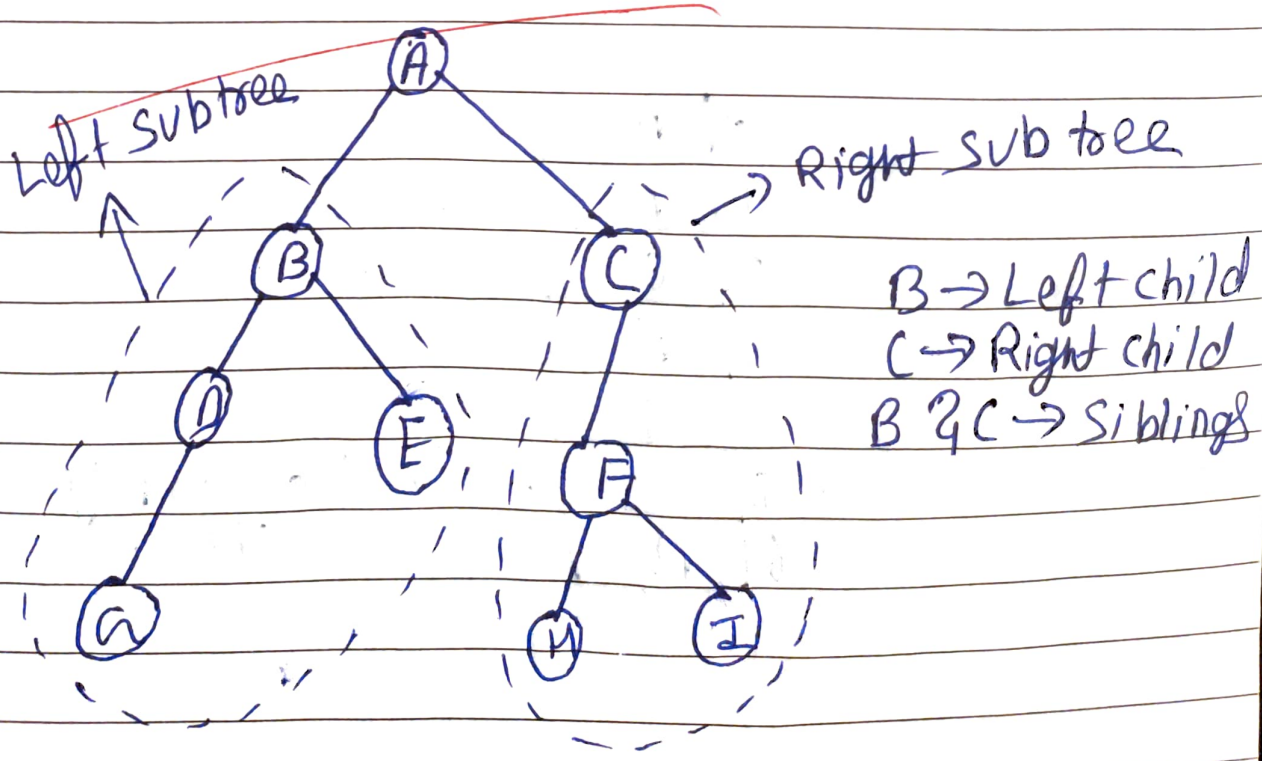
POORNIMA

Assignment = 04

Q1 What is binary Tree ? Describe representation of binary tree using arrays and linked list

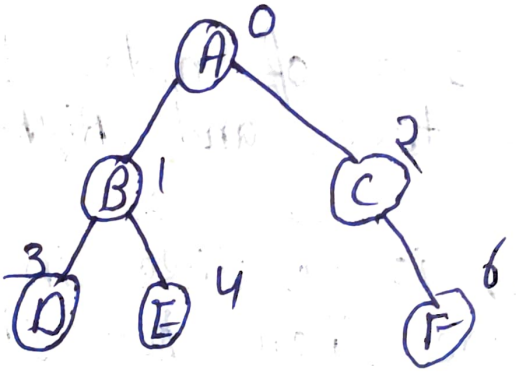
- A tree is a binary tree if each node of it can have at most two branches
- A binary tree 'T' is either empty or consists of special node called root node
- Tree has two sets of nodes L_T and R_T called the left sub tree and Right sub tree.
- A binary tree, is a tree in which no node can have more than two children (0-1-2).

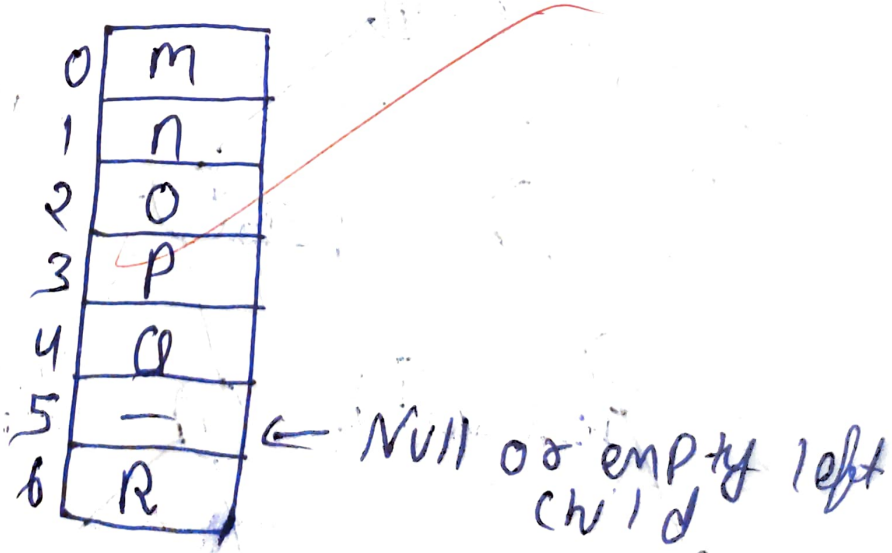


Representation of Binary tree :-

1) Sequential or Array Representation :-

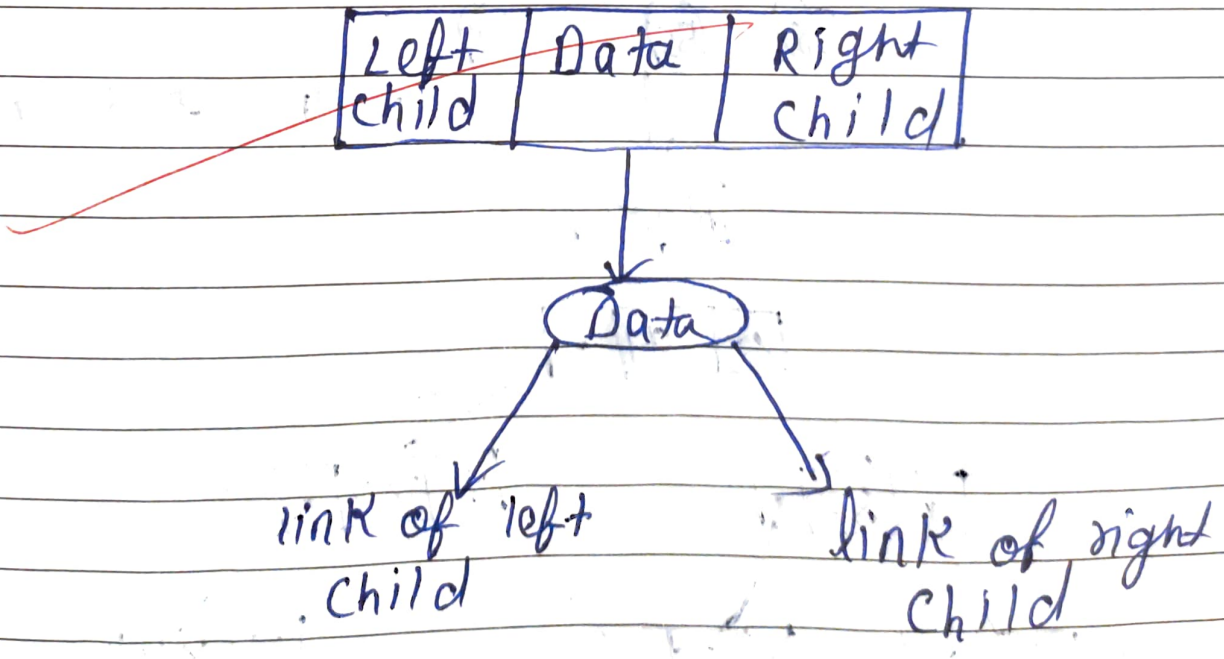
- useful when tree is complete or nearly complete
- used 1-D array
- In array the root node is always numbered by 0 and then it's left child and right child is stored at 1 and 2 respectively





2] Linked List Representation

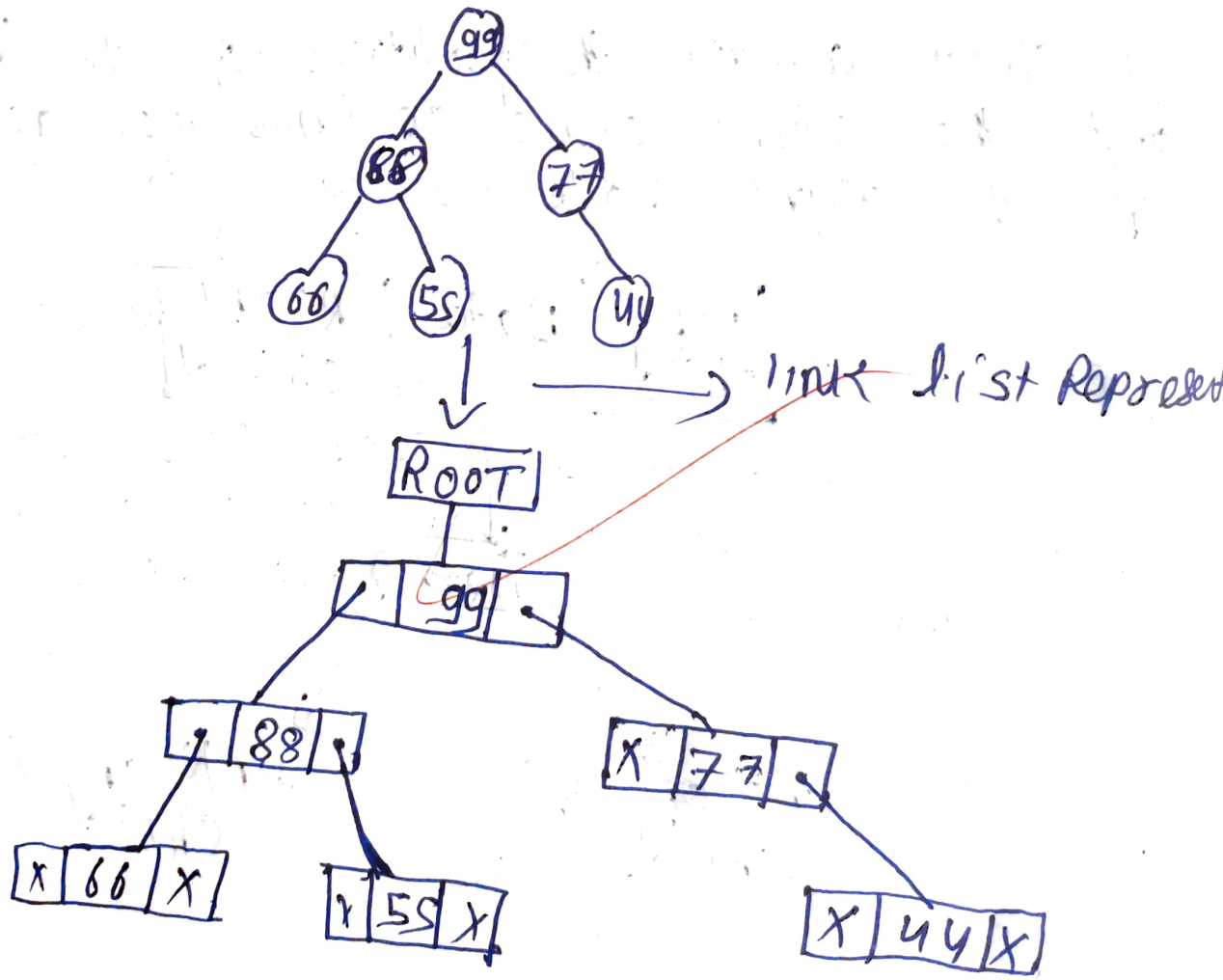
- Each binary tree node is represented as an object whose data type is Binary tree node.
- Basic component in binary tree is node. Each node has three fields, one for the link of left child, second for the data and third for the link of right child.
- Left child contains the address of its left node and the right child contains the address of its right Node.



Representation :-

```
struct Node
{
  struct node * Lchild;
  int Data;
  struct node * Rchild;
};
typed of struct Node Bi Tree;
```

eg



POORNIMA

Q2 Describe Operations and its Algorithm on Binary Tree ?

Ans Operation on Binary tree

① Traversal of a binary tree
② Insertion of nodes
③ Deletion of nodes
④ Searching for the nodes

① Traversal of a binary tree

- Traversal is a process to visit all the nodes of a tree and may print their values too.
- All nodes are connected via edges (link) we always start from the root (head) node.
- In tree creation we take three parameters node, left child, and Right child.
- There are three ways which we use to traverse tree.

① Pre-order Traversal
② In-order Traversal
③ Post-order Traversal.

1) Pre order Traversal :- (Root - left - Right)

(a) Process the root
(b) Traverse the left subtree in pre order
(c) Traverse the Right subtree in pre order

Algo

1. Begin
2. IF tree Not equal to NULL then
3. write : Data [Tree]
4. call pre order (Left (Tree))
5. call pre order (Right (Tree))
   [End of it]
6. Return.

2) In-order Traversal :- (Left - Root - Right)

(a) Process the left Subtree in inorder
(b) Process the Root
(c) Traverse the Right subtree in order

Algo

1. Begin
2. IF Tree Not equal to NULL then

3. Call Inorder (Left[Tree])
4. Call Inorder (Right[Tree])
   [End of IF]
6. Return.

3) Post-order Traversal

1. Traverse the left subtree in post order
2. Traverse the Right subtree in post order
3. Process Root (N)

(2) Insertion of a node in Binary search Tree

- Insert fun^n is used to add a new element in a binary search tree at appropriate location
- Insert fun^n is to be designed in such a way that, it must node violate the property of BST at each value.

(1) Allocate the memory for tree
(2) Set the data part to the value & set the left and Right pointer of tree, point to NULL
(3) IF the item to be inserted, will be the first element of the tree, then left and Right of this node will point to NULL

(5) If this is false, then Perform this operation recursing with the Right sub-tree of root.

Algo :-

1. Begin
2. IF Root = NULL then [Tree is empty]
   (a) Create a Root node using malloc Fun^n
   (b) Set Root → Data = Num.
   (c) Set Root → Left = NULL
   (d) Set Root → Right = NULL
3. Else if Num < Root → Data then

   Root → left = Insert (Root → left, num)
4. Else

   Root → Right = Insert (Root → Right - Num)

   [End of IF - else]
   [End of IF - else]
5. Exit.

POORNIMA

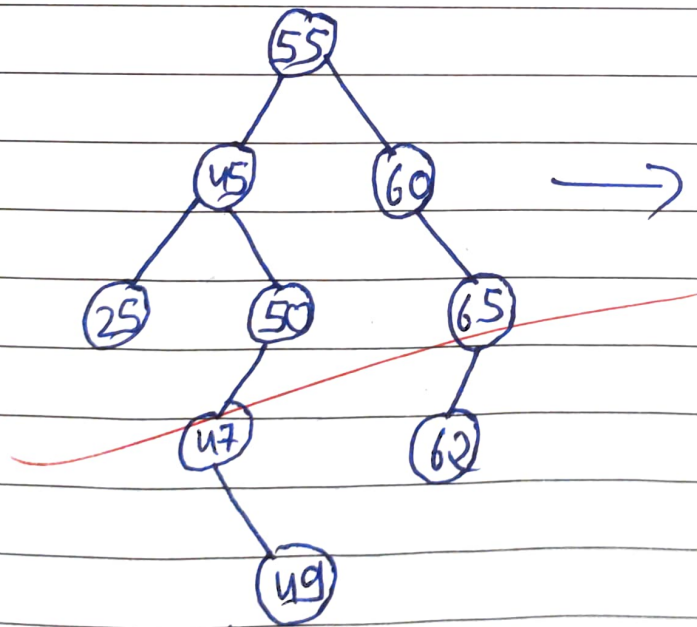
3] Deletion of a node in Binary Tree :-

There are four cases

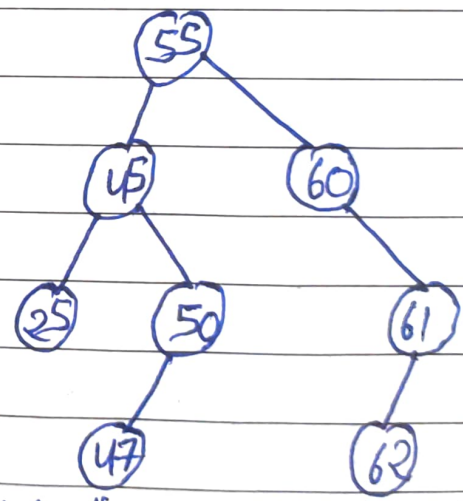
Case 1 IF the node X to be deleted has no child

- In this case, node X is deleted from the tree by simply replacing the location of node in the parent node by the NULL pointer.

eg :- X = 49



Before deletion



Node 49 is deleted

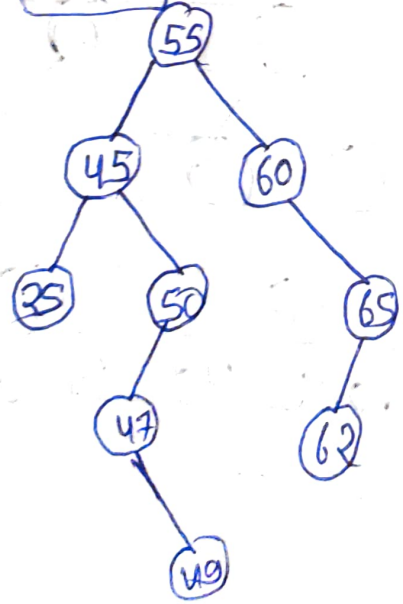
Algo

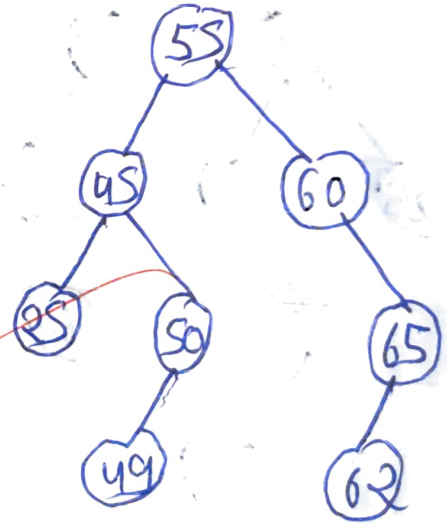
1. Begin

2. IF X → left = NULL AND X → Right = NULL then
3. IF PARENT → RIGHT = 'X' then
   Set PARENT → RIGHT = NULL
   Else
   Set PARENT → LEFT = NULL
   [End of IF-ELSE]
4. Delete or Free node x
   [End of IF]
5. Exit.

Case 2 :- IF the node X to be deleted has on
right child

eg :- x = 47





Node is deleted
47

POORNIMA

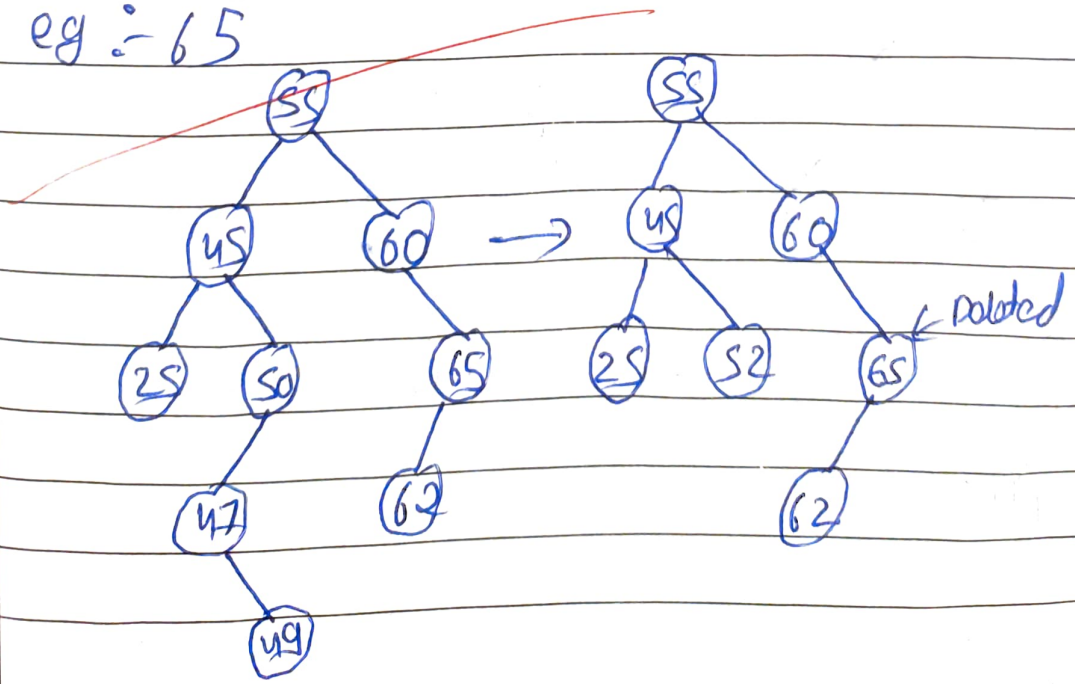
Algo

① Begin
② IF X → left = NULL and X → Right ≠ Null
③ IF PARENT → Left = X then
Set PARANT → left = X → RIGHT
Else
Set PARENT → RIGHT = X → RIGHT
[End of IF else]

④ Delete or Free node X
[End of IF]
⑤ Exit

Case3 IF the node X to be deleted has only left child

eg :- 65



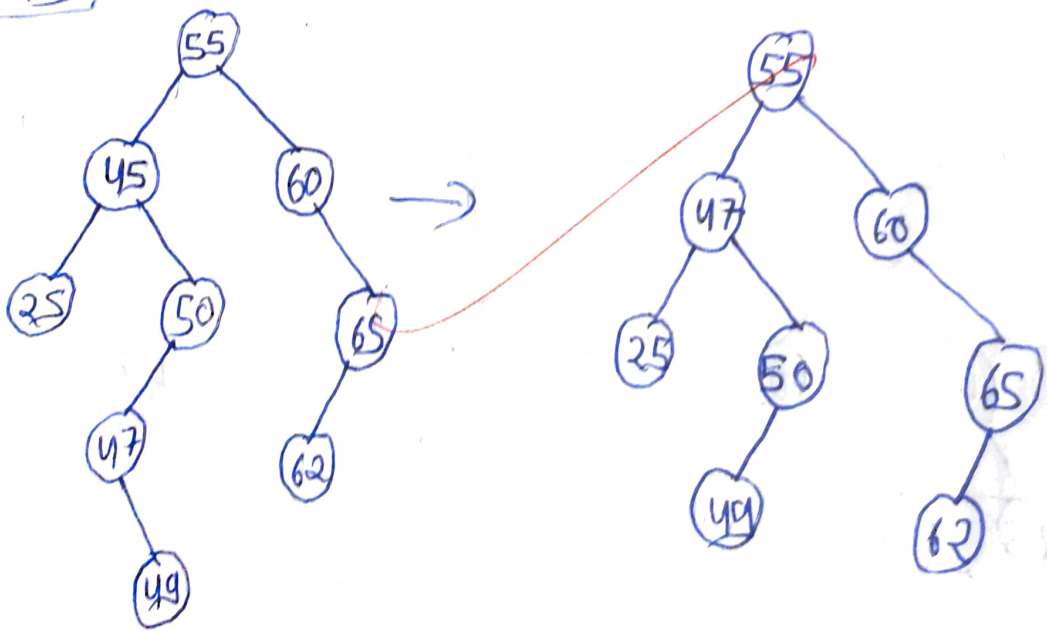
Algo

1. Begin
2. IF X → left ≠ Null and X → Right = NULL the
3. IF PARENT → left = x then
   set PARENT → left = X → left
   else
   set PARENT → Right = X → left

   [End of IF else]
4. Deleted or Free node X
5. Exit

Case 4 :- IF the node X to be deleted has two childrgen

X = 45



POORNIMA

Algo :-

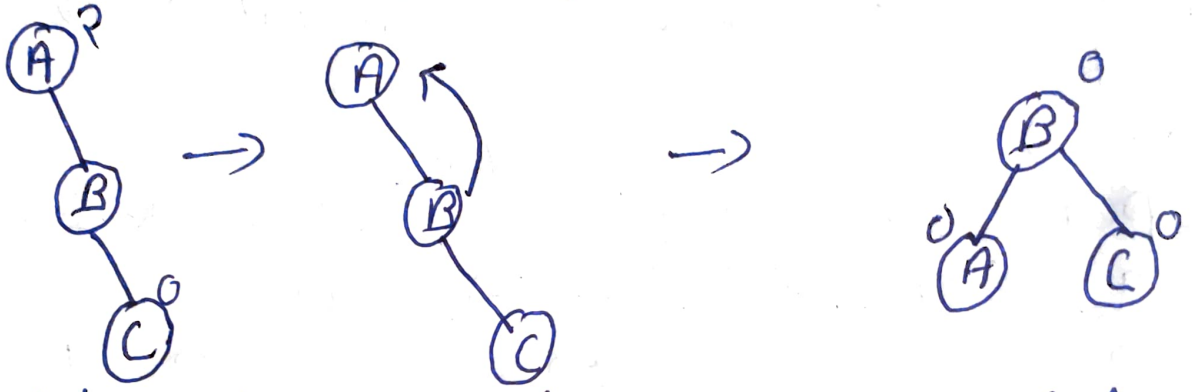
1. Begin
2. IF X → Left ≠ NULL & X → Right ≠ Null then
3. Set Parent = X
4. Set Xsucc = X → Right
5. Repeat while Xsucc → Left ≠ NULL
   (a) Set PARENT = X succe
   (b) set X → Info = X succ → IFO
6. Set X → INIFO = X Succ → IFO
7. Set X = X Succ.

Q3 What is AVL Tree its Applications ? Create an AVL tree or Height balanced tree? Inserted Data :- F, V, E, W, D, X, Y, B, Z.

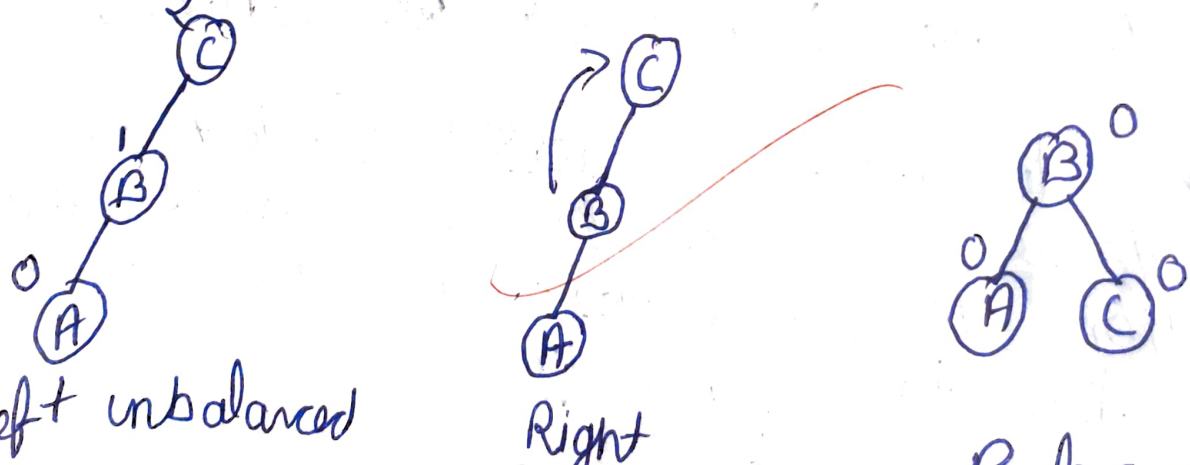
Ans It the difference in height of left and right sub-trees is More than 1. There are four rotation techniques that we use to balance the AVL trees namely

- Left rotation
- Right rotation
- Left - Right rotation
- Right - left rotation

1. Left Rotation : Here we can see that when a becomes unbalanced when a node is inserted the right subtree of the right subtree, then perform a single left rotation.



2. Right Rotation : AVL tree may also become unbalanced, if a node is inserted in left subtree of left subtree. The tree then needs a right rotation



3. Left-Right Rotation : Double rotations are slightly complex version of already explained version of rotations. To understand them better, we should take note of each action performed while rotation. To put it simply LR rotation is when we perform a left rotation followed by a Right rotation.

4. Right-left Rotation : This one is similar as of the above but the only difference is that this has a combination of right rotation followed by left rotation

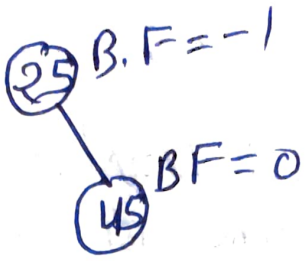
Create Height balanced tree in AVl tree

AVL Tree nodes 25, 45, 50, 55, 60

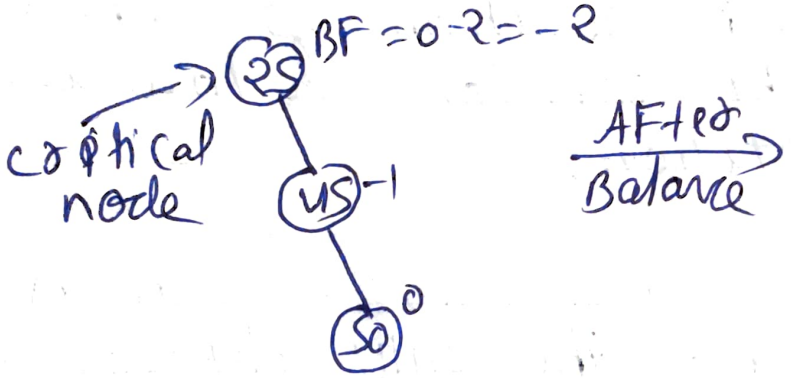
1. start from first node 25

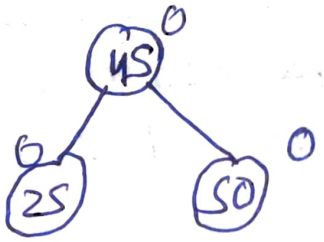
(25) B.F = 0

2. Insert 45 Node. 45 > 25, So it will come Right side of 25.

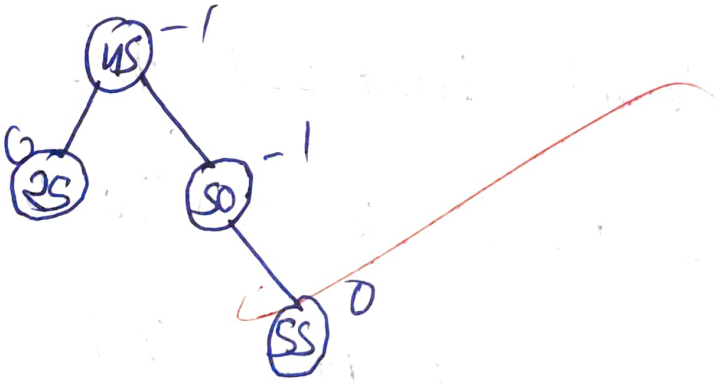


3. Insert 50, since 50 > 45, it will come rig side of node 45

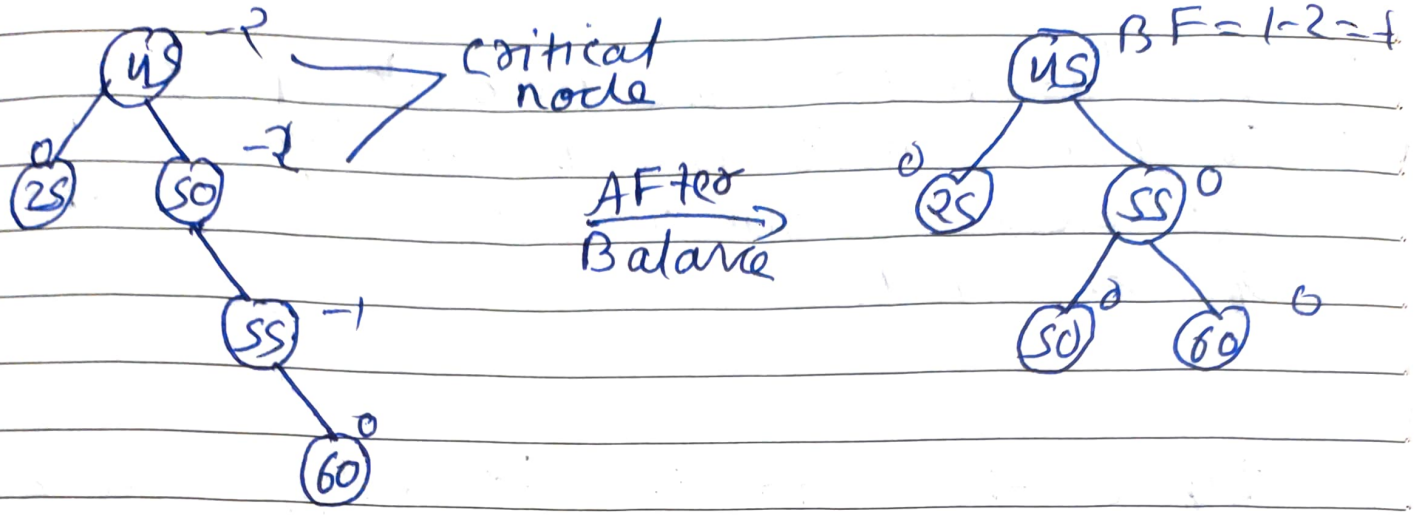




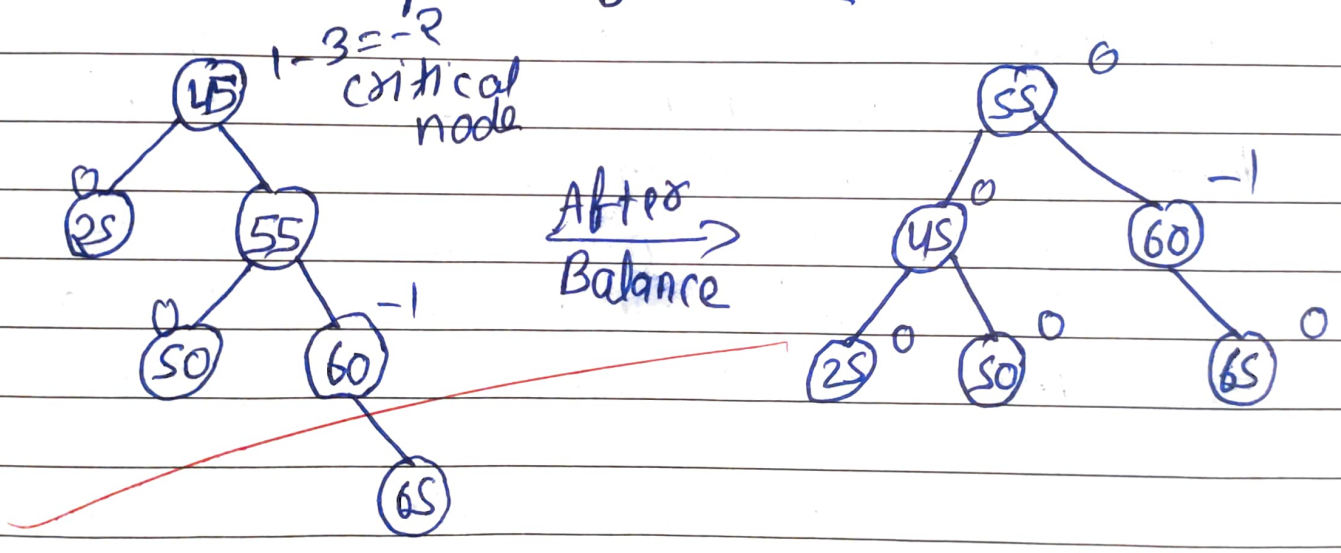
4. Insert 55 in the right side of 50 (sinc 55 > 50) Now find B.F of every node



5. Insert 60 in the right side 55 (since 60 > 55)



6. Insert 65 in right side of 60 and now Find B.F of every node



Insertion in AVL Tree :-

a Insertion of a node in an AVL tree is similar to that in a binary search tree.

b Insertion may disturb the balance factor of an AVL tree and therefore.

c] For this purpose, we need to perform data
d] Every new node will be at leaf node
e] At every insertion we will calculate the B.F
of each node.

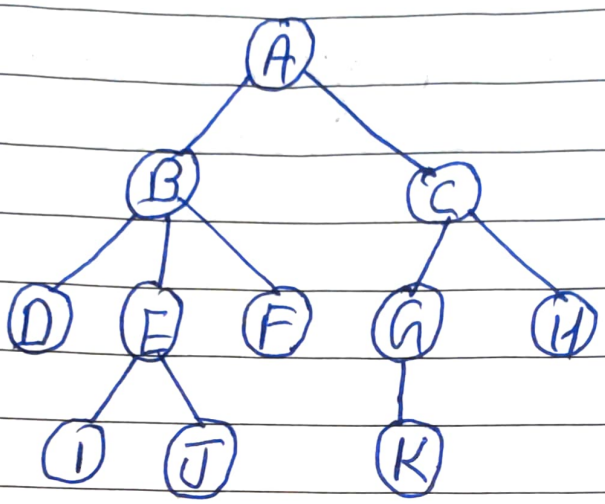
Q4 Define tree and its terminology with examp

Ans Tree - Terminology

In linear data structure data is organized in sequential order and in non-linear data stru data is organized in random order. A tree a very popular non-linear data structure used in a wide range of application. A tree data structure can be defined

- Tree is a non-linear data structure whi organizes data in hierachical structure a this is a recursive definition
- Tree data structure is a collection of (Node) which is organized in hierarchical structure recursively

Example

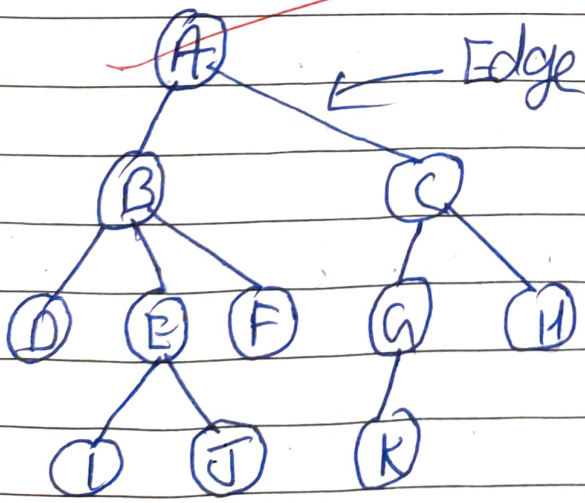


TRee with 11 nodes and 10 edges

1) Root

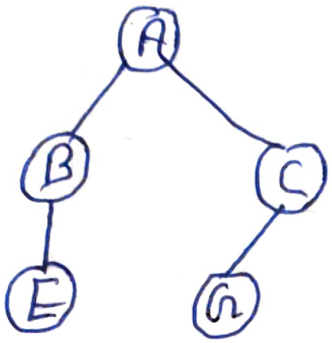
Here 'A' is the 'root' node

2) Edge



- In any tree 'Edge' is a connecting link between two nodes

3. Parent



Here A, B, C, E are Parent

- In any tree the node which has child/children is called 'Parent'

4. Leaf

Here D, I, J, F, K & H are Leaf

- In any tree the node which does not have children is called 'Leaf'.

5. Path

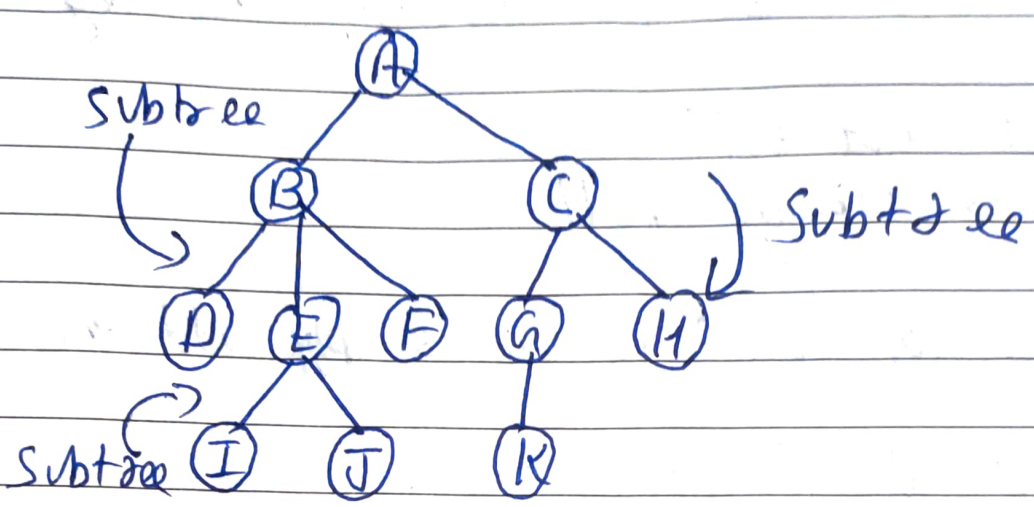
- In any tree 'Path' is a sequence of nodes and edges between two nodes

Here 'Path' between A & J is
A - B - E - J

Here, 'Path' between C & K is
C - G - K

POORNIMA

6. Sub Tree



Q5 What is B Tree? Describe insertion and Deletion with example?

Ans B Tree :-

→ A, B-tree is a balance M-way tree
→ B-Tree is a self balancing search tree
→ B-Tree of order M has following properties

(a) It should be perfectly balanced i.e BF = 0 and all leaf nodes must be at same level.
(b) Each node (except root node) contain min. M/2 children and max. M children
(c) Each node has one key value (fewer value) than children. no node can contain more than M-1 values.
(d) Keys are arranged in defined order with in a node

Insertion in B-Tree :-

We have two case for inserting the key

1. Node is not full
2. Node is already full

1) Node is not full :- We can simply add the key in

2) Node is already Full :-

- We need to split the node into two nodes and medi key will go to parent of that node.
- IF parent is also full then same thing will be repeated until non full parent node

eg :- Let take a list of keys and create a B-Tre of order 5.

10, 70, 60, 20, 110, 40, 80, 130, 100, 50, 190, 90, 180, 24 120, 140, 160

Sol :-

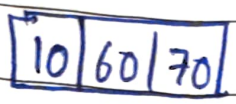
Step 1 :- Insert 10

10

Step 2 :- Insert 70

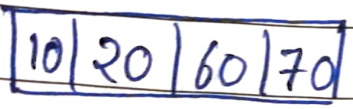
10	70

Step 3 :- Insert 60

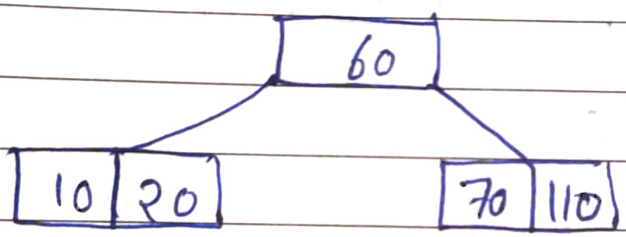


* After 60 the Keys in node

Step 4 :- Insert 20

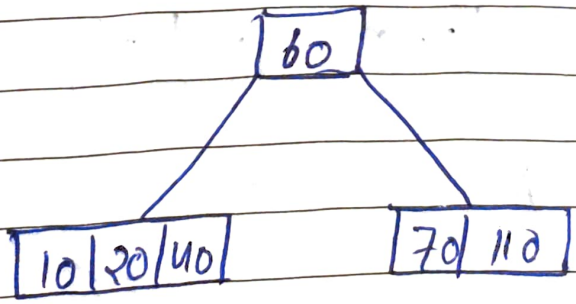


Step 5 :- Insert 110

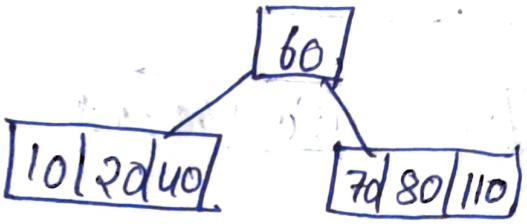


Note:- Here node was already full (5 order), so after insertion of 110, it should be splitted into two nodes. 60 is the median key 20 it will go in the parent node

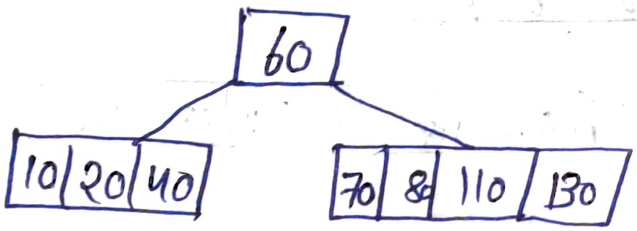
Step 6 :- Insert 40.



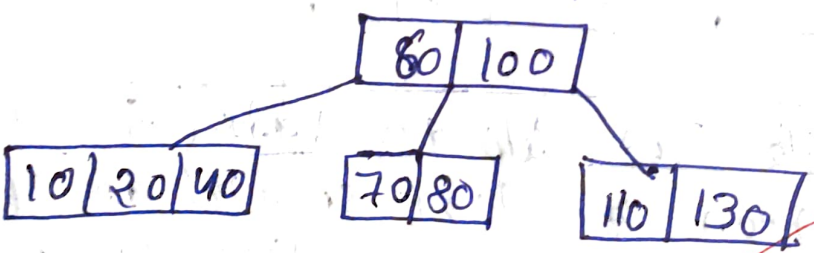
Step 7 :- Insert 80



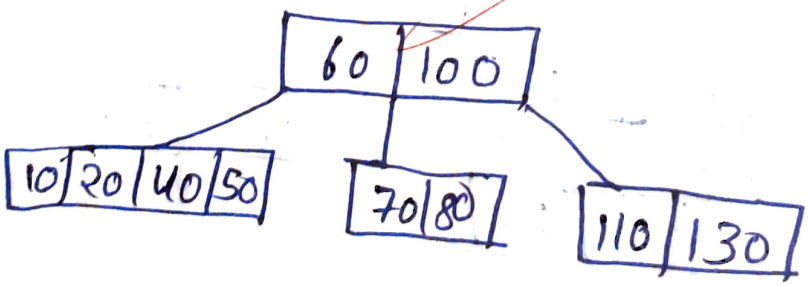
Step 8 :- Insert 130



Step 9 :- Insert 100

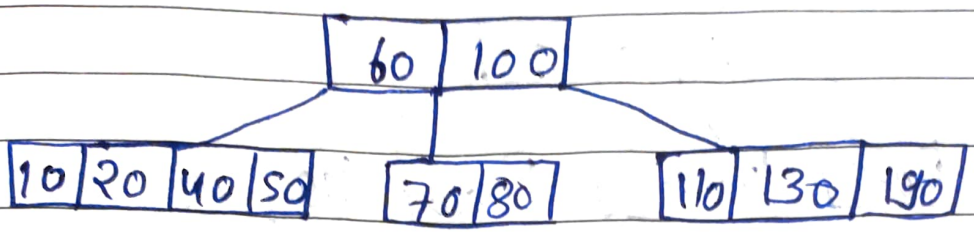


step 10 :- Insert 50

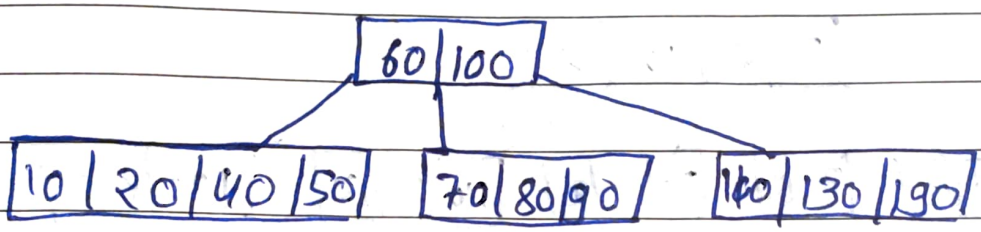


step 11 :- Insert 190

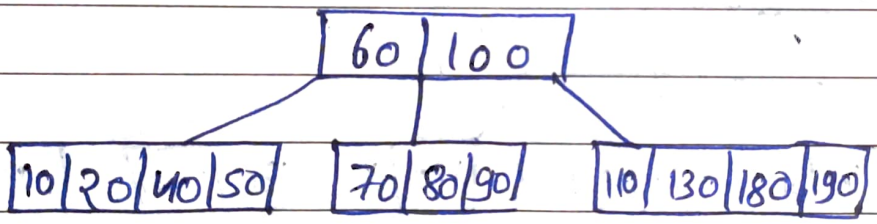
POORNIMA



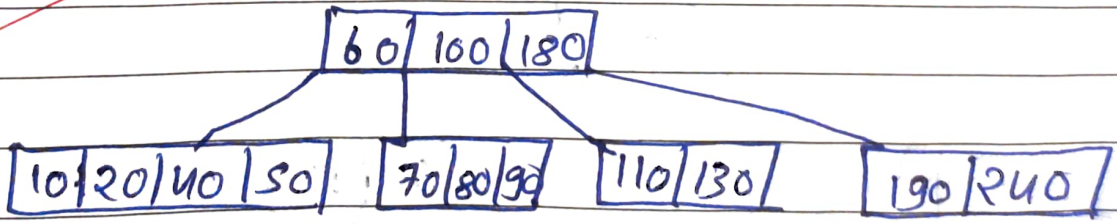
step 12 :- Insert 90



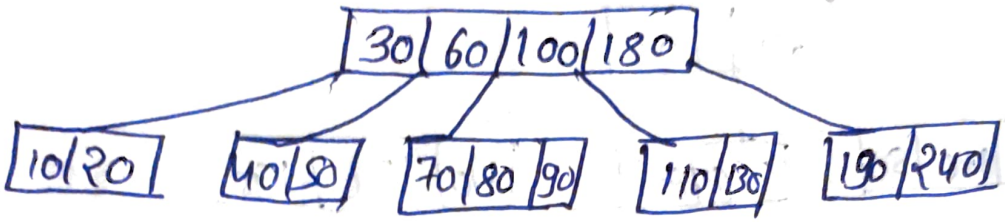
step 13 :- Insert 180



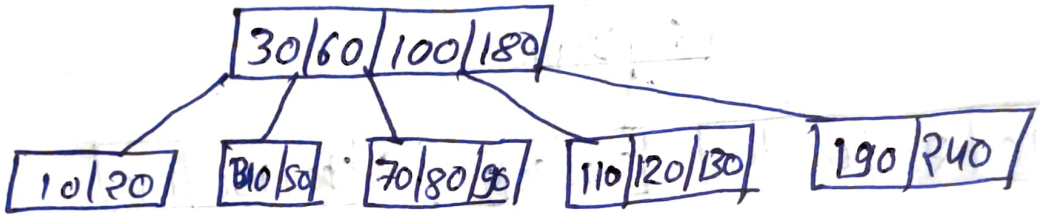
Step 14 :- 240



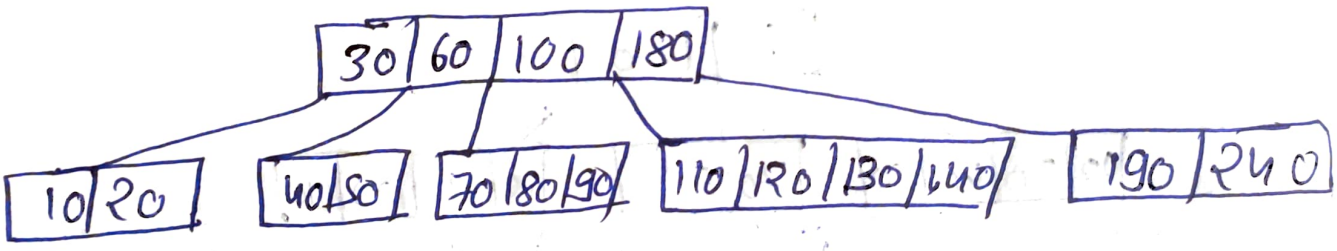
Step 15 :- 30



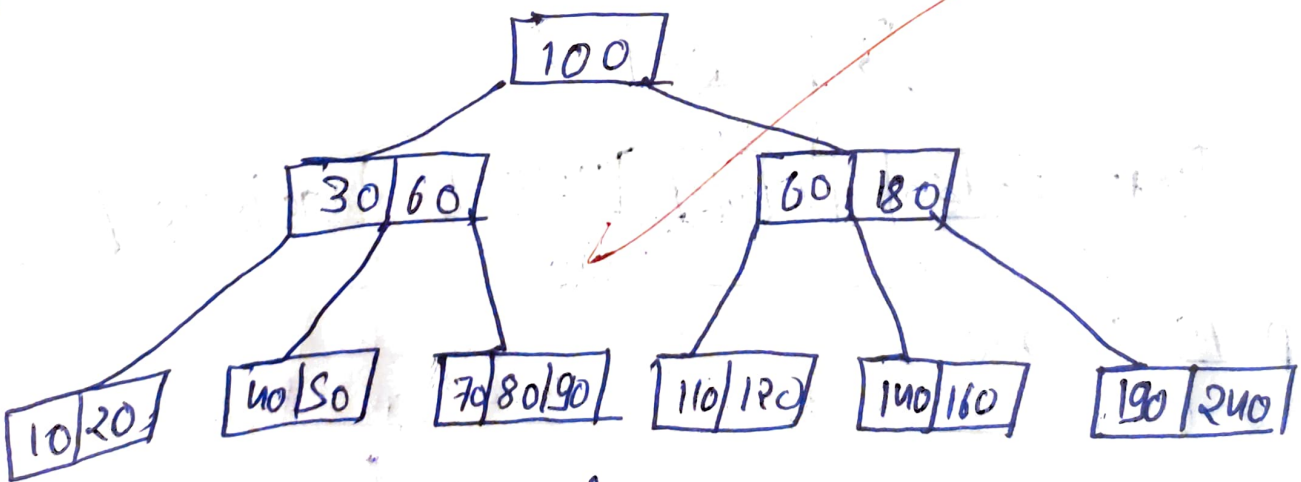
Step 16 :- 120



Step 17 :- 140



step 18 :- 160



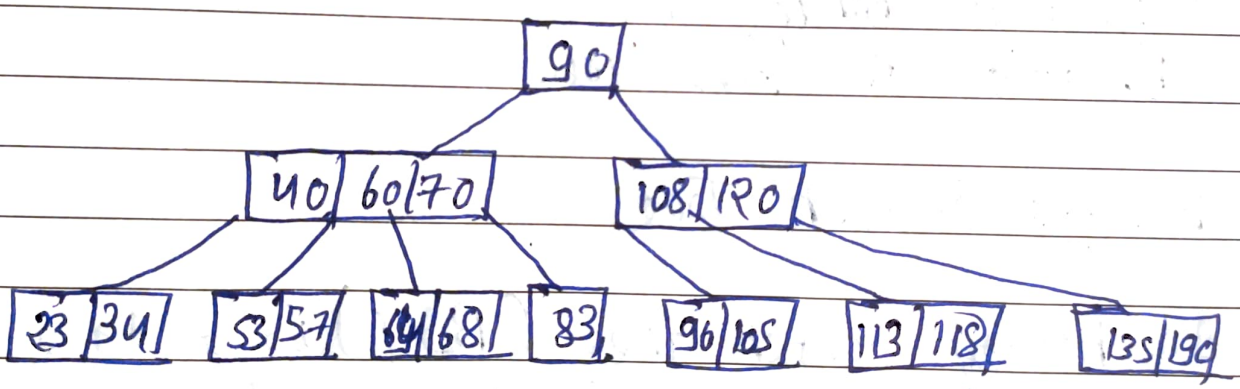
final B-tree

Deletion in B. Tree

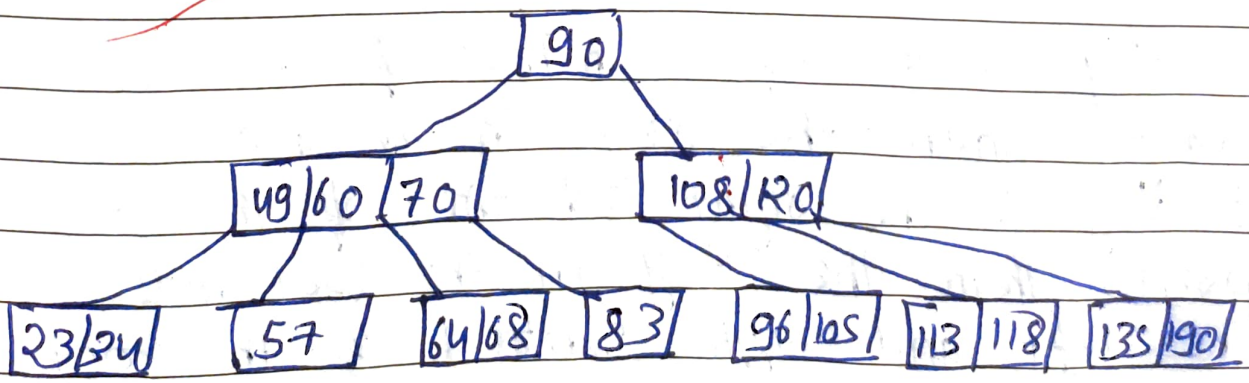
- Deletion of key also requires first traversal in B-tree. After reading on particular node two case may occur :-

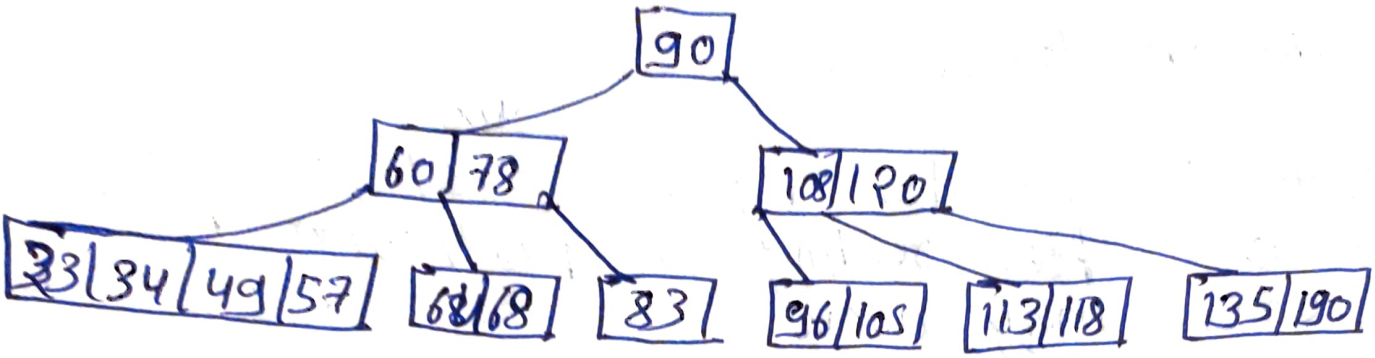
(a) Node is leaf node
(b) Node is non leaf node

B. Tree of order 5



Delete 53





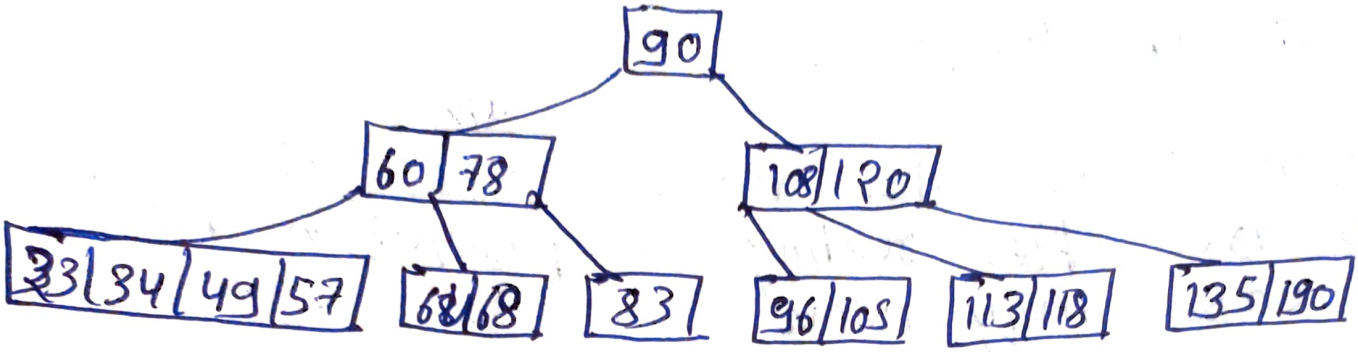
Q6 Short Notes with example

a] Binary Search Tree
b] B' tree
c] Threaded Binary Tree
d] Deletion in AVL Tree

a] Binary Search Tree

A binary Search tree (BST) is a tree that either empty or in which every node contain a Key (value) and satisfies the following condition

- All Key in the left subtree of the root are smaller than the Key in the root node.
- All Key in the right sub-tree of the root are greater than the Key in the root node
- The left and right subtrees of the root are again binary search tree.



Q6 Short Notes with example

a] Binary Search Tree
b] B tree
c] Threaded Binary Tree
d] Deletion in AVL Tree

a] Binary Search Tree

A binary Search tree (BST) is a tree that either empty or in which every node contain a Key (value) and satisfies the following condition

- All Key in the left subtree of the root are smaller than the Key in the root node.
- All Key in the right sub-tree of the root are greater than the Key in the root node
- The left and right subtrees of the root are again binary search tree.

POORNIMA

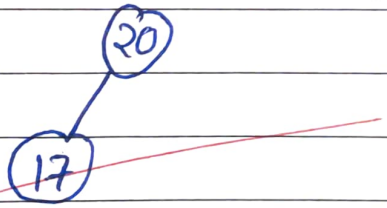
- If a binary search tree is traversed in order, then the contents of each node are printed in ascending order.
- In binary search tree we can search for an element easily an average running time

$$F(n) = O(\log_2 n)$$

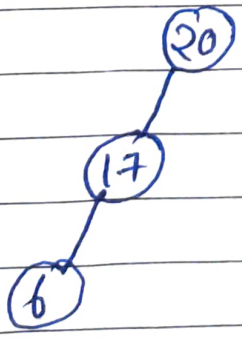
eg: ~~50, 30, 60, 25, 40, 35~~ 20, 17, 6, 8, 25, 10, 23, 5, 27, 9

step 1 :- (20)

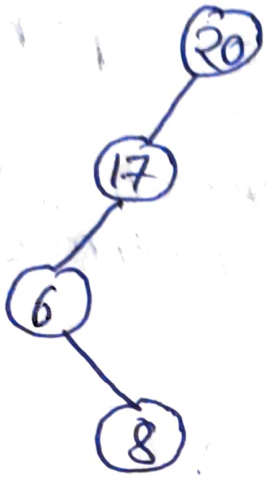
step 2 :- Data = 17



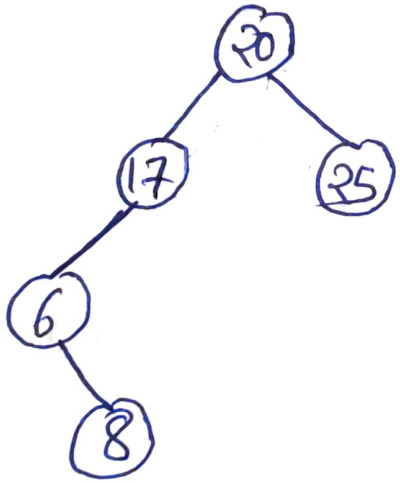
step 3 :- Data = 6



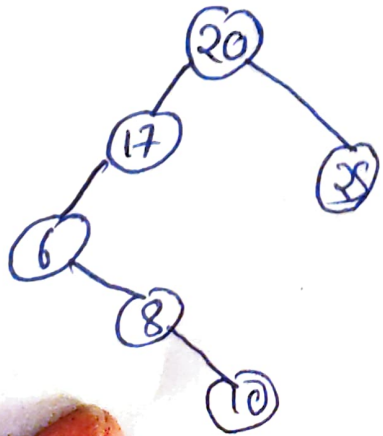
Step 4 :- Data = 8



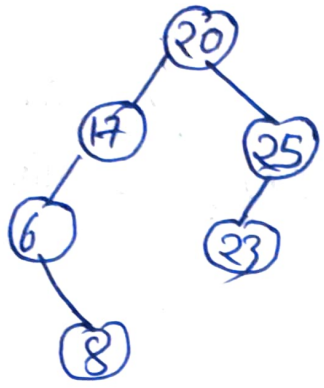
Step 5 :- Data = 25



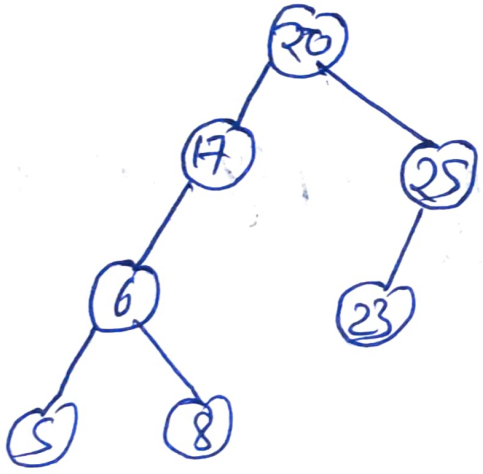
Step 6 :- Data := 10



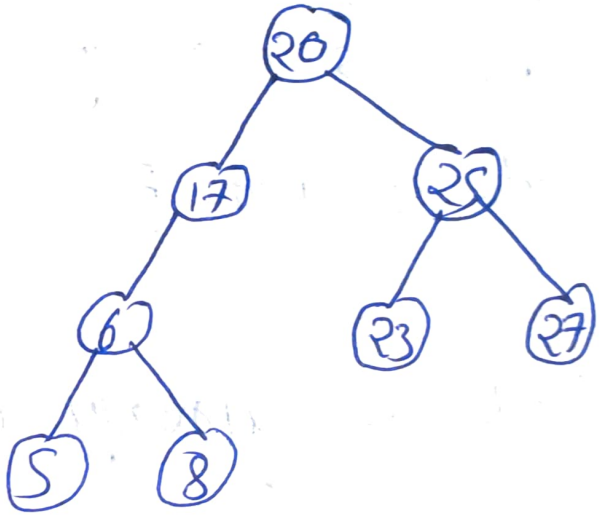
Step 7 :- Data = 23



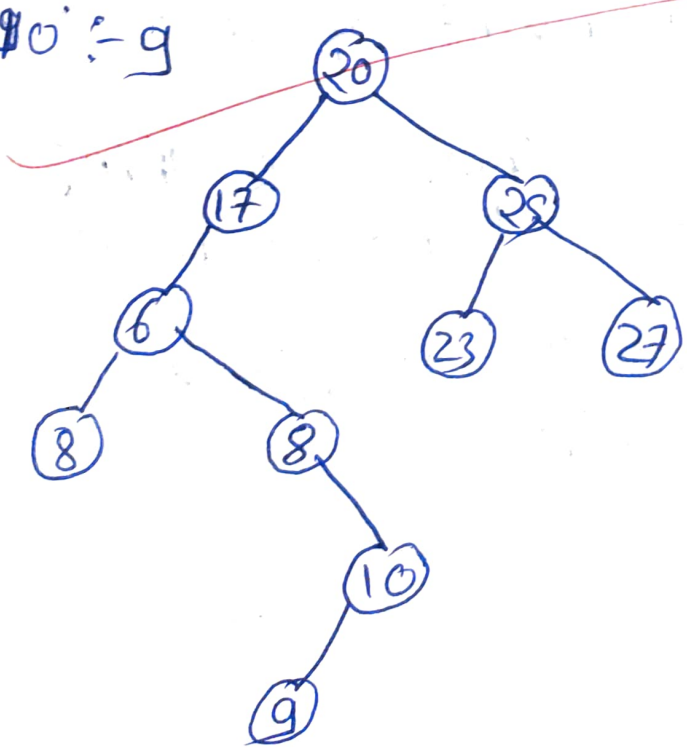
Step 8 :- Data :- 5



Step 9 :- 27



Step 10 :- 9



b] B Tree

→ B-tree is a balance M-way tree (5-way
→ B-Tree is a self balancing search tree

Insertion is B-Tree

* We have two case for inserting the Key

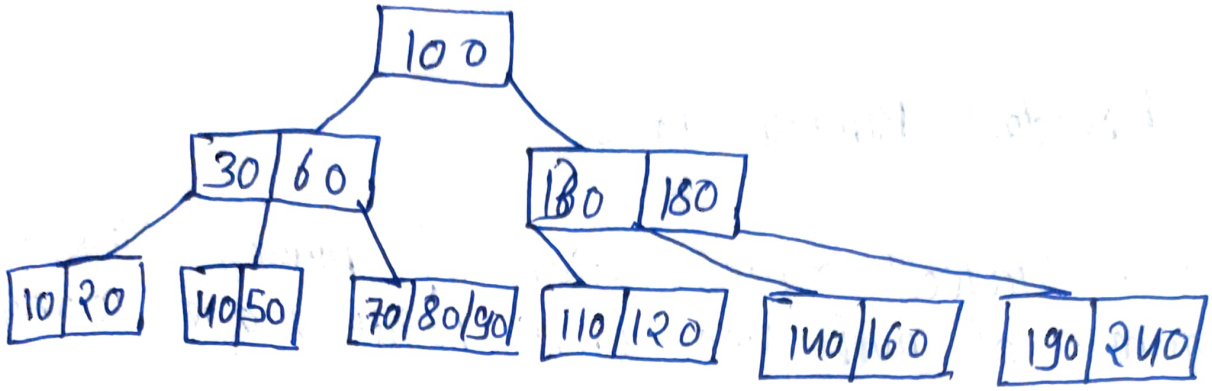
1] Node is not Full
2] Node is already full

① Node is not Full :- We can simply add the Ke in node

② Node is already Full :-

- We need to split the node into two nodes and medium Key will go to Parent of that node
- IF Parent is also full then Same thing will be repeated until non full Parent node

eg :- 10, 70, 60, 20, 110, 40, 80, 130, 100, 50, 90, 190, 9
180, 240, 30, 120, 140, 160

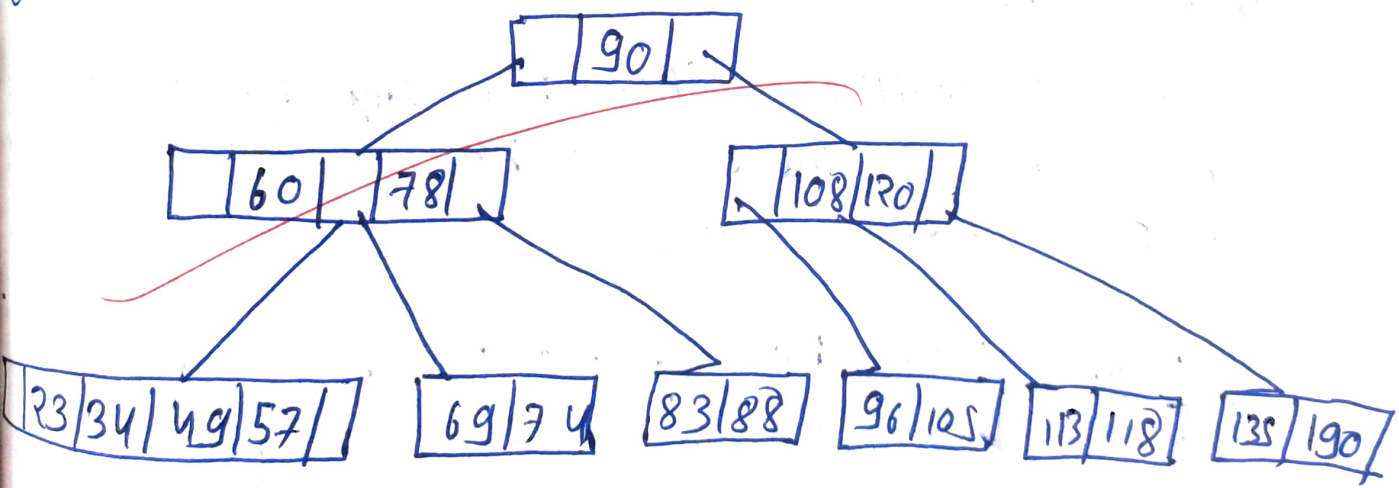


Deletion in B Tree

Deletion of key also requires first traversal in B-tree. After reading on particular node two case may occur

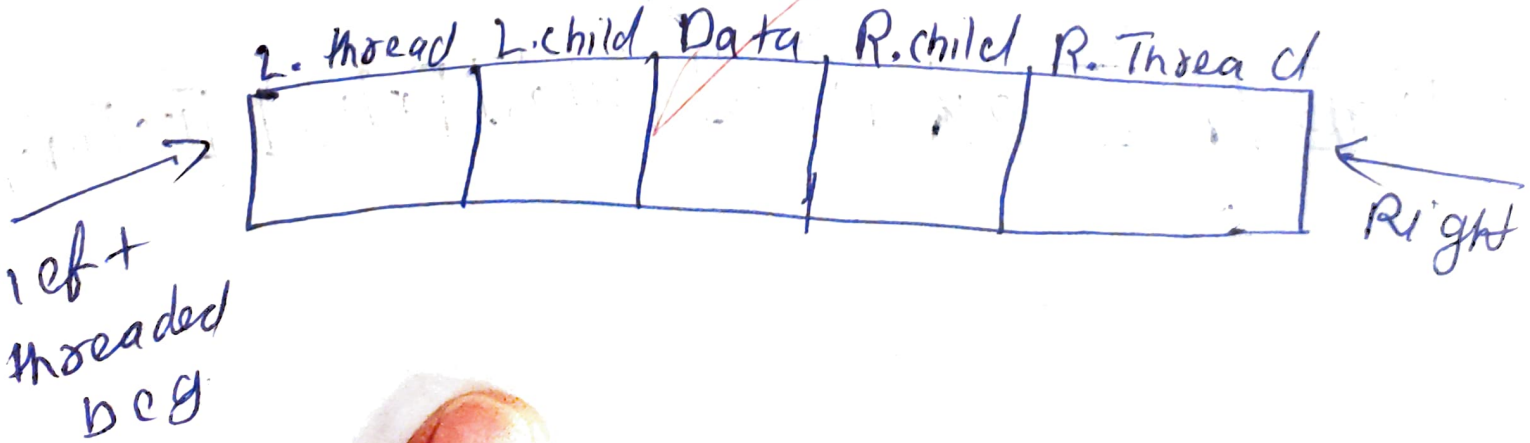
a) Node is leaf node
b) Node is non leaf node

eg



5] Threaded Binary Tree

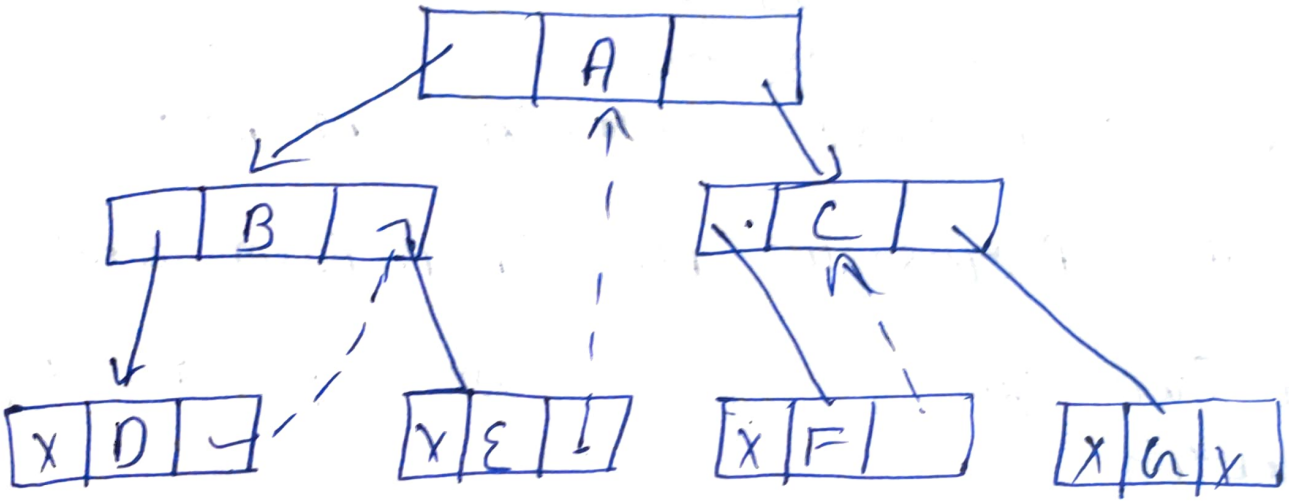
- In a linked representation of binary tree, the number of null links (null pointers) are actually more than non-null pointers
- In above binary tree, there are 8 null pointers and actual 6 pointers.
- In all there are 14 pointers
- The objective here to make effective use of these null pointers.
- And binary tree with such pointers are called Threaded tree.
- We can generalize it that for any binary tree with n nodes there will be (n+1) null pointers and 2n total pointers



Types of Threaded Binary Tree :-

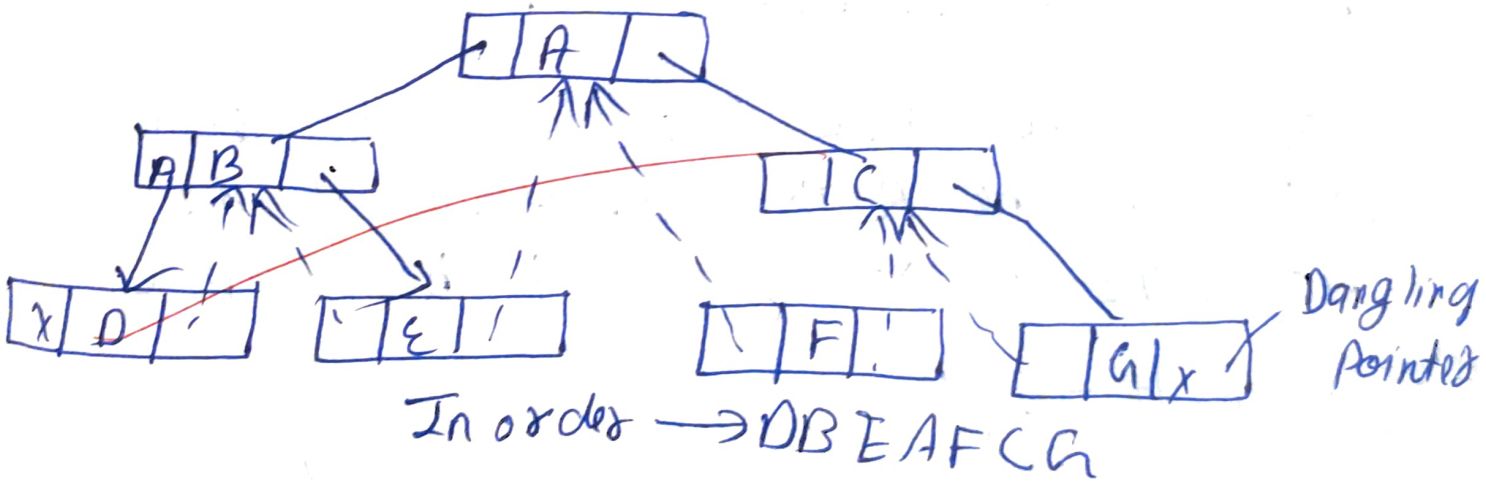
1. one way Threading
2. Two way Threading

1) one way Threading



Right Pointer = next node of in order

2) Two way Threading.



In order → DBEAFCG

Right Pointer = next node in - order traversal.

Left Pointer = previous node of in order traversal

4] Deletion in AVL Tree

1] Deleting a node from an AVL tree is similar to that in a binary search tree

2] Deletion may disturb the balance Factor of an AVL tree and therefore, the tree needs to be rebalanced in order to maintain the balance-ness.

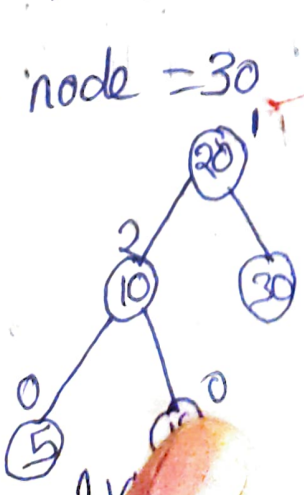
3] For this purpose, we need to perform rotation

4] The Two type of rotations are left and Right Rotation

5] IF the node which is to be deleted is present in the left Subtree of the Critical node then L rotation need to be applied

6] Else if the node which is to be deleted is present in the Right Sub-tree of the Critical node, the R rotation will be applied

ex Delete node = 30



Delete node 30