Unit-5

Trees

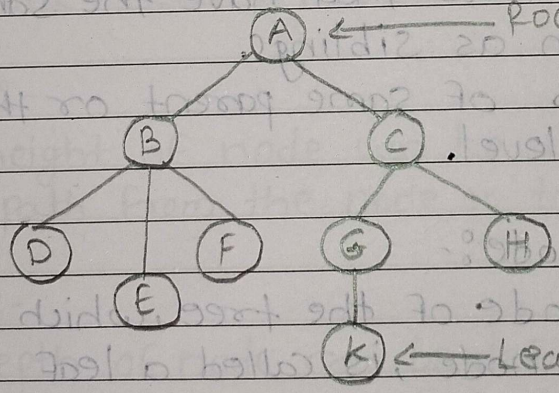
Que- What is Tree and Explain Basic Terminology?

⇒

Tree:- — Recursive definition

- A tree is also one of the data structures that represent hierarchical data.
- A tree data structure is defined as a collection of object or entities known as nodes that are linked together to represent or simulate hierarchy.
- A tree data structure is a non-linear data structure because it does not store in a sequential manner.
- It is a hierarchical structure as elements in a Tree are arranged in multiple levels.

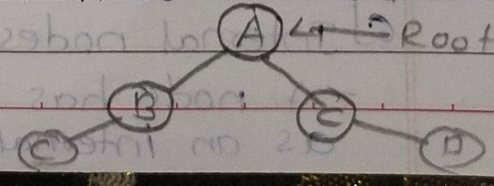
Diagram:-



- Basic Terminology :-

1) Root:-

- The root node is the topmost node in the tree hierarchy.
- In other words, the root node is the one that doesn't have any Parent.



2] Child node :-
- If the node is a descendant of any node, then the node is known as a child node.
- Any node which has parent is called child node.

3] Parent :-
- If the node contains any sub-node, then that node is said to be the parent of that sub-node.
- A node which has children is called Parent node.

4] Sibling :-
- The nodes that have the same parent are known as siblings.
- Nodes of same parent or the tree of same level.

5] Leaf Node :-
- The node of the tree, which doesn't have any child node, is called a leaf node.
- A leaf node is the bottom-most node of the tree.
- There can be any number of leaf nodes present in a general tree.
- Leaf nodes can also be called external nodes.

6] Internal nodes :-
- A node has atleast one child node known as an internal node

- Except Leaf node all are Internal Node.

7] Edge :-
- If there are n nodes, then there would n-1 edges.
- Each arrow in the structure represents the link or path.
- Each node, except the root node, will have atleast one incoming link known as an edge.
- edge is connectivity between nodes.

8] Level :-
- In the hierarchy each node stored in different level.
- Root node - Level-1, children node - level 2 so on.

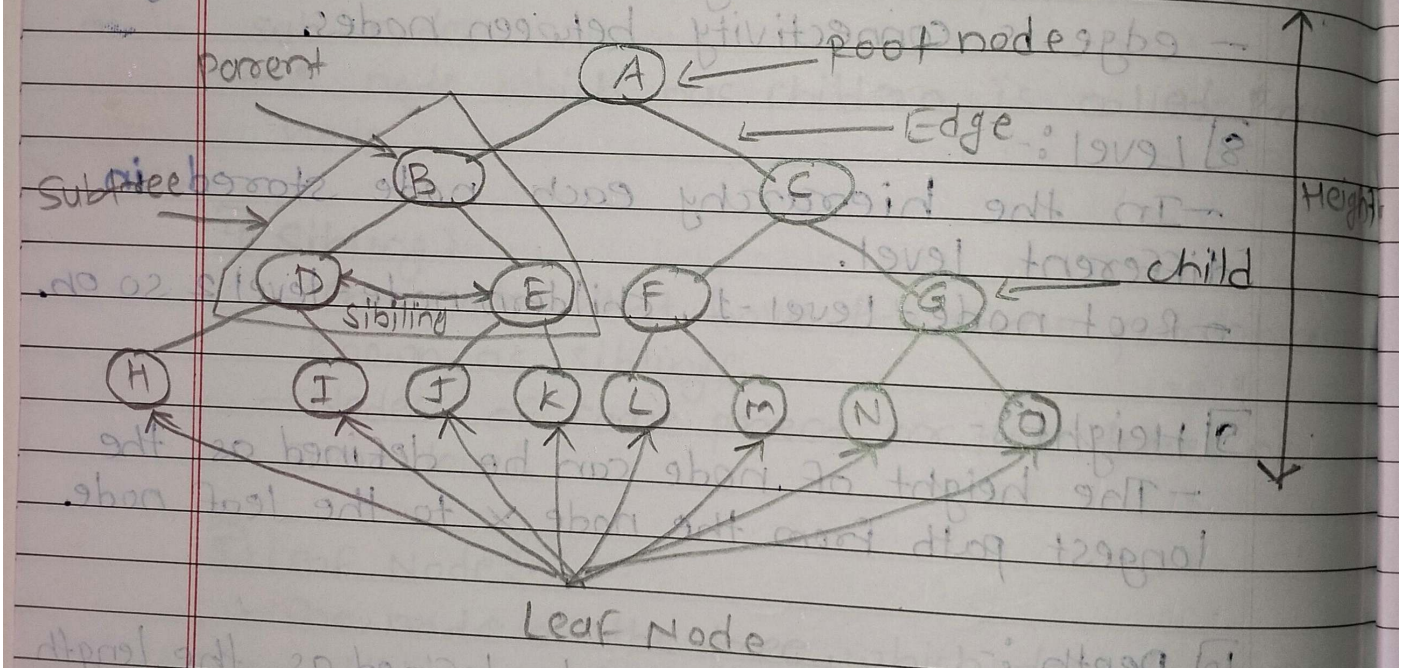
9] Height :-
- The height of node can be defined as the longest path from the node x to the leaf node.

10] Depth :-
- The depth of node can be defined as the length of the path from the root to the node.
- Total number of edges from root node to particular node is called Depth.

11] Degree :-
- The total number of children of a node is called Degree of node.

12 Path :-
- Path is defined by identify the Root from source to the destination by selected edges.

13 Subtree :-
- subtree can be defined as. Any group of node with parent or child node.



Que- Explain Types of Tree data structure?

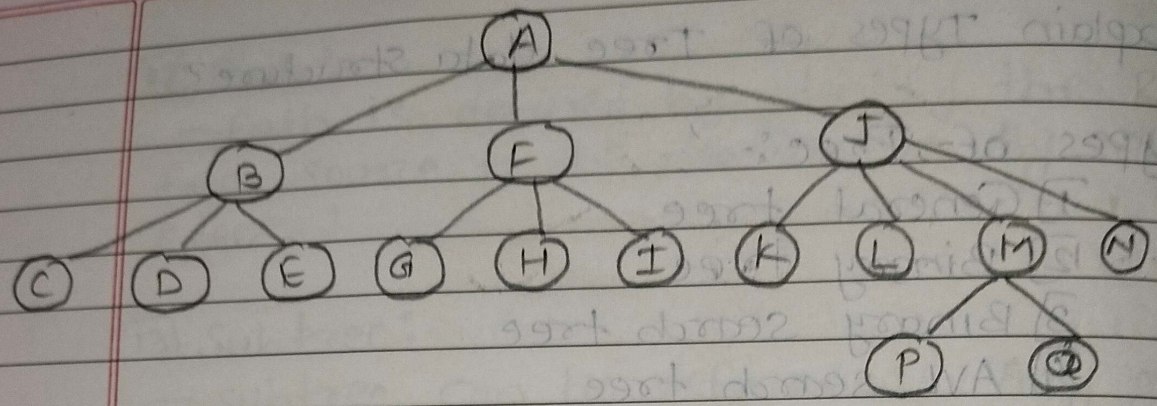
→

Types of Tree:-

1] General tree
2] Binary tree
3] Binary search tree
4] AVL search tree

1] General tree:-

- The general tree is one of the types of tree data structure.
- In the general tree, a node can have either 0 or maximum n number of nodes.
- There is no restriction imposed on the degree of the node (the number of nodes that a node can contain).
- The topmost node in a general tree is known as a root node.
- The children of the parent node are known as subtrees.
- There can be n number of subtrees in a general tree.
- In the general tree, the subtrees are unordered as the nodes in the subtree cannot be ordered.
- Every non-empty tree has a downward edge, and these edges are connected to the nodes known as child nodes.
- The root node is labeled with level 0.
- The nodes that have the same parent are known as siblings.



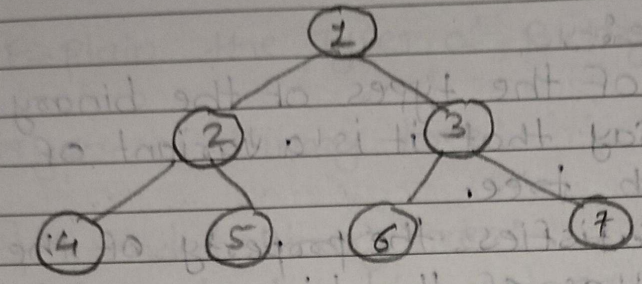
2] Binary tree:-

- Here, binary name itself suggests two numbers i.e. 0 and 1.
- In a binary tree, each node in a tree can have atmost two child nodes.
- Here, atmost means whether the node has 0 nodes, 1 node or 2 nodes.
- In a binary tree, each node can have a maximum of two children linked to it.
- The topmost node in a binary tree is called the root and the bottom-most nodes are called leaves.
- A binary tree can be visualized as a hierarchical structure with the root at the top and the leaves at the bottom.

• Representation of Binary tree:-

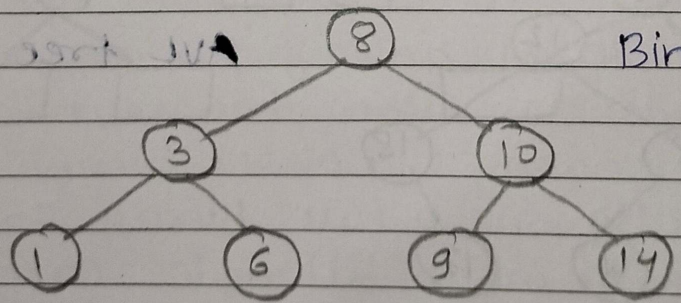
- Each node in the tree contains the following:
  - Data
  - Pointer to the left child
  - Pointer to the right child

Binary tree



3) Binary search tree :-

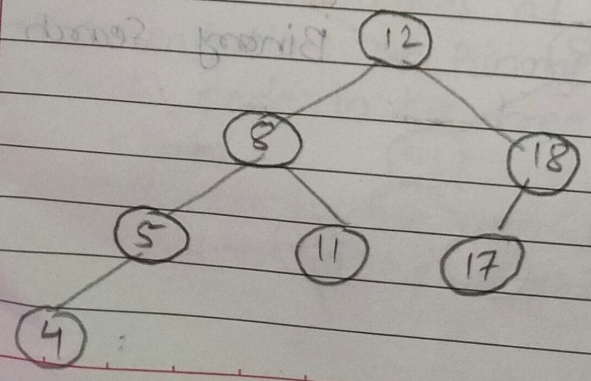
- Binary search tree is a non-linear data structure in which one node is connected to n numbers of nodes.
- It is a node-based data structure.
- A node can be represented in a binary search tree with three fields i.e. data part, left-child and right child.
- A node can be connected to the utmost two child nodes in a binary search tree, so the node contains two pointers (left child and right child pointer).
- Every node in the left subtree must contain a value less than the value of the root node, and the value of each node in the right subtree must be bigger than the value of the root node.



Binary Search tree

4] AVL tree :-

- It is one of the types of the binary tree, or we can say that it is a variant of the binary search tree.
- AVL tree satisfies the property of the binary tree as well as of the binary search tree.
- It is a self-balancing binary search tree that was invented by Adelson Velsky Landis.
- Here, self-balancing means that balancing the heights of left subtree and right subtree. This balancing is measured in terms of the balancing factor.
- We can consider a tree as an AVL tree if the tree obeys the binary search tree as well as a balancing factor.
- The balancing factor can be defined as the difference between the height of the left subtree and the height of the right subtree.
- The balancing factor's value must be either 0, -1, or 1; therefore, each node in the AVL tree should have the value of the balancing factor either as 0, -1 or 1.



AVL tree

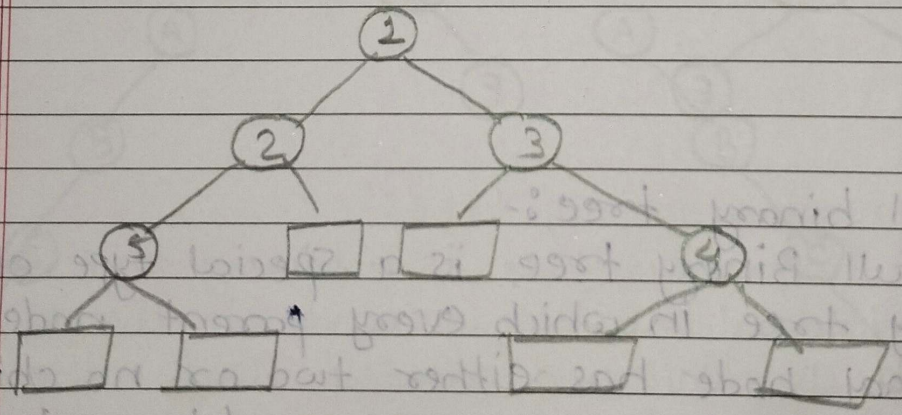
que- Explain the Types of Binary tree?

⇒

Types of Binary tree:-

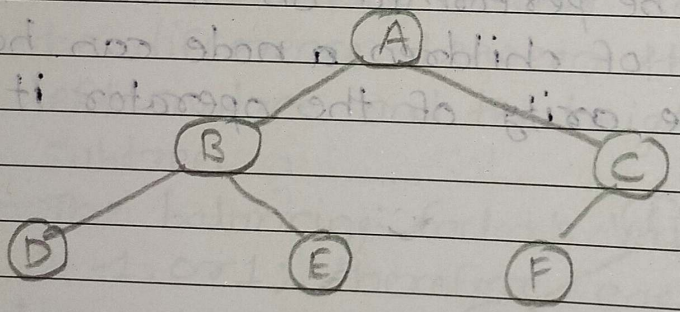
1) Extended Binary tree:-

- An extended binary tree is a type of binary tree that extends the structure of a regular binary tree to allow for the representation of expressions with an arbitrary number of operands.
- In an extended binary tree, each node can have an arbitrary number of children.
- The nodes in an extended binary tree represent either operators or operands.
- operand nodes contain data (such as variables or constants), while operator nodes represent operations to be performed on their children.
- The number of children a node can have depends on the arity of the operator it represent.



2] Complete Binary tree :-

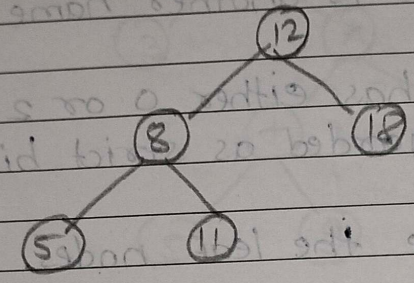
- A complete binary tree is a special type of binary tree where all the levels of the tree are filled completely except the lowest level nodes which are filled from as left as possible.
- A complete binary tree is said to be a proper binary tree where all leaves have the same depth.
- In a complete binary tree number of nodes at depth 'd' is 2^d.
- In a complete binary tree with n nodes height of the tree is $\log(n+1)$.
- All the levels except the last level are completely full.



3] Full binary tree :-

- A Full Binary tree is a special type of binary tree in which every parent node/ internal node has either two or no children.
- It is also known as a proper binary tree.
- we can also say a Full binary tree is a binary tree in which all nodes except leaf nodes have two children.
- A Binary Tree is a Full binary tree if

every node has 0 or 2 children.

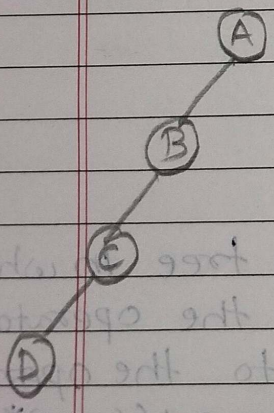


Full Binary tree

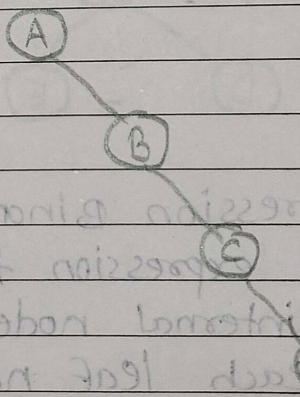
4] Skewed Binary tree :-

- A skewed binary tree is a pathological / degenerate tree in which the tree is either dominated by the left nodes or the right nodes.
- Thus, there are two types of skewed binary tree:
  1] left-skewed binary tree
  2] Right skewed binary tree.

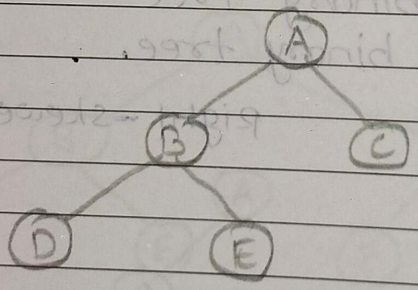
Left-skewed Binary tree



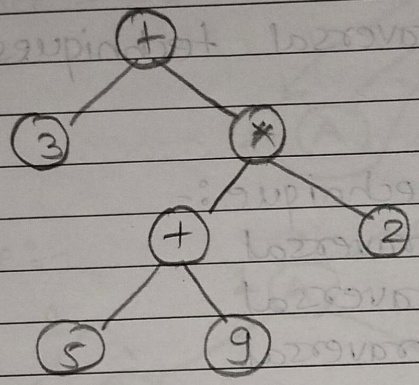
Right-skewed Binary tree



5] Strictly Binary tree:-
- complete binary tree is another name for the strict binary tree.
- only if each node has either 0 or 2 offspring can the tree be regarded as strict binary tree.
- where all nodes, save the leaf nodes, must have two children.
- The whole binary tree must have at least $2*h-1$ nodes.
- The whole binary tree has a minimum height of $\log_2(n+1)-1$.
- The maximum number of nodes is $2h+1-1$, which corresponds to the number of nodes in the binary tree.



6] Expression Binary tree:-
- The expression tree is a binary tree in which each internal node corresponds to the operator and each leaf node corresponds to the operand so for example expression tree for 3+((5+9)*2) would be.



- Expression trees are a special kind of binary tree used to evaluate certain expression.
- Two common types of expressions that a binary expression tree can represent are algebraic and boolean.
- It is also used to find out the associativity of each operator in the expression.
- It is also used to solve the postfix, Prefix and infix expression evaluation.

Que- Explain Tree traversal techniques with examples?

⇒

Tree traversal technique :-

1] Preorder traversal
2] Inorder traversal
3] Postorder traversal

1] Preorder traversal :-

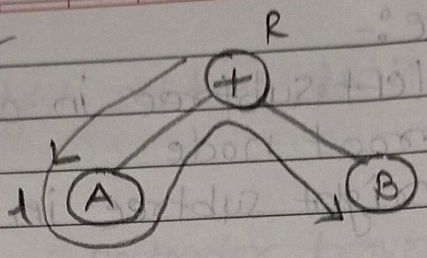
- This technique follows the 'root left right' Policy.
- It means that, first root node is visited after that the left subtree is traversed recursively and finally, right subtree is recursively traversed.
- As the root node is traversed before (or Pre) the left and right subtree, it is called preorder traversal.

- The applications of preorder traversal include-
  - It is used to create a copy of the tree
  - It can also be used to get the prefix expression of an expression tree.

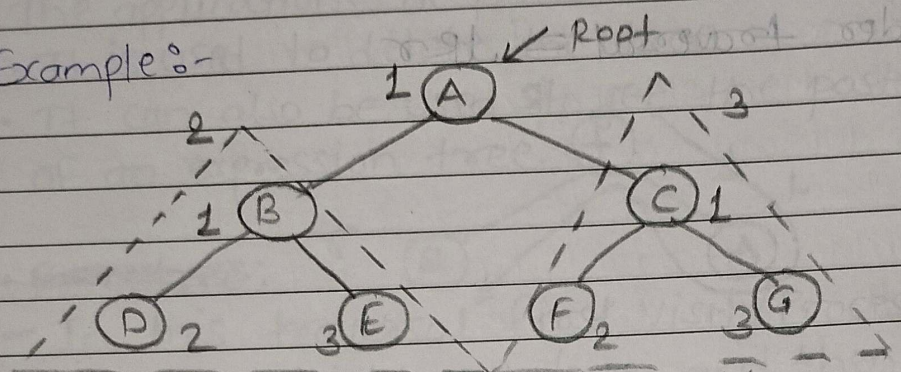
- In this traversal method first process root element, then left sub tree and then right sub tree.

- Procedure :-
  1) visit root node
  2) visit left sub tree in Preorder
  3) visit right subtree in preorder.

Preorder = R l r



Example:-



preorder traversal:-

A → B → D - E → C → F → G

2) Inorder traversal:-

- This technique follows the 'left root right' policy.
- It means that first left subtree is visited after that root node is traversed, and finally, the right subtree is traversed.
- As the root node is traversed between the left and right subtree, it is named Inorder traversal.

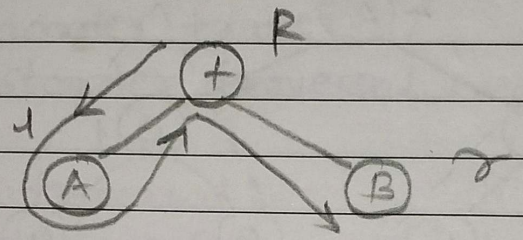
- The applications of Inorder traversal include-
- It is used to get the BST nodes in increasing order.
- It can also be used to get the prefix expression of an expression tree.

- In this traversal method, first process left element then root element and then the right element.

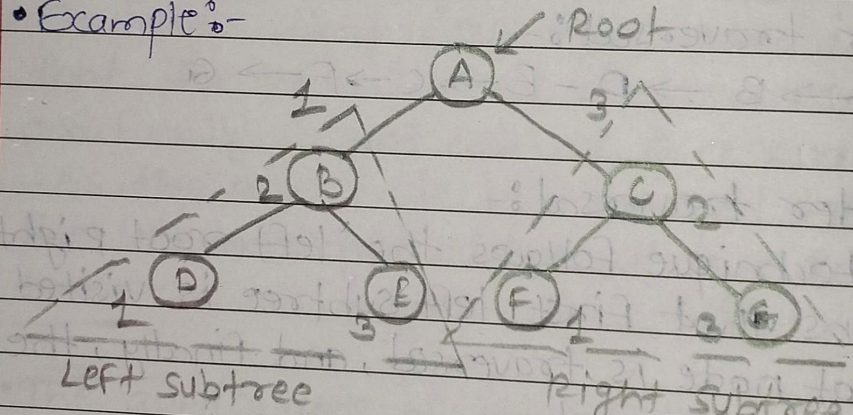
• Procedure :-

1] visit left subtree in order
2] visit root node
3] visit right subtree in order.

Inorder traversal = LRr



• Example :-



inorder traversal :-

D → B → E → A → F → C → G

3] Post order traversal :-

-This technique follows the 'Left-Right root' policy
-It means that the first left subtree of the root node is traversed, after that recursively traverses the right subtree, and finally the root node is traversed.
-As the root node is traversed after (or Post)

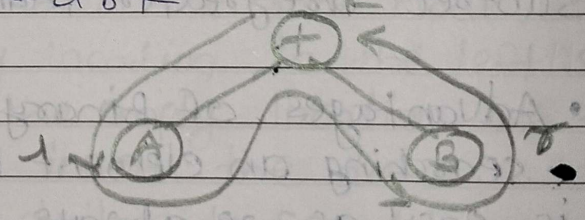
the left and right subtree, it is called Postorder traversal.

- The applications of postorder traversal include -
  - It is used to delete the tree
  - It can also be used to get the postfix expression of an expression tree.

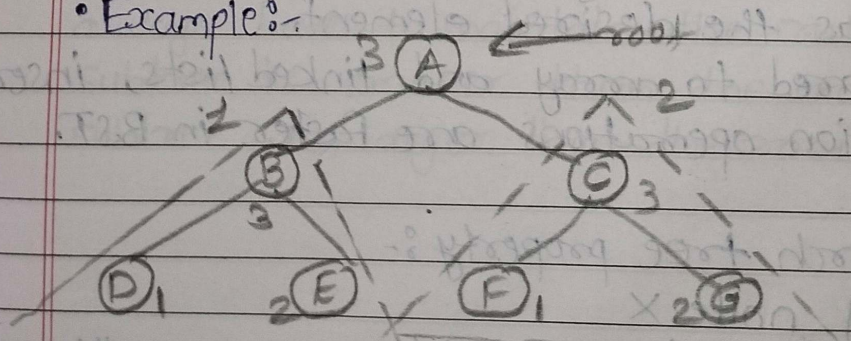
- ~~Procedure~~:
  - In this traversal first visit/Process left sub tree, then right sub tree and then the root element.

- procedure:-
  1) visit left sub tree in postorder
  2) visit right sub tree in postorder
  3) visit root node.

Postorder traversal = L R R



- Example:-



Postorder traversal = D → E → B → F → G → C → A

Que- Explain Binary search tree and operations of Binary search tree.

⇒

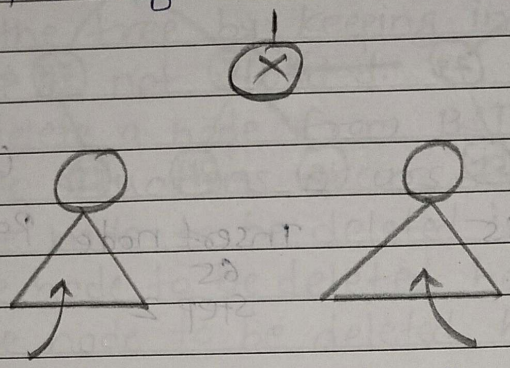
Binary search tree :-

- A binary search tree follows some order to arrange the elements.
- In a Binary search tree, the value of left node must be smaller than the parent node, and the value of right node must be greater than the parent node.
- This rule is applied recursively to the left and right subtrees of the root.
- A binary search tree (BST) or "ordered binary tree" is a empty or in which each node contain a key that satisfies following conditions :-
  1) All keys are distinct
  2) all elements in its left subtree are less to the node (<), and all the elements in its right subtree are greater than the node (>).

- Advantages of Binary search tree :-
- searching an element in the Binary search tree is easy as we always have a hint that which subtree has the desired element.
- As compared to array and linked lists, insertion and deletion operations are faster in BST.

- Binary search tree property :-
  For every node x
  - All the keys in its left subtree are smaller than the key value in x.

- All the keys in its right subtree are larger than the key value in x.



For any node y in this subtree. key(y) < key(x)

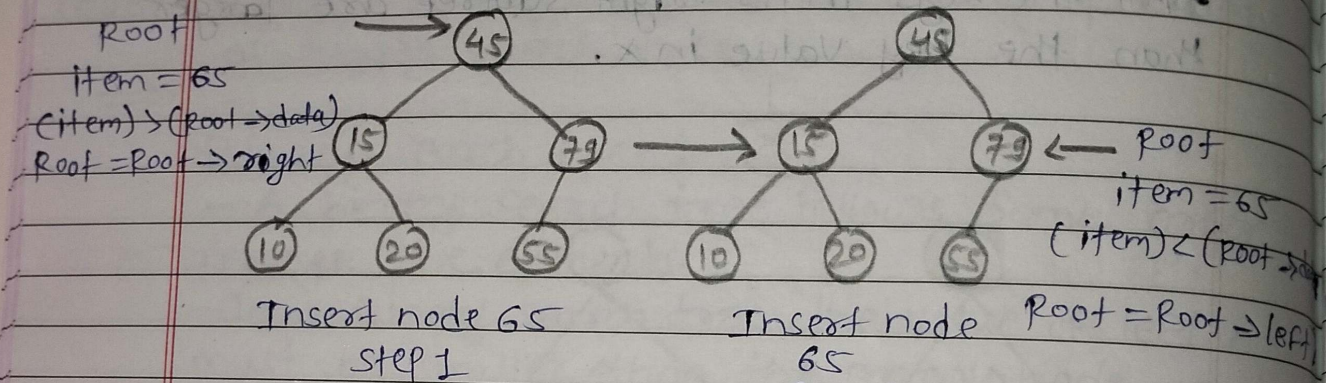
For any node z in this subtree key(z) > key(x).

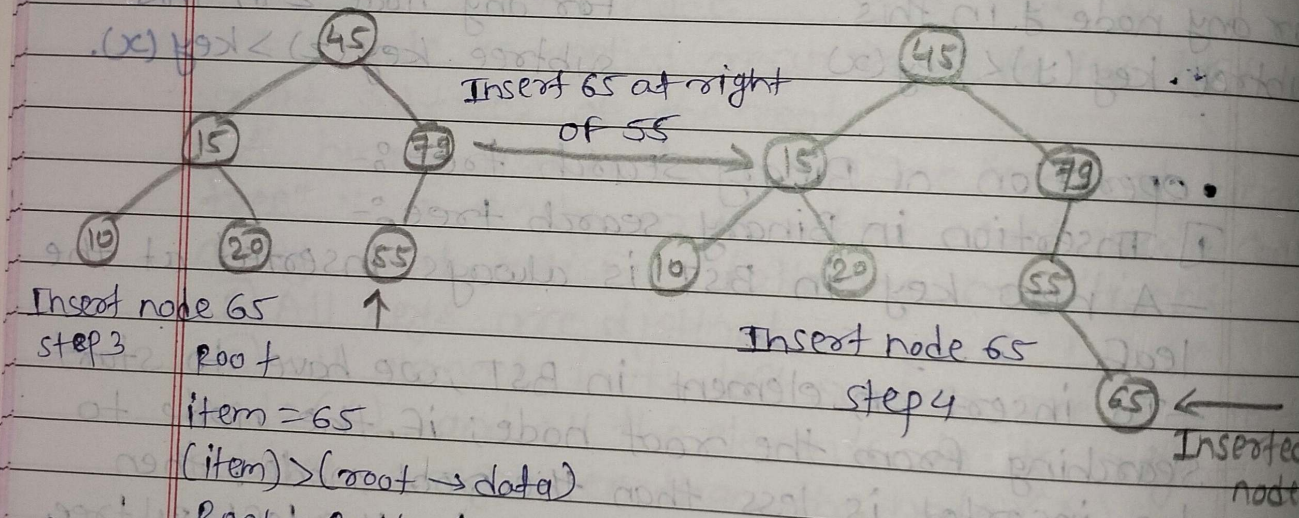
- operation of Binary search tree :-

1] Insertion in Binary search tree :-

- A new key in BST is always inserted at the leaf.
- To insert an element in BST, we have to start searching from the root node; if the node to be inserted is less than the root node, then search for an empty location in the left subtree.
- Else, search for the empty location in the right subtree and insert the data.
- Insert in BST is similar to searching, as we always have to maintain the rule that the left subtree is smaller than the root, and right subtree is larger than the root.

Example:-


Root
item = 65
(item) > (root → data)
Root = Root → right
45
15
79
10
20
55
Insert node 65
step 1
Insert node
65
step 2
Root
item = 65
(item) < (root →
Root = Root → left


Insert 65 at right
of 55
Insert node 65
step 3
Root
item = 65
(item) > (root → data)
Root = Root → right
Insert node 65
step 4
65
Inserted
node

2) Deletion in Binary search tree :-

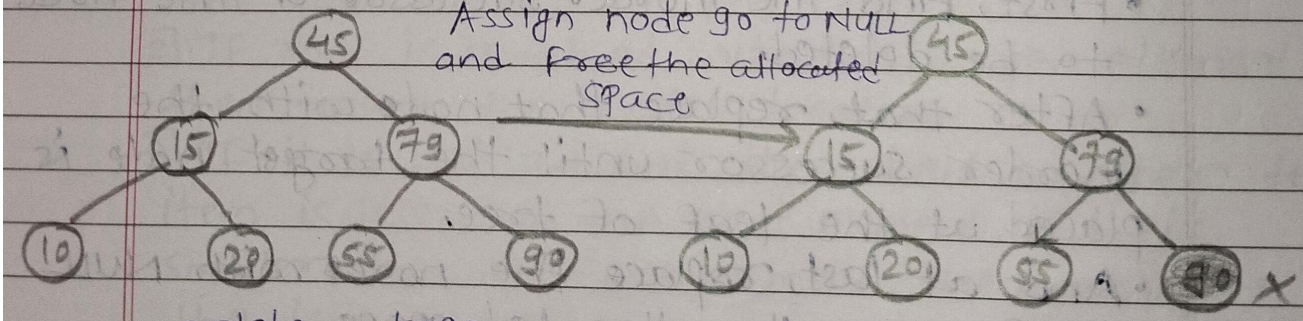
- In a binary search tree, we must delete a node from the tree by keeping in mind that the property of BST is not violated.
- To delete a node from BST, there are three possible situations occurs -
  - The node to be deleted is the leaf node
  - The node to be deleted has only one child
  - The node to be deleted has two children.

Situations :-

1) When the node to be deleted is the leaf node :-

- It is the simplest case to delete a node in BST.
- Here, we have to replace the leaf node with NULL and simply free the allocated space.

Example :-



delete node go

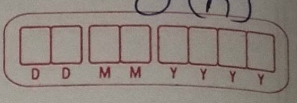
Delete node

2) When the node to be deleted has only one child.

- In this case, we have to replace the target node with its child, and then delete the child node.
- It means that after replacing the target node with its child node, the child node will now contain the value to be deleted.

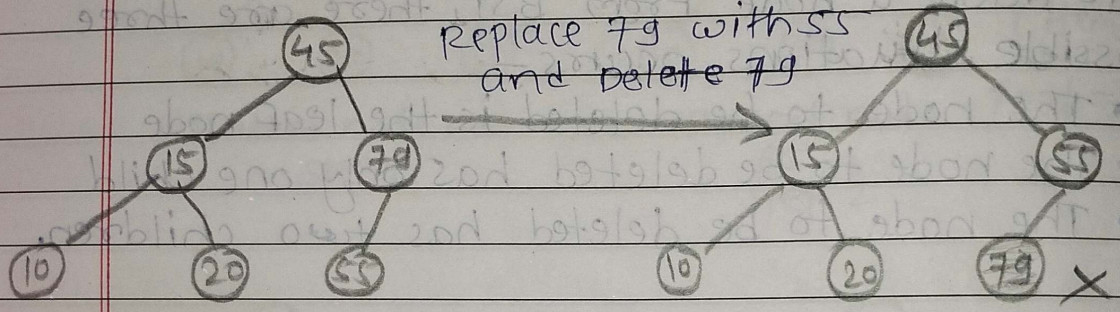
Time complexity:-
Best case - O(log n)
Average case - O(log n)
Worst case - O(n)

Complexity space:-
O(n)



-so, we simply have to replace the child node with NULL and free up the allocated space.

•Example:-

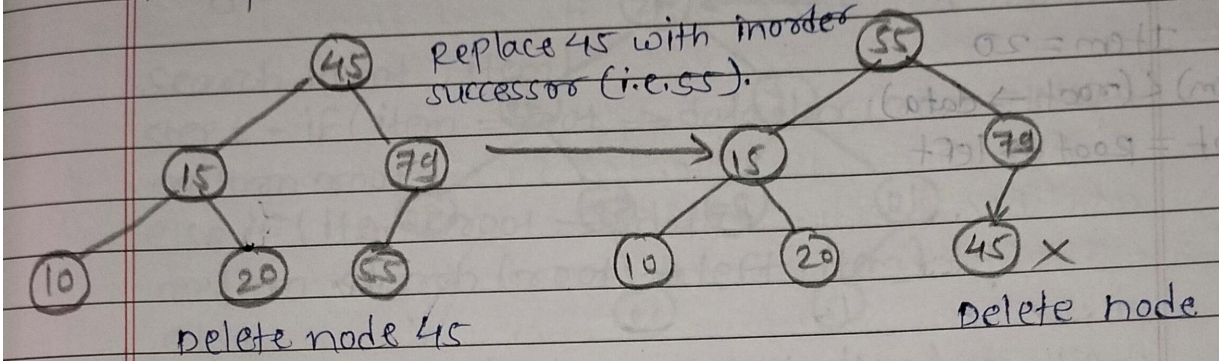


Delete node 79 Delete node

3) When the node to be deleted has two children:
-This case of deleting a node in BST is a bit complex among other two cases.
-In such a case, the steps to be followed are listed as follows -
• First, find the inorder successor of the node to be deleted.
• After that, replace that node with the inorder successor until the target node is placed at the leaf of tree.
• And at last, replace the nodes with NULL and free up the allocated space.

-The inorder successor is required when the right child of the node is not empty.
-we can obtain the inorder successor by finding the minimum element in the right child of the node.

Example:-



3] Searching in Binary search tree:-
- searching means to find or locate a specific element or node in a data structure.
- In Binary search tree, searching a node is easy because elements in BST are stored in a specific order.
- The steps of searching a node in Binary search tree are listed as follows -
1] First, Compare the element to be searched with the root element of the tree.
2] If root is matched with the target element, then return the node's location.
3] If it is not matched, then check whether the item is less than the root element, if it is smaller than the root element, then move to the left subtree.
4] If it is larger than the root element, then move to the right subtree.
5] Repeat the above procedure recursively until the match is found.
6] If the element is not found or not present in the tree, then return NULL.

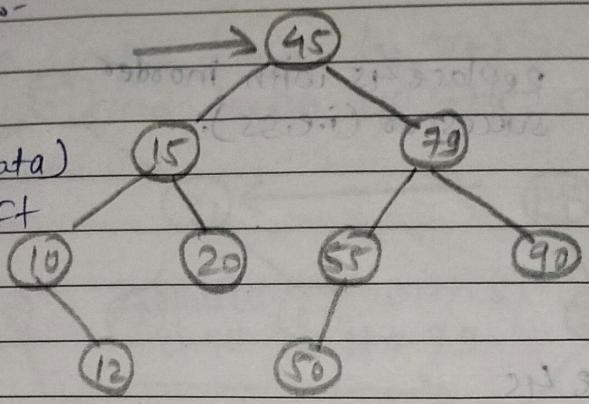
Step:-1:-

Root

Item = 20

(item) < (root → data)

Root = Root → left



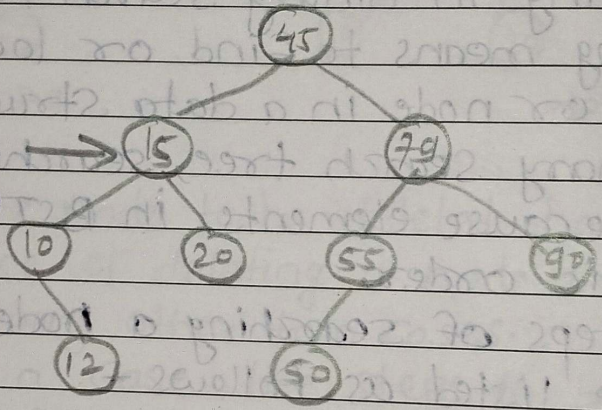
Step 2:-

root

Item = 20

(Item) > (root → data)

Root = Root → Right



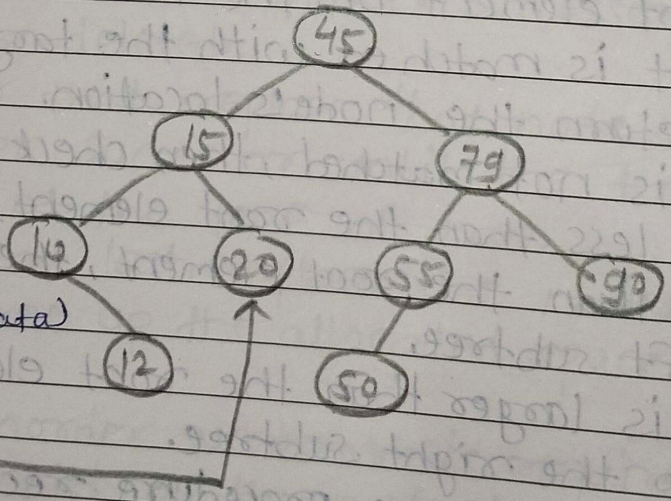
Step 3:-

root

Item = 20

(item) = (root → data)

return Root



• Algorithm to search an element i-

Search (root, item)
step 1 - if (item = root → data) or (root = NULL)
return root
else if (item < root → data)
return search (root → left, item)
else
return search (root → right, item)
END if
Step 2 - END

Que- Explain AVL tree and its rotations.

⇒

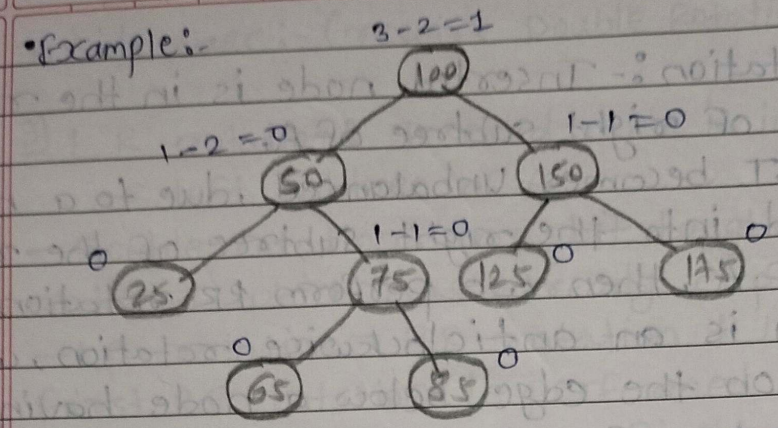
AVL tree :-

- AVL Tree is invented by GM Adelson-Velsky and EM Landis in 1962.
- The tree is named AVL in honour of its inventors.
- AVL Tree can be defined as height balanced binary search tree in which each node is associated with a balance factor which is calculated by subtracting the height of its right sub-tree from that of its left sub-tree.
- Tree is said to be balanced if balance factor of each node is in between -1 to 1, otherwise the tree will be unbalanced and need to be balanced.

Balance factor(k) = height (left(k)) - height (right(k))

- If balance factor of any node is 1, it means that the left sub-tree is one level higher than the right sub-tree.
- If balance factor of any node is 0, it means that the left sub-tree and right sub-tree contain equal height.
- If balance factor of any node is -1, it means that the left sub-tree is one level lower than the right sub-tree.

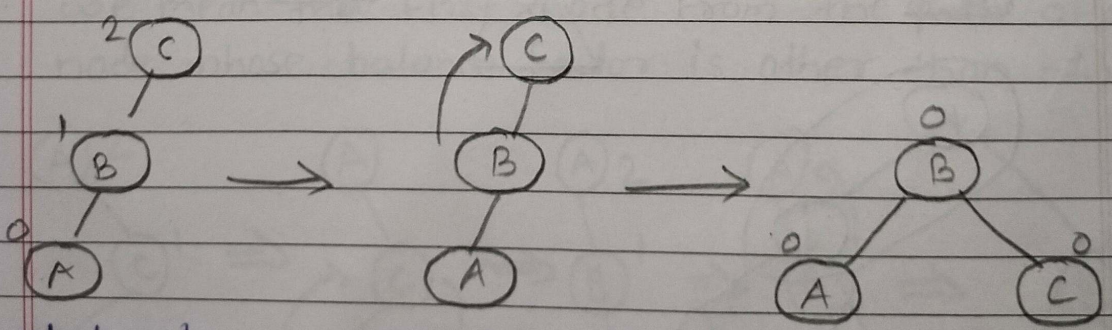
•Example:-



AVL tree

•Complexity:-
Time complexity = O(logn)
space complexity = O(n)

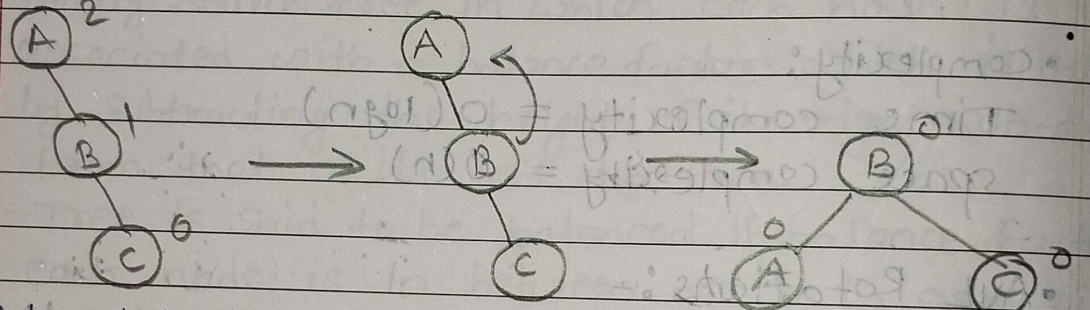
•AVL Rotations:-
There are basically four types of rotations which are as follows:-
outside cases (require single rotation)
1] LL rotation:- Inserted node is in the left subtree of Left subtree of A.
-When BST becomes unbalanced, due to a node is inserted into the left subtree of the left subtree of C, then we perform LL rotation, LL rotation is clockwise rotation, which is applied on the edge below a node having balance factor 2.



Left unbalanced Tree
Right Rotation
Balanced Tree.

2] R R rotation :- Inserted node is in the right subtree of right subtree of A.

- When BST becomes unbalanced, due to a node is inserted into the right subtree of the right subtree of A, then we perform RR rotation, RR rotation is an anticlockwise rotation, which is applied on the edge below a node having balance factor -2.



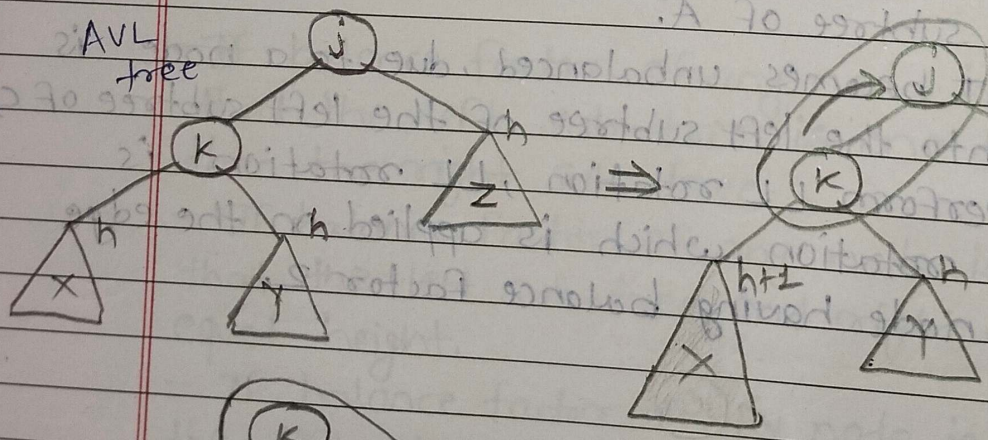
Right unbalanced tree

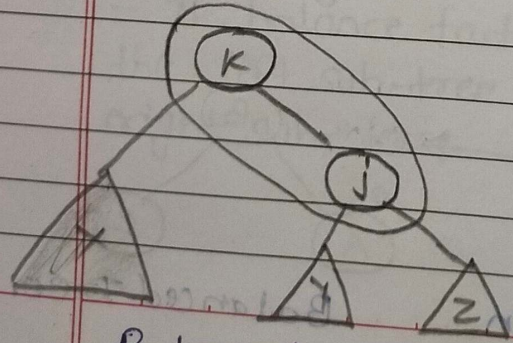
Left Rotation

Balanced

- AVL Insertion :- outside case

Inserting into X destroys the AVL property at node j



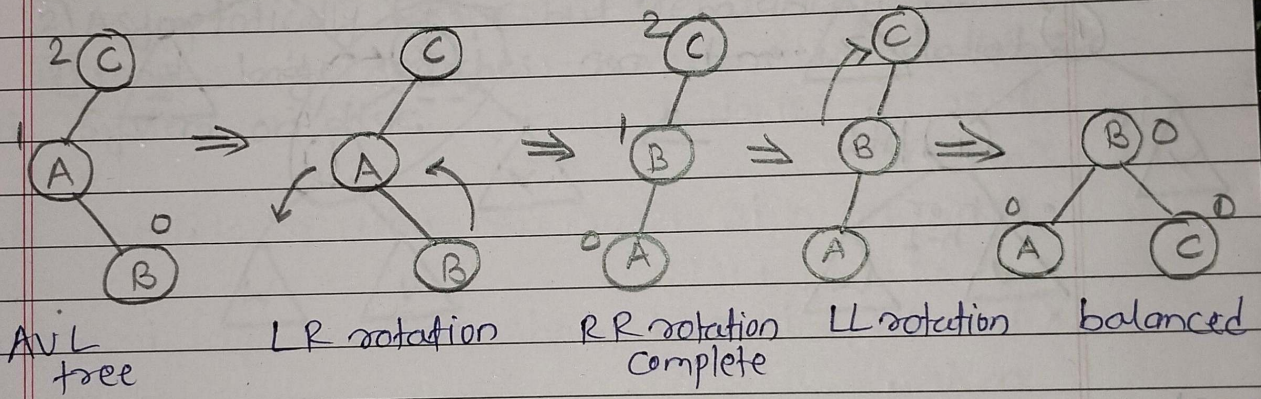


Balanced

Inside case:- (require Double Rotation)

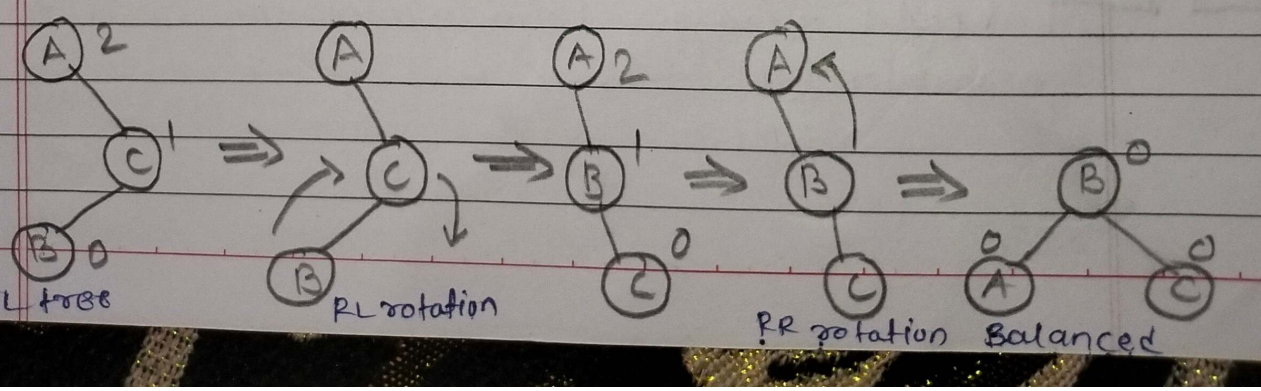
3] LR rotation: Inserted node is in the right subtree of left subtree of A.

- Double rotations are bit tougher than single rotation which has already explained above.
- LR rotation = RR rotation + LL rotation i.e. First RR rotation is performed on full tree, by full tree we mean that first node from the path of inserted node whose balance factor is other than -1, 0 or 1.



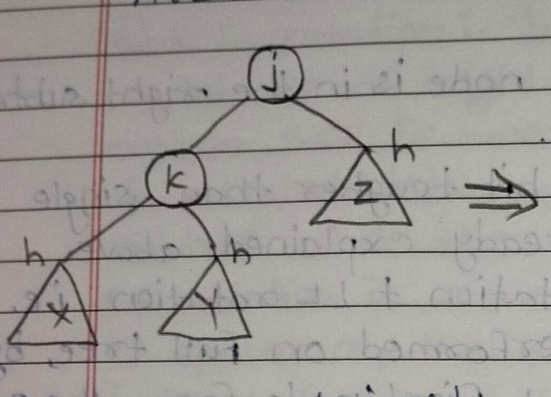
4] RL rotation: Inserted node is in the left subtree of right subtree of A.

- RL rotation = LL rotation + RR rotation i.e. First LL rotation is performed on subtree and then RR rotation is performed on full tree, by full tree we mean the first node from the path of inserted node whose balance factor is other than -1, 0, or 1.

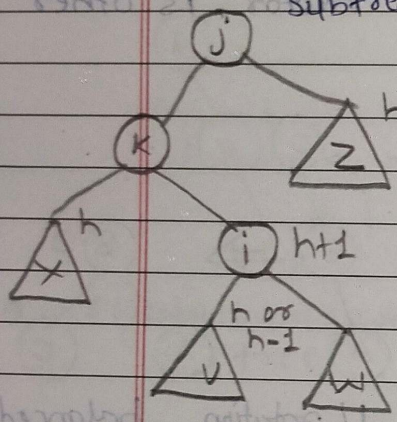


• AVL Insertion : Inside Case

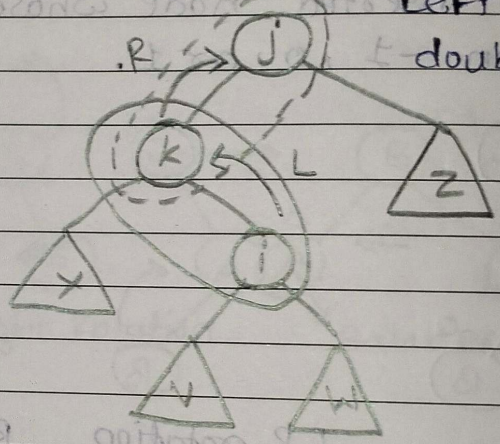
Inserting into Y destroys the AVL property at node j



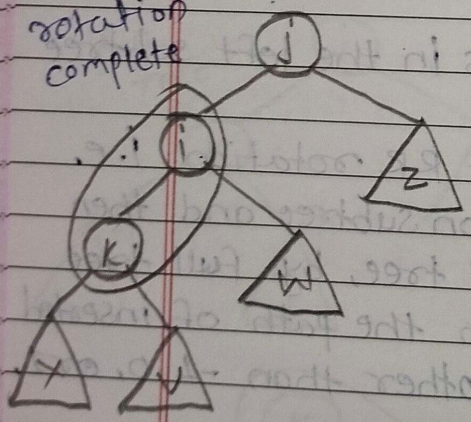
Y = node i and subtree V and W



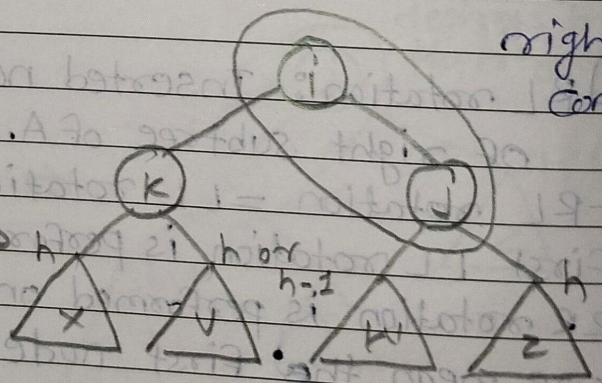
Left-right double rotation



Left rotation complete



right rotation complete



Balanced.

Advantages of AVL tree:-

1) search is O(logN) since AVL trees are always balanced.
2) Insertion and Deletion are also O(logn)
3) The height balancing adds no more than a constant factor to the speed of insertion

Disadvantages of AVL tree:-

1) Difficult to program and debug; more space for balance factor
2) Asymptotically faster but rebalancing cost time.
3) most large searches are done in database system on disk.

Que- Explain M-Way tree with examples.

⇒

M-Way tree :-

- A multiway tree is defined as a tree that can have more than two 'children'.
- If a multiway tree can have maximum m children, then this tree is called as multiway tree of order m (or an m-way tree).
- The nodes in an m-way tree will be made up of m-1 key fields and pointers to children.
- By definition an m-way search tree is a m-way tree in which following condition should be satisfied -
  - Each node is associated with m children and m-1 key fields
  - The keys in each node are arranged in ascending order.
  - The keys in the first j children are less than the j-th key.
  - The keys in the last m-j children are higher than the j-th key.

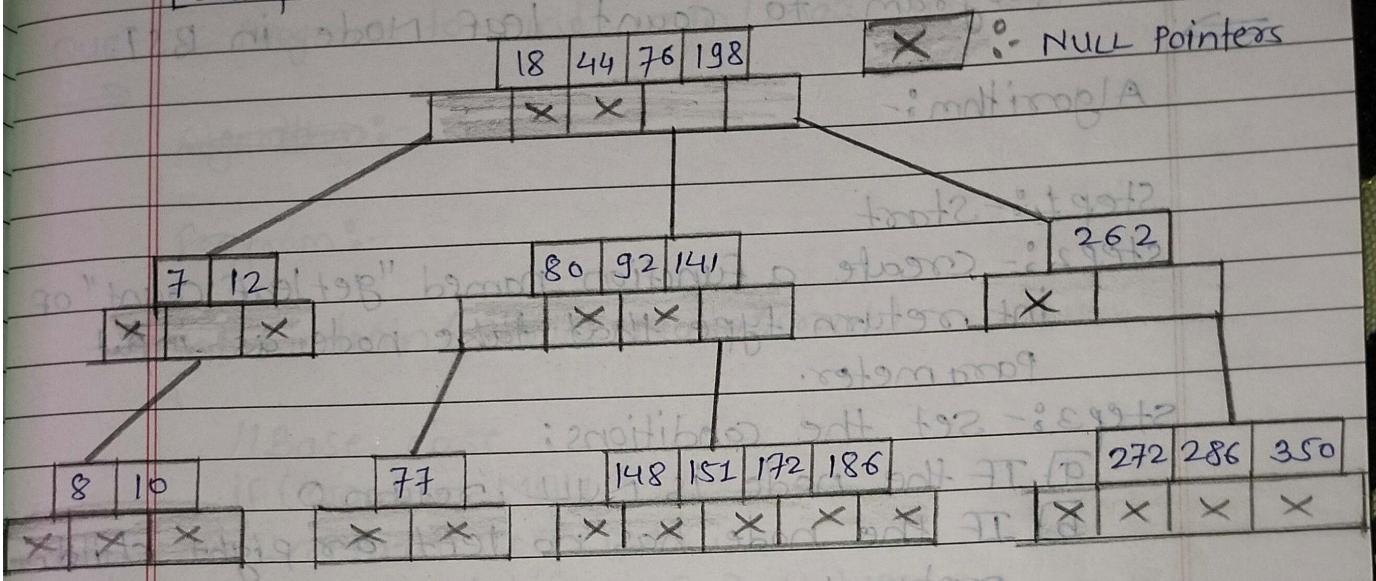
- In an m-way tree of order m, each node contains a maximum of m-1 elements and m children.
- The number of elements in an m-way search tree of height h ranges from a minimum of h to a maximum of $m^h - 1$.

Key Field

: Pointers to subtree

: NULL Pointers

Example:-



M-Way tree

Que- C Program to count leaf Node in BST.

⇒

Algorithm:-

Step 1:- Start
step 2:- create a function named "getleaf count" of int return type that take node as input Parameter.
step 3:- set the conditions:
a] If the node is NULL, return 0
b] If the node has no left or right child, return 1.
c] Recursively call "getleafcount" on the left and right child nodes if the node has left or right children, and then return the total of the results.
step 4:- End

Program:-

```
usigned int getLeafCount (struct node *node)
{
  if (node == NULL)
     return 0;
  if (node -> left == NULL && node -> right == NULL)
    return 1;
  else
    return getLeafCount (node -> left) +
           getLeafCount (node -> right);
}
```

Que- C Program to find min/max in BST.

→

~~Algorithm~~:-

Program:-

```
int Findmin(struct Node*root)
{
   //Base case
   if(root==NULL)
      return INT_MAX;
  // Return minimum of 3 values:
  1) Root's data   2) Max in Left
  3) Max in right subtree

  int res = root -> data;
  int lres = Findmin(root->left);
  int rres = Findmin(root->right);
  if (lres < res)
     res = lres;
  if(rres < res)
    res = rres;
    return res;
}
```