Binary Search Tree

A Binary Search Tree may be empty. If it is not empty then it satisfies the following properties.

(1) Every Element has a key i.e key(x) for element x and no two elements have the same key.

(2) The keys in the left subtree are smaller than the key in the root.

(3) The keys in the right subtree are larger than the key in the root.

(4) The left and right subtree are also binary search trees.

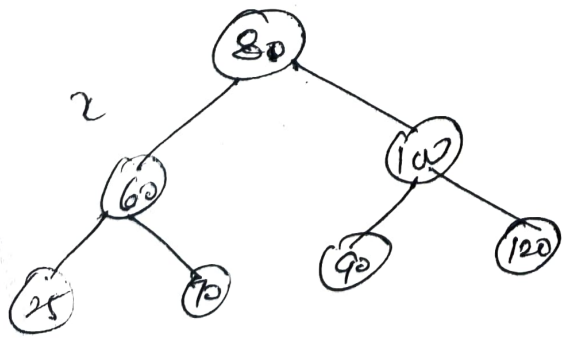
Let x be a node in BST.

- if y is a node in the left subtree of x then

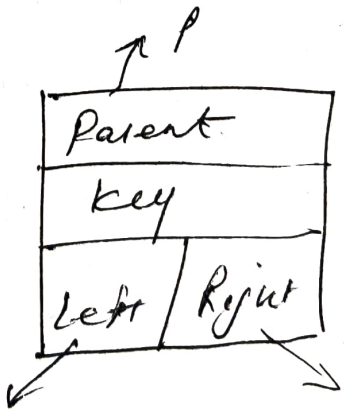
$$key[y] \leq key[x]$$

- If y is a node in the right subtree of x then

$$key[x] \leq key[y]$$



A BST



Node Structure

In the tree key[x] = 60.

If y is the node in the left subtree then key[y] = 25.

ie key[y] ≤ key[x]

and if y is the node in the right subtree then key[y] = 70

ie key[y] ≥ key[x]

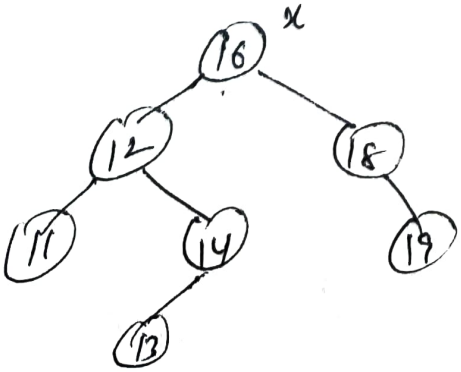
Various operations on BST

(a) Searching in a BST

Tree_Search (x, k)

1. If x == NULL or k = key[x]
2. then return x
3. If k < key[x]
4. then return Tree-Search (left[x], k)
5. Else return tree-Search (right[x], k)

Search operation takes time O(h), where h is the height of a BST.



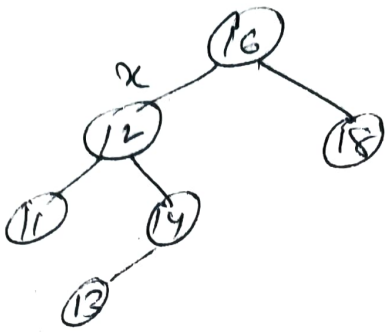
Suppose Tree_Search (x, 14)

Here x ≠ NIL and key[x] = 16 which is not equal to k.

Here k = 14 and 14 < 16 (True)

then Tree_Search (left(x), k)

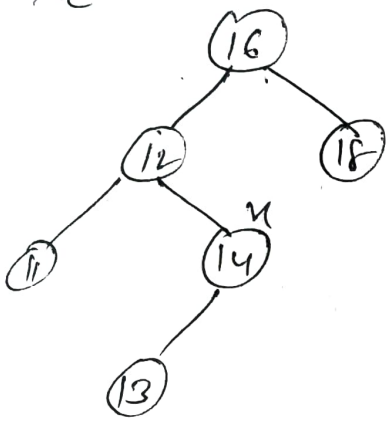
Now left[x] becomes x ('12 node)



Now $x \neq NIL$ and $14 \neq 12$

$14 \nless 12$

So, Tree_Search (right[x], 14)



Now, $x \neq NIL$ and $k = key[x]$

So return x.

(2) Minimum & Maximum key Element in BST

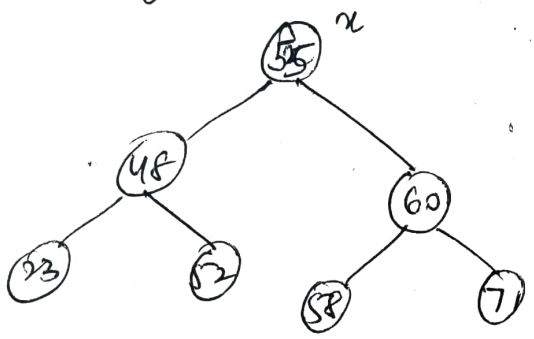
The minimum key value element can always be found by following left child pointers from the root until a NIL is encountered.

```
Tree_Minimum(x)
1. while left[x] ≠ NIL
2. do x → left[x]
3. return x.
```

```
Tree_Maximum(x)
1. while Right[x] ≠ NIL
2. do x → Right[x]
3. Return x
```

(3) Tree-Successor (x).

1. If right [x] ≠ NIL
2. then return Tree_minimum (right [x])
3. $y \leftarrow p[x]$
4. while $y \neq$ NIL and $x = right[y]$
5. do $x \leftarrow y$
6. $y \leftarrow p[y]$
7. return y



. we called Tree_Successor (55)

Here right [x] ≠ NIL

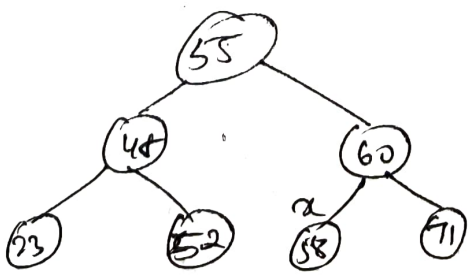
So we call Tree_minimum (right [x])

ie Tree_minimum (60)

Here left [x] ≠ NIL

So, left (x) is now, ie $x \leftarrow$ left (x)

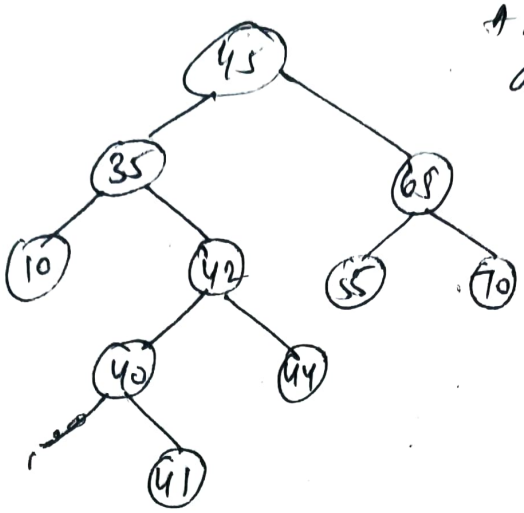
and return it, it node 58



Thus, 58 node is the successor of node 55.

Traversal Operations

All the traversal operations are applicable in BST.



The inorder traversal of the BST →

10, 35, 40, 41, 42, 44, 45, 55, 65, 70

It gives the increasing order of the numbers in the BST.

(I) **Insertion of data into a Binary Search Tree.**

To insert a new value w into a binary search tree T we use the procedure TREE_INSERT. The procedure is passed a node z for which key[z] = w left[z]=NIL and Right[z] = NIL.

```
1.  y ← NIL
2.  x ← root[T]
3.  while x ≠ NIL
4.      do y ← x
5.          if key[z] < key[x]
6.              then x ← left[x]
7.          else x ← right[x]
8.  p[z] ← y
9.  if y = NIL
10. then root[T] ← z
```

11. else if key[z] < key[y]
12. then left[y] ← z
13. else right[y] ← z

Algorithm:

(1) Start from the root node.

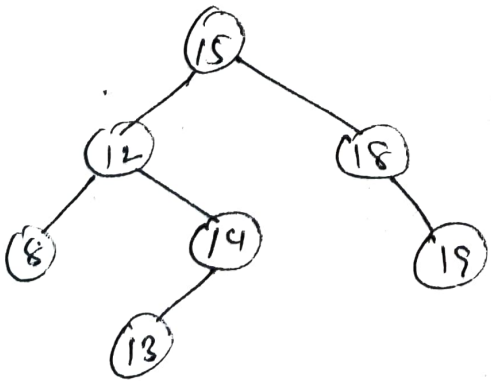
(2) If the data to be inserted is w, compare this with the value of the root node.

(i) If they are equal, just stop because this value of the root node.

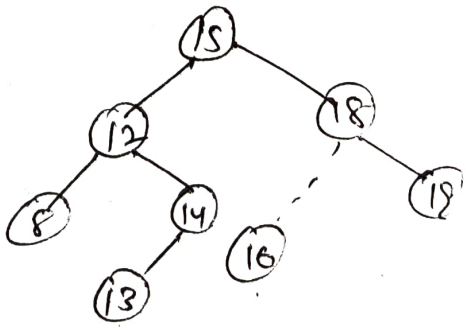
(ii) If they are different and if the data to be inserted is less than the value of the root node, choose the left sub-tree.

(iii) If the data to be inserted (w) is greater than the value of the root node, choose the right sub-tree.

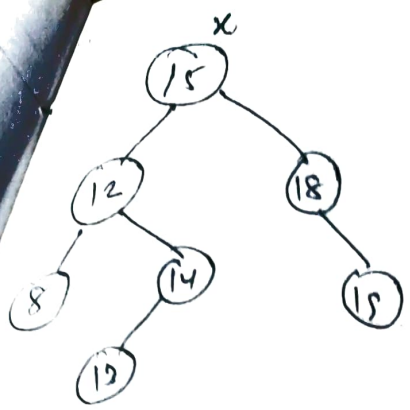
(3) Repeat step(2) until the leaf node is encountered where the data to be inserted.



BST before insertion



BST after insertion



key[z] = 16

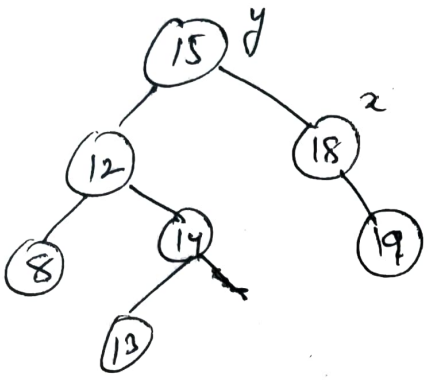
key[x] = 15

while $x \neq NIL$

$y \leftarrow x$

Here, key[z] > key[x] i.e. 16 > 15

so $x \leftarrow right[x]$



Now; $y \leftarrow x$

key[x] = 18 key[z] = 16

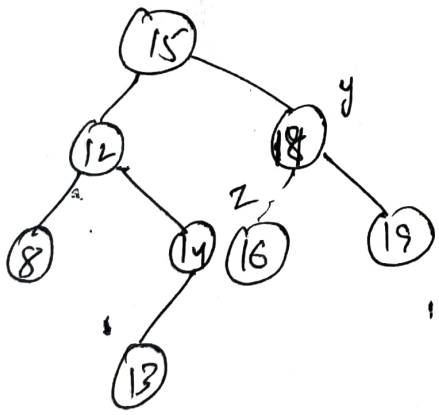
Here key[z] < key[x]

so $x \leftarrow$ left[x]

and here left[x] = NIL

z = NIL

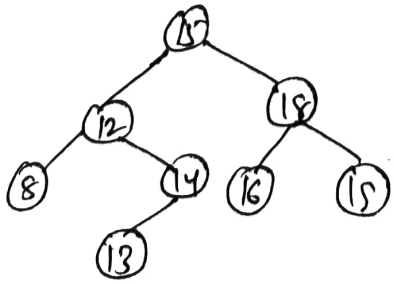
$P[z] \leftarrow y$ i.e.



Here $y \neq NIL$ and key[z] < key[y]

So, z is left child of y

and final tree after insertion



Structure of BST

```
Struct bstree node
{
   int info;
   Struct bstree node *left;
   Struct bstree node *right;
};
Struct bstree node *bt, *root;
```

Deletion of node from BST

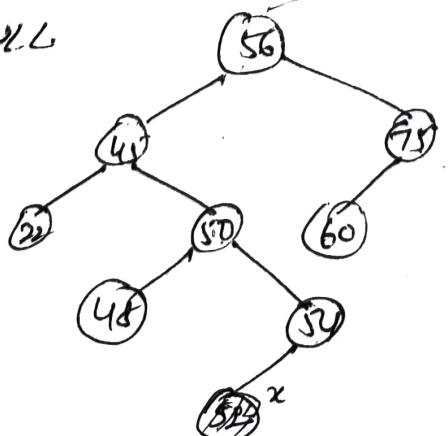
Deletion of a node from BST depends on the number of its children. Suppose delete a node with key = z from BST T. There are 3 cases that can occur.

Case 1 : z has no child (ie leaf node)

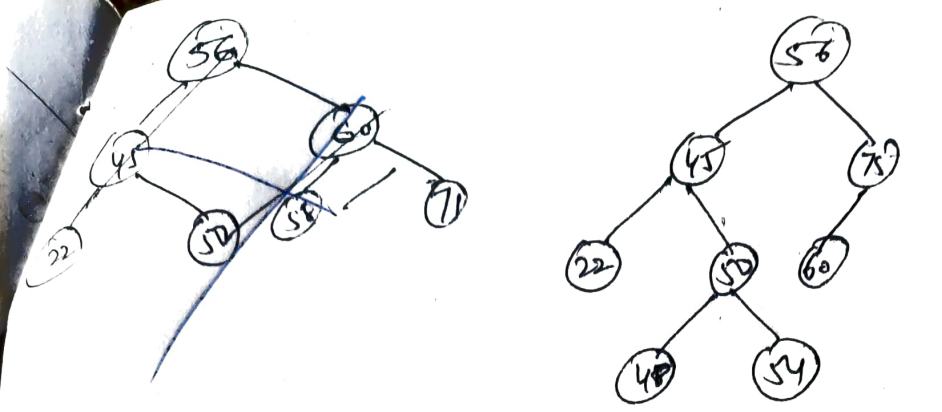
Case 2 : z has exactly one child

case 3 : z has two child.

Case 1 : If the node to be deleted z is leaf node simply make the pointer field of the parent node of z is NULL

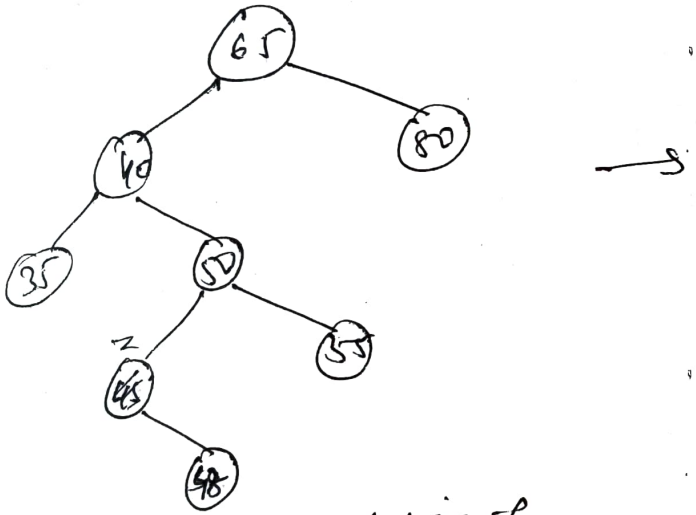


Before deletion of node z

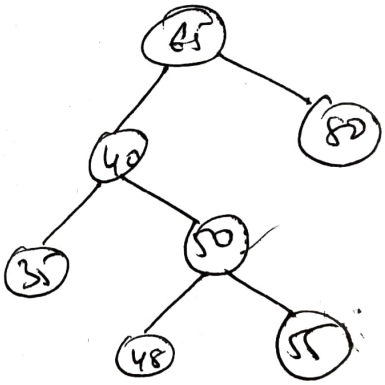


After deletion of node 22.

Case 2: If the node to be deleted z has a child then simply set the pointer field of the parent node of z to point to the only child node of z.



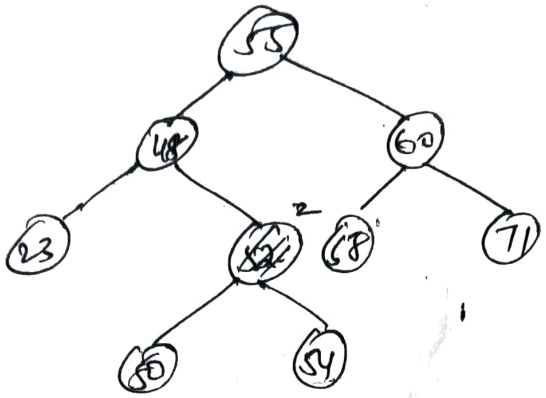
Before deletion of node 45



After deletion of node 45.

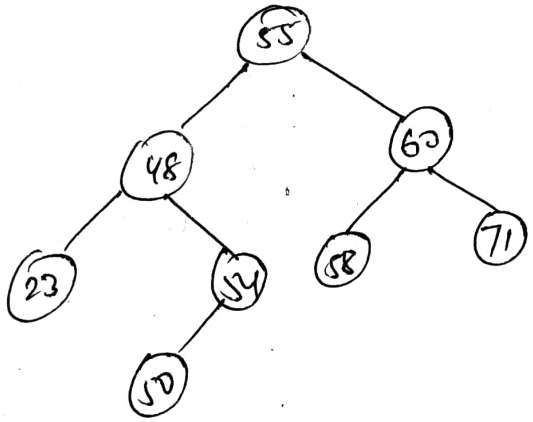
Case 3: If the node to be deleted z has two children. This case is a little more difficult since two subtree need to be reattached and both must maintain the same relation to the parent of the node being deleted and to each other. Here, find the inorder successor of the node to be deleted z. Place it in the node z, Then delete the inorder

Successor node Similar to case 1 or case 2.



Before deletion of node 52.

Find Successor of node 52, we get node 54. place it in the node 52 and then delete the successor node.



After deletion of node 52.

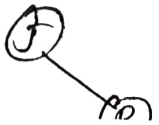
Q Suppose the following list of letters is inserted in order into an empty BST find the final tree.

J R D T G E A M H F Q U B.

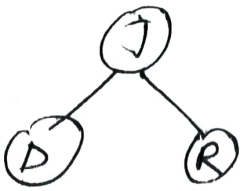
Ans Insert the nodes one after the other maintaining BST property

1. Insert node J in empty BST

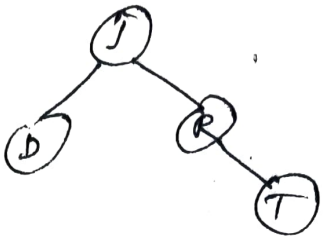
2. Insert node R



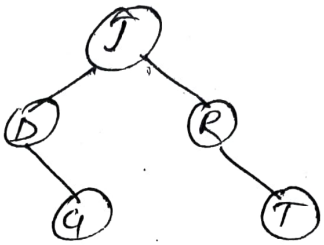
3. Insert D



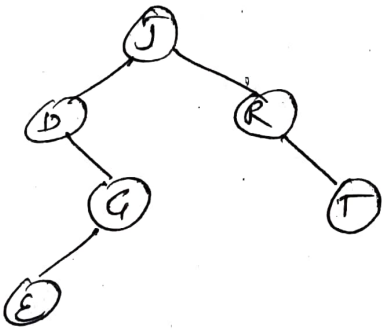
4. Insert node T



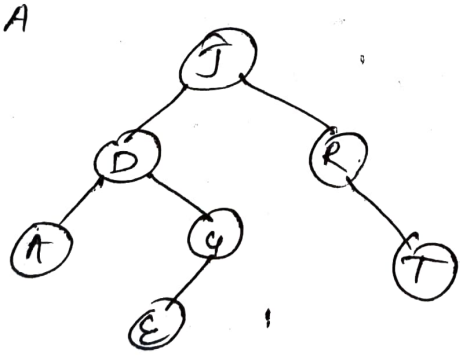
5. Insert node G



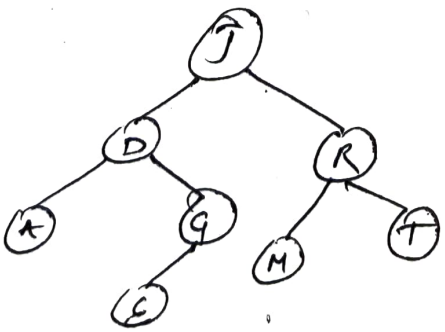
6. Insert node 'E



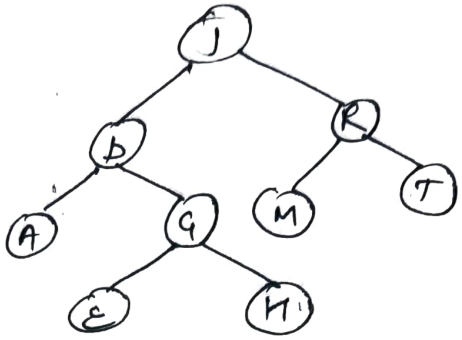
7. Insert node A



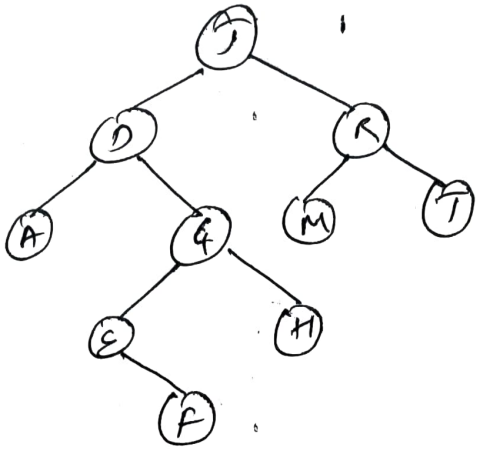
8. Insert node M



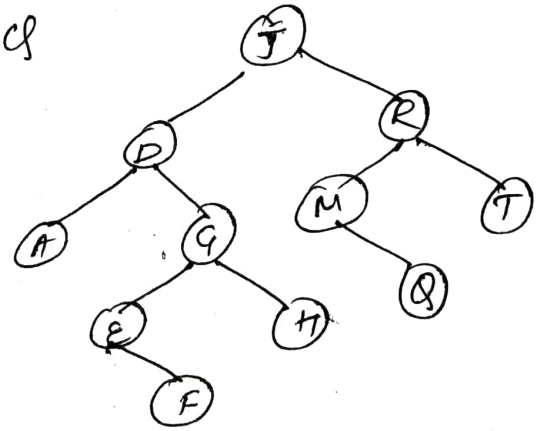
9. Insert node H



10. Insert node F



11) Insert node Q



12) Insert node U