Chapter 6 :- Trees

TYPES OF TRESS

A Tree is defined as a finite s et of one or more elements(nodes) such that,

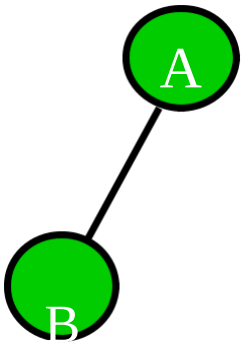
1.There is a one special Node called Root(R).

2.The remaining nodes are di vided into N(N>0) disjoint sets, named as T1 ,T2....Tn , such that T1 is itself a tree.

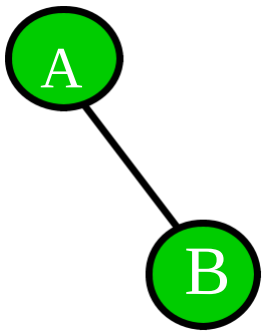
3.Each element of a tree is known as node. The sets of T1,T2....Tn are call ed sub-tree of Root(R).

Special Node called ROOT.

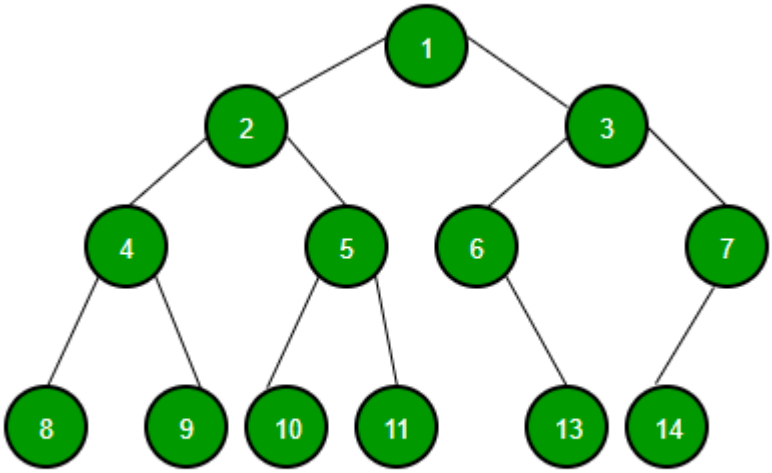
Left sub-tree



Right sub-treee



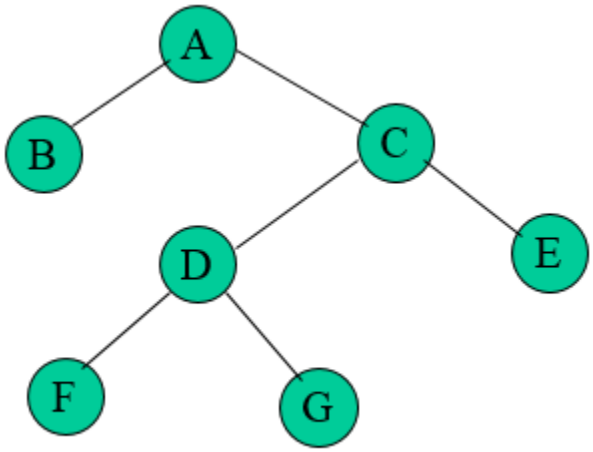
Binary Tree: A tree is called binary tree, if each and every node can have most two



branches.

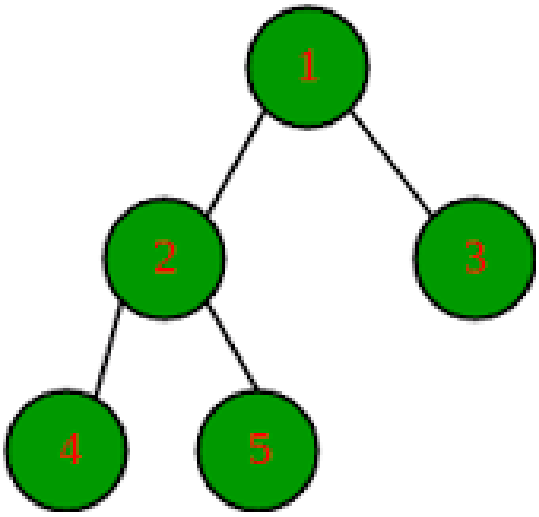
Strictly Binary Tree:

If every non-leaf node in a binary tree has non-empty left and right sub-tree is called strictly binary tree.



Complete Binary Tree:

A binary tree is calle d complete binary tree, if all its levels exce pt last have maximum no. of possible nodes.



Pre-order Traversal

Three steps for Pre-order trav ersal:

1).Process root node first,

2).Traverse left sub-tree of ro ot in pre-order(left),

3).Traverse right sub-tree of r oot in pre-order(Right).

So, pre-order traversal is perf ormed by through ROOT-LEFT-RIGHT in s equence.

Example:

Consider a binary tree with 12 nodes.

Step 1:First process root node :[A]

Step 2:Traversing left sub-tre e in pre-order

2-3-4-12

Step 3:Traversing right sub-tree in pre-order

5-6-9-10-11-7-8.

So, the linear list of node is:

2-3-4-12-5-6-9-10-11-7-8.

Algorithm: PREORDER(T)

```
Step 1:if(T<>NULL)
        then write(DATA(T))
        else write("Empty or Null Tree")
        return.
Step 2:if(LPTR(T)<>NULL)
        then call PREORDER(LPTR(T))
Step 3:if(RPTR(T)<>NULL)
        then call PREORDER(RPTR(T))
Step 4:Return.
```

IN-ORDER Traversing

The In –order Traversing foll ows the following steps:

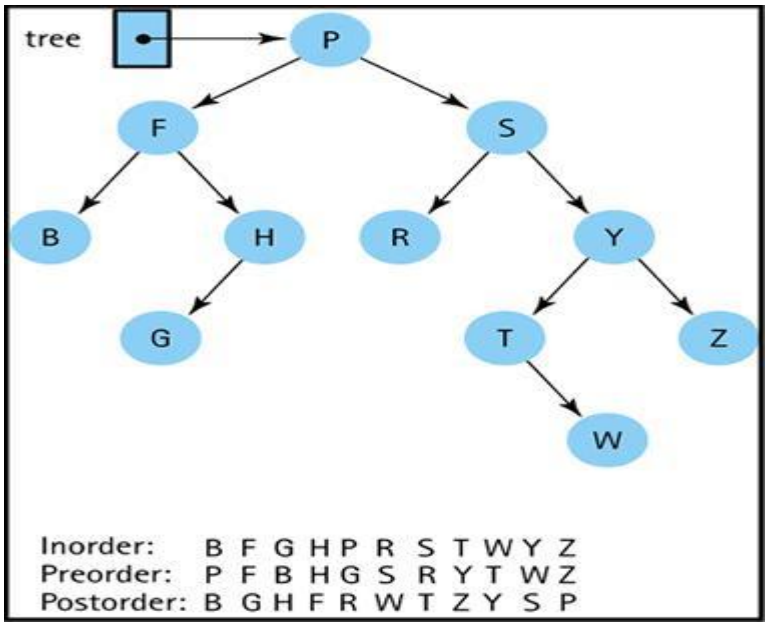
(a) Traverse the left sub tree of root node in-order (left) first,

(b) Process the root node

(c) Traverse the right sub tree of root in-order.

Example :

Left use consider the binary tree of fig. to traversing this Binary tree in-order,



- Step 1: Traverse the left sub tree in order

  B–F–H-G

- Step 2: Process the root node

  A

- Step 3: Traverse the right sub tree in-

  order R–S–T–M–Y–T–W–Z

  After completion of in-order traversing of binary tree, we get linear list of mode : B,F,G,H,P,R,S,Y,T,W,Z [Pre-order traversing]

Algorithm: INORDER(T)

Step 1:if T=NULL

then write("Empty Tre e")

Return.

Step 2:if(LPTR(T)<>NULL)

then call INORDER(L PTR(T))

Step 3:write(DATA(T)).

Step 4:if(RPTR(T)<>NULL)

then call INORDDER (RPTR(T))

Step 5:Return.

Tree Traversing by : PO ST-ORDER Traversal

The Pot-order Traversing foll ows the following steps:

1. Traverse the left sub tr ee of root node in post order first,
2. Traverse the right sub tree of root node in post order,
3. Process the root node.

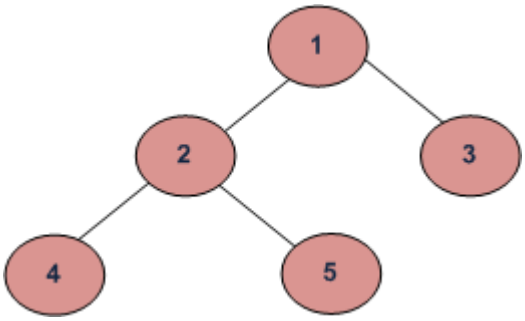
So, past-order traversal is performed by through LEFT-RIGHT-ROOT in sequence.

Example:

Step 1: Traverse the left s ub tree Post-order

Step 2: Traverse the right sub tree Post-order

Step 3: Process the root n ode



The answer is:4-5-2-3 -1

ALGORITHM: POSTO RDER(T)

Step 1: if T=NULL then

Write (“Empty tree”).

Return

Step 2: if (LPTR(T)<>NULL)

Then call POSTORDE R(LPTR(T))

Step 3: if (RPTR(T)<>NULL)

Then call POSTORDE R(RPTR(T))

Step 4: Write (DATA(T))

Step 5: Return.

BINARY SEARCH TREE

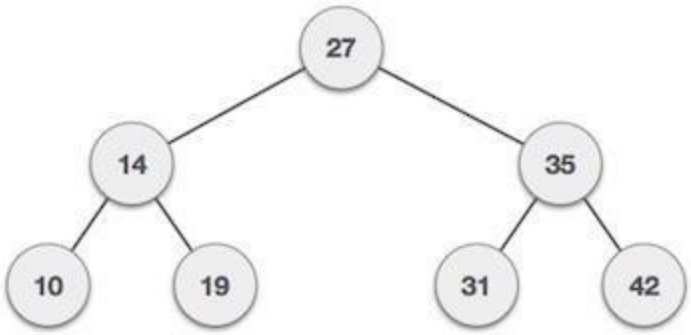
A Binary Search Tree (BST) is a tree in which all the nodes follow the below-mentioned properties –

- The left sub-tree of a node has a key less than or equal to its parent node's key.
- The right sub-tree of a node has a key greater than to its parent node's key.

Thus, BST divides all its sub-trees into two segments; the left sub-tree and the right sub-tree and can be defined as –

left_subtree (keys) ≤ node (key) ≤ right_subtree (keys)

While searching, the desired key is compared to the keys in BST and if found, the associated value is retrieved. Following is a pictorial representation of BST −



Following operations are performed on Binary trees,

1. Insertion
2. Deletion
3. Searching
4. Traversal

1. INSERTION :-

- Inserting a value in the correct position is similar to searching because we try to maintain the rule that left subtree is lesser than root and right subtree is larger than root.

- We keep going to either right subtree or left subtree depending on the value and when we reach a point left or right subtree is null, we put the new node there.

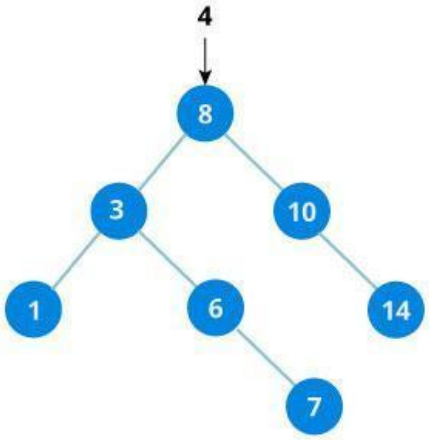
ALGORITHM :

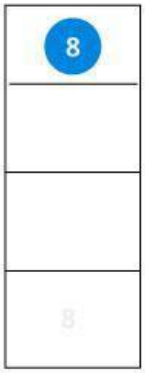
1. If node == NULL Return
   createNode(data)

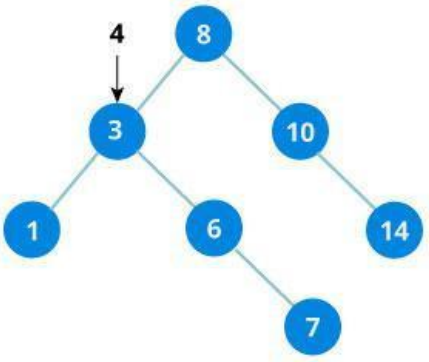
2. If (data < node data)
   Node left = insert(node left, data);

3. Else if (data > node data)

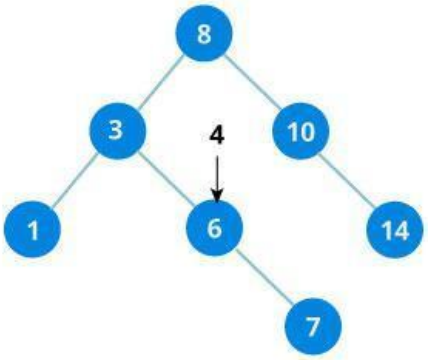
   Node right = insert(node right, data);
   Return node

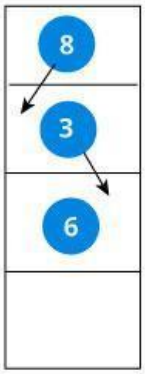

4
8
3
10
1
6
14
7

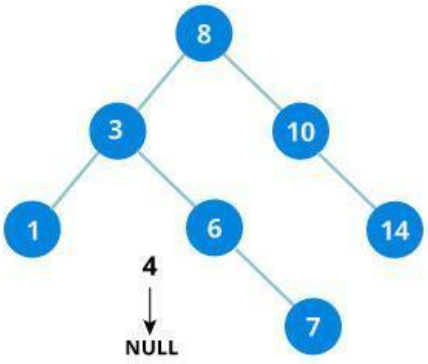

8

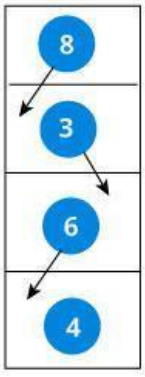

4
8
3
10
1
6
14
7


8
3

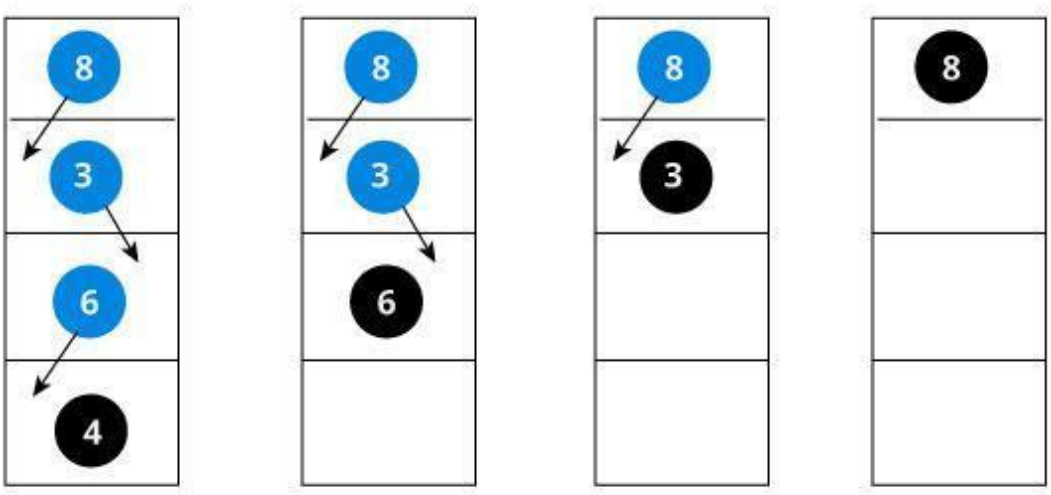

8
3
4
10
1
6
14
7


8
3
6


8
3
10
1
6
14
4
NULL
7


8
3
6
4

We have attached the node but we still have to exit from the function without doing any damage to the rest of the tree.
This is where the return node; at the end comes in handy.
In the case of NULL, the newly created node is returned and attached to the parent node, otherwise the same node is returned without any change as we go up until we return to the root.

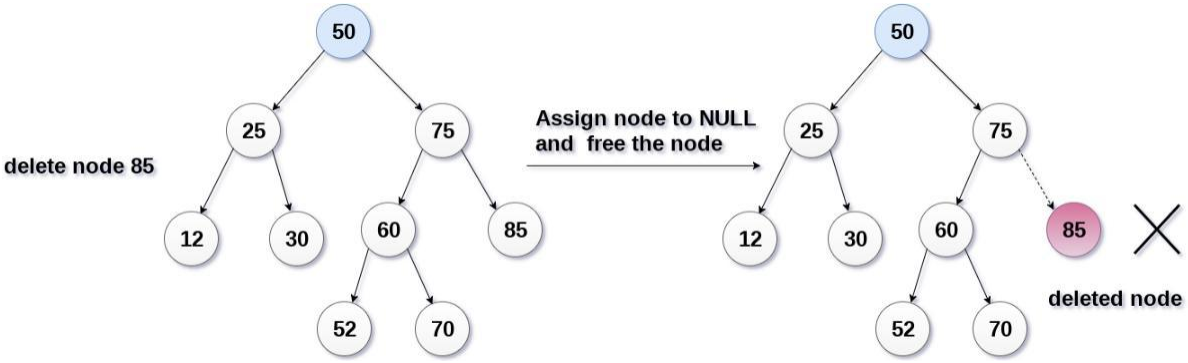


2.DELETION

Delete function is used to delete the specified node from a binary search tree.
However, we must delete a node from a binary search tree in such a way, that the property of binary search tree doesn't violate.
There are three situations of deleting a node from binary search tree.

THE NODE TO BE DELETED IS A LEAF NODE

It is the simplest case, in this case, replace the leaf node with the NULL and simple free the allocated space.
In the following image, we are deleting the node 85, since the node is a leaf node, therefore the node will be replaced with NULL and allocated space will be freed.



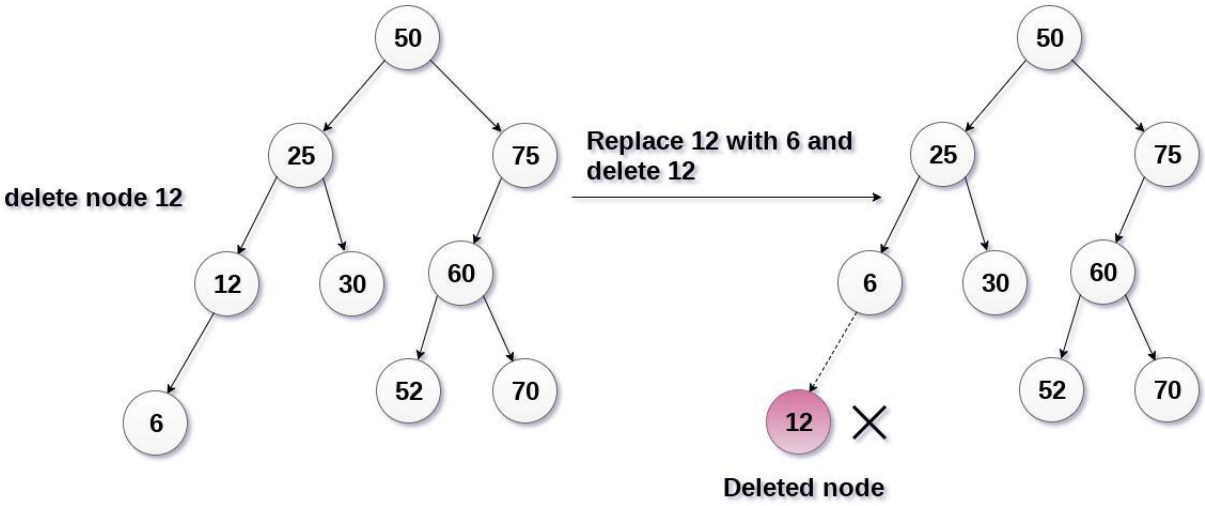
THE NODE TO BE DELETED HAS ONLY ONE CHILD

In this case, replace the node with its child and delete the child node, which now contains the value which is to be deleted.

Simply replace it with the NULL and free the allocated space.

In the following image, the node 12 is to be deleted. It has only one child.

The node will be replaced with its child node and the replaced node 12 (which is now leaf node) will simply be deleted.



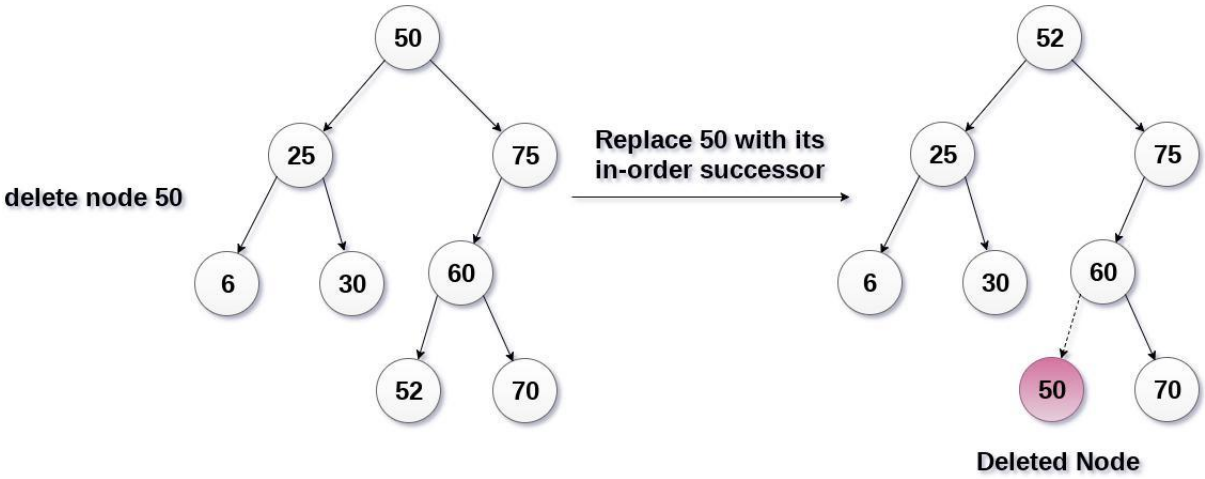
THE NODE TO BE DELETED HAS TWO CHILDREN

It is a bit complex case compare to other two cases. However, the node which is to be deleted, is replaced with its in-order successor or predecessor recursively until the node value (to be deleted) is placed on the leaf of the tree. After the procedure, replace the node with NULL and free the allocated space.

In the following image, the node 50 is to be deleted which is the root node of the tree. The in-order traversal of the tree given below.

6, 25, 30, 50, 52, 60, 70, 75.

replace 50 with its in-order successor 52. Now, 50 will be moved to the leaf of the tree, which will simply be deleted.



ALGORITHM:

delete (tree, item)

Step 1: if tree = null

write "item not found in the tree" else if item < treedata
delete(tree left, item)

else if item > tree data
delete(tree right, item)
else if tree left and tree right
set temp = findlargestnode(tree left)
set tree data = temp data
delete(tree left, temp data)
else
set temp = tree
if tree left = null and tree right = null
set tree = null
else if tree left != null
set tree = tree left
else
set tree = tree right
[end of if]
free temp
[end of if]

Step 2: end