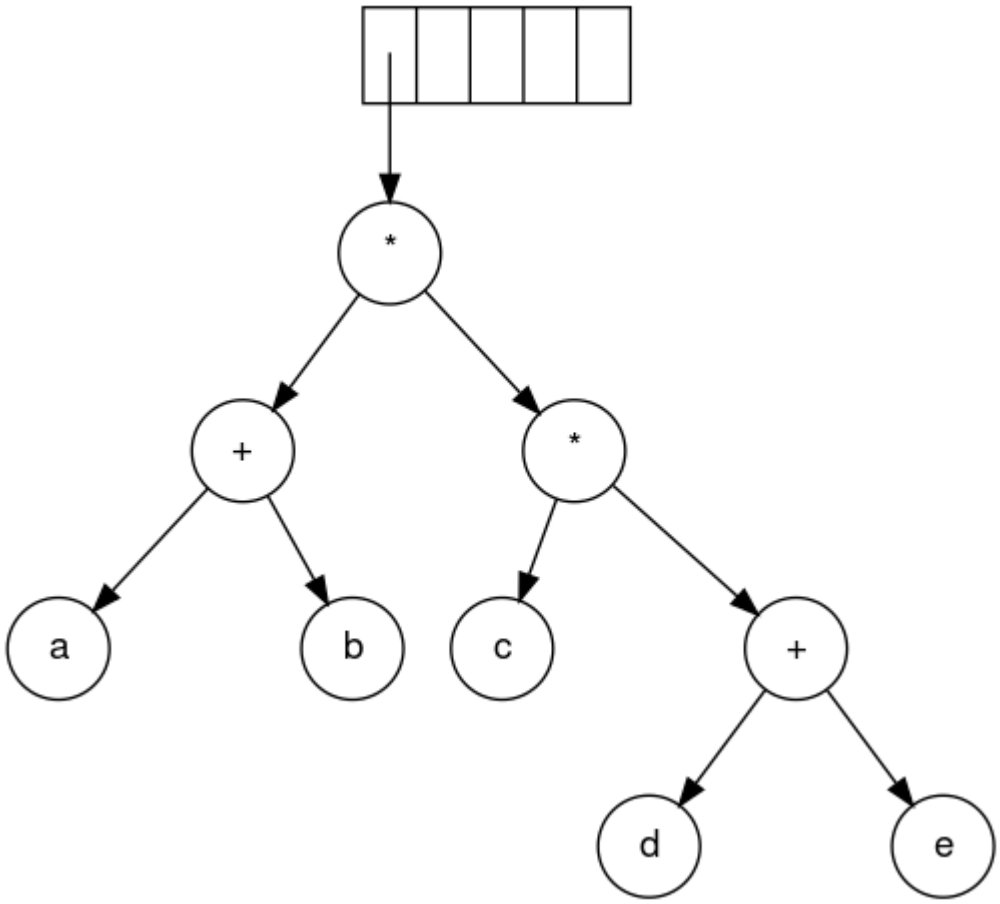


Now, a '*' is read. The last two tree pointers are popped and a new tree is formed with a '*' as the root.



Forming a new tree with a root

Finally, the last symbol is read. The two trees are merged and a pointer to the final tree remains on the stack.



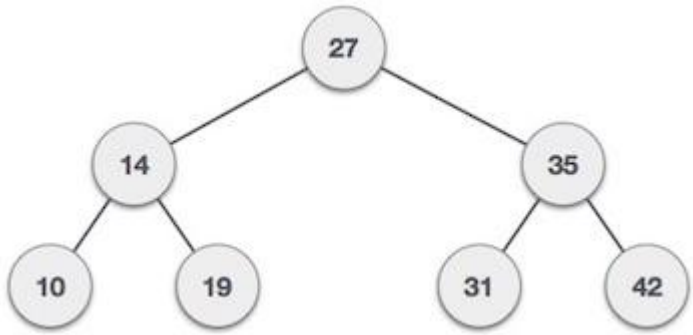
Binary Search Trees (BST):-

A Binary Search Tree is a binary tree in which the left sub tree contains the values which are less than the parent node and the right sub tree contains the values which are greater than the parent node.

To make the searching process faster, we uses binary search tree.

Binary Search Tree is a combination of binary tree and binary search.

Example of Binary Search Tree:-



Binary Search Tree Operaions:-

1. Insertion
2. Find min
3. Find max

4. Search
5. Delete

1. Insertion:- This Operation is used to insert an element into a Binary Search Tree.

Case 1: When tree is empty

(i) Create new node.

(ii) Make new node's left and right field as NULL i.e. new→left=new→right=NULL.

NULL		NULL

(iii) Store value in new node's data field i.e. new→data=value.

NULL	Value	NULL

Case 2:- When tree is not empty

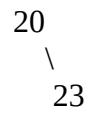
(i) Insert value in root left if value<root→data

(ii) Insert value in root right if value>root→data

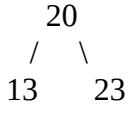
Ex:-Construct binary search tree for the elements 20, 23, 13, 9, 14, 19, 21, 27 and 24.

Insert 20 into the Binary Search Tree.
Tree is not available. So, create root node and place 20 into it.

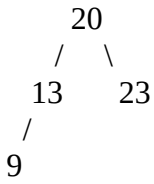
Insert 23 into the given Binary Search Tree. 23 > 20 (data in root). So, 23 needs to be inserted in the right sub-tree of 20.



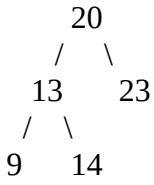
Insert 13 into the given Binary Search Tree. 13 < 20(data in root). So, 13 needs to be inserted in left sub-tree of 20.



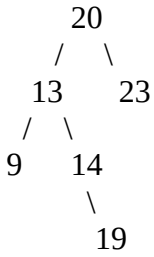
Insert 9 into the given Binary Search Tree.



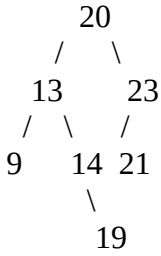
Inserting 14 into the given Binary Search Tree.



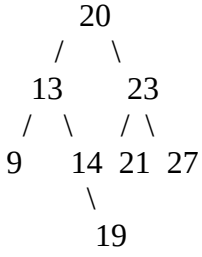
Inserting 19 into the given Binary Search Tree.



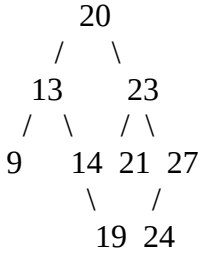
Inserting 21 into the given Binary Search Tree.



Inserting 27 into the given Binary Search Tree.



Inserting 24 into the given Binary Search Tree.



2.Find min:-

This operation returns the address of the smallest element of the tree when the tree is not empty. To find the minimum element, we start at root node and we go left as long as there is a left child. The stopping element is the smallest element of the tree.

Function:-

```
struct node *find_min(struct node *root)
{
        if(root==NULL)
                return root;
        else if(root->left==NULL)
                return root;
        else
                return find_min(root->left);
}
```

3.Find max:-

This operation returns the address of the biggest element of the tree when the tree is not empty. To find the maximum element, we start at root node and we go right as long as there is a right child.
The stopping element is the biggest element of the tree.

Function:-

```
struct node *find_max(struct node *root)
{
        if(root==NULL)
                return root;
        else if(root->right==NULL)
                return root;
        else
                return find_max(root->right);

}
```

4.Search:-

Searching for a node is similar to inserting a node. We start from root, and then go left or right until we find (or not find the node). A recursive definition of search is as follows. If the root is equal to NULL, then we return root. If the root is equal to key, then we return root. Otherwise we recursively solve the problem for left sub tree or right, depending on key.

Function:-

```
struct node * search(struct node *root,int key)
{
        if(root==NULL)
                return root;
        else if(key==root->data)
                return root;
        else if(key<root->data)
                search(root->left,key);
        else if(key>root->data)
                search(root->right,key);

}
```

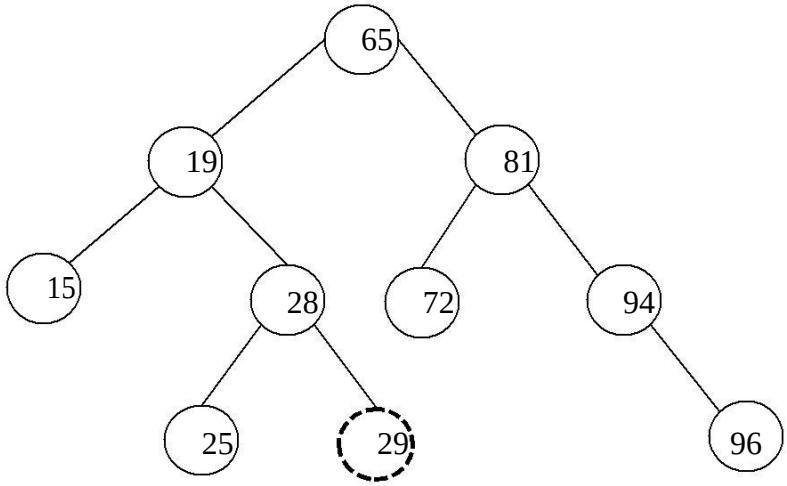
5.Delete:-

There are three different cases that needs to be considered for deleting a node from binary search tree.

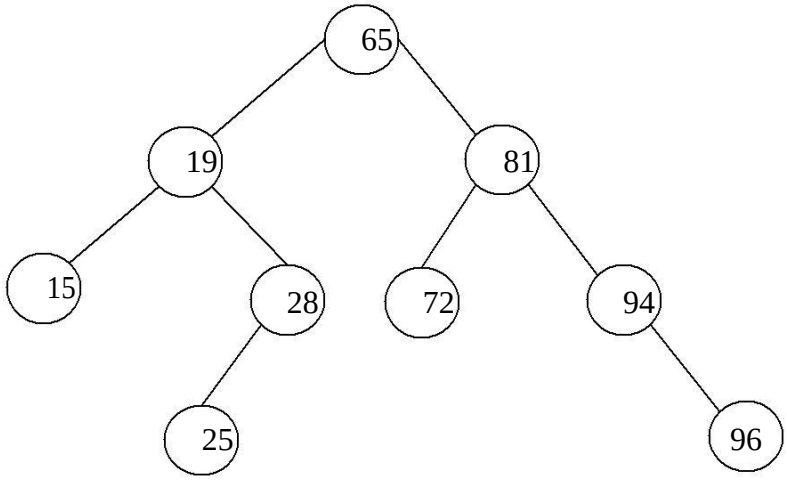
case 1: Delete a leaf node
case 2: Delete a node with one child
case 3: Delete a node with two children.

Case 1: Delete a leaf node:- This case is quite simple. Set corresponding link of the parent to NULL and dispose that node.

1



Deletion of node 29

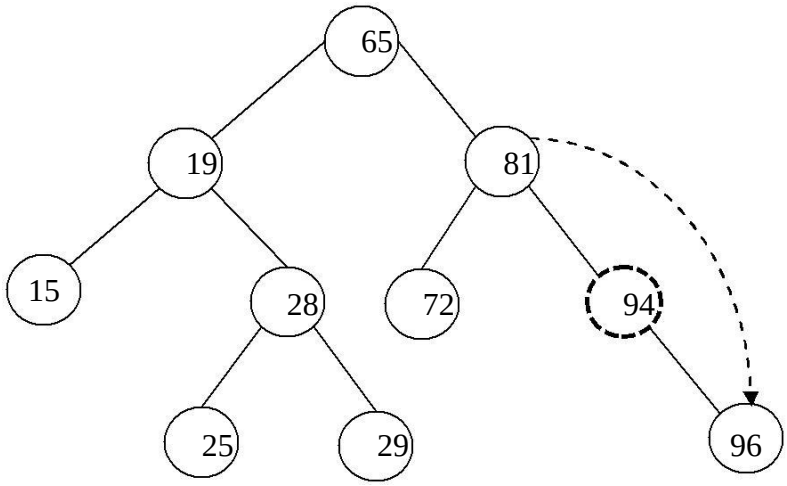


After deletion of node 29 form given BST

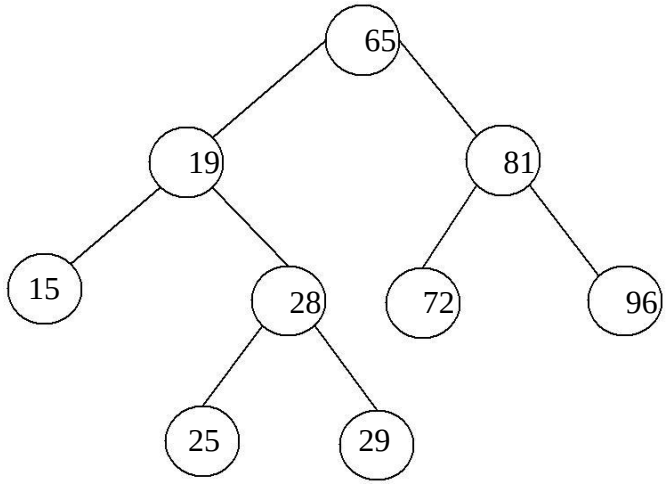
case 2: Delete a node with one child:- In this case, node is deleted from the tree and links single child (with it's sub tree) directly to the parent of the removed node.

If the node to be deleted is a left child of the parent, then we connect the left pointer of the parent (of the deleted node) to the single child. Otherwise if the node to be deleted is a right child of the parent, then we connect the right pointer of the parent (of the deleted node) to single child.

.

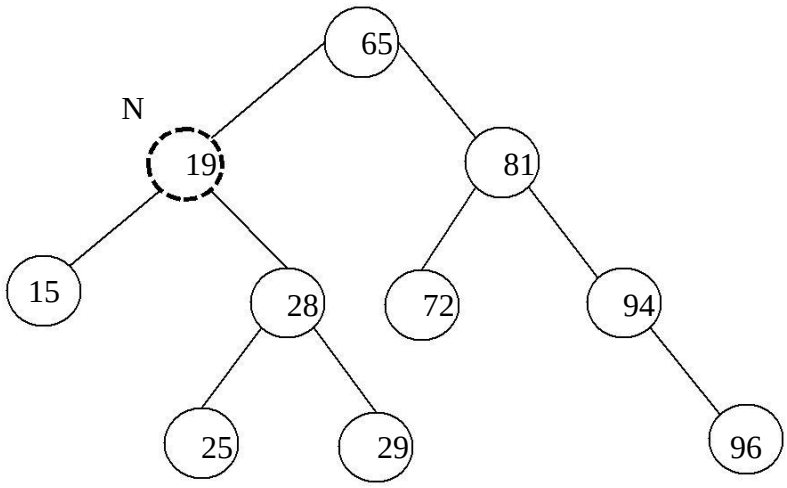


Deletion of node 94

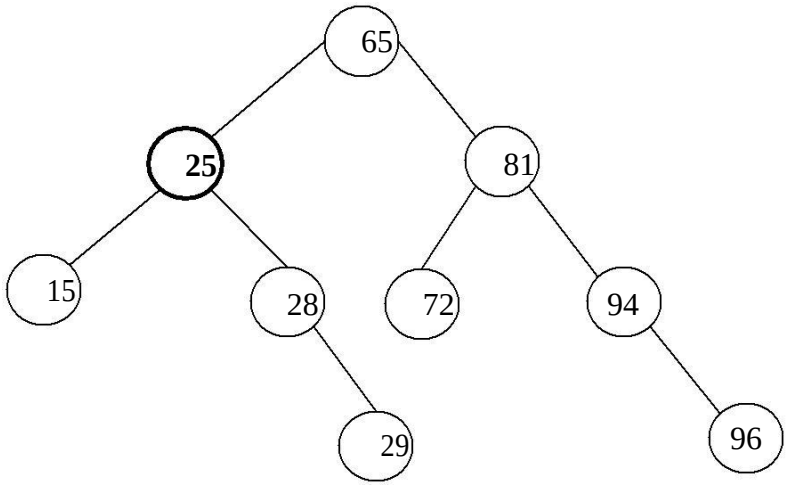


After deletion of node 94 form given BST

Case 3: Delete a node with two children:-If the node to be deleted is N, then find the minimum value in the right sub tree of N and replace deleted node N with minimum value.



Deletion of node 19



After Deletion of node 19

Function:-

```
struct node *delete(struct node *root,int ele)
{
        struct node *temp;
        if(root==NULL)
                return root;
        else if(ele<root->data) /*if the node to be deleted <root key then it lies in left sub tree*/
                root->left=delete(root->left,ele);
        else if(ele>root->data)/*If the node to be deleted is > root then it lies in right sub tree*/
                root->right=delete(root->right,ele);
        else  /*If key is same as root key then this is the node to be deleted*/
        {
                if(root->left==NULL)  /* node with only one child or no child */
                {
                        temp=root;
```

```
                root=root->right;
                free(temp);
        }
        else if(root->right==NULL)
        {
                temp=root;
                root=root->left;
                free(temp);
        }
        else  /*node with 2 Childs*/
        {
                temp=find_min(root->right);  /* find min value of right sub tree */
                root->data=temp->data;       /*replace root node with min value */
                root->right=delete(root->right,temp->data); /*remove the node */
        }
    }
    return root;
}
```

Ex:- Construct BST by inserting 25,10,12,15,39,64,53, and then delete 15, 10, and insert 45, 23, and 30.