Definitions of Tree:

Tree is a non-linear data structure which organizes data in hierarchical structure and this is a recursive definition.

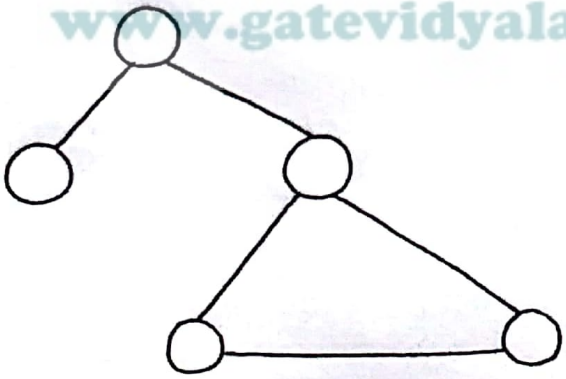
OR

A tree is a connected graph without any circuits.

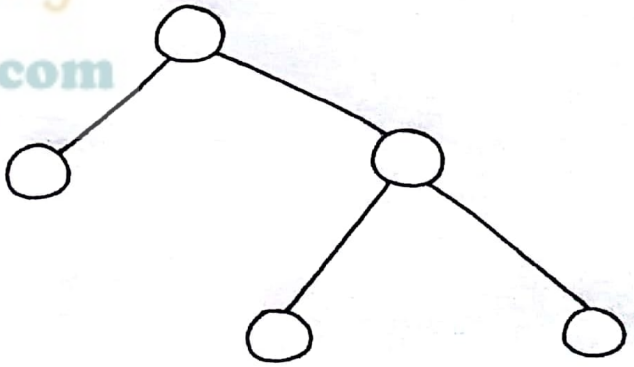
OR

If in a graph, there is one and only one path between every pair of vertices, then graph is a tree.

Example:



Graph which is not a tree

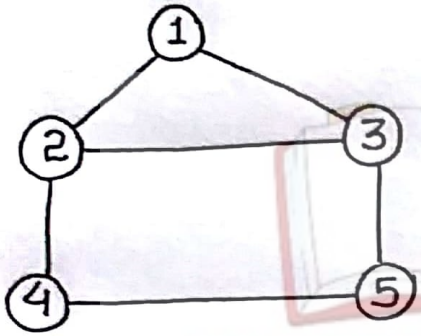


Graph which is a tree

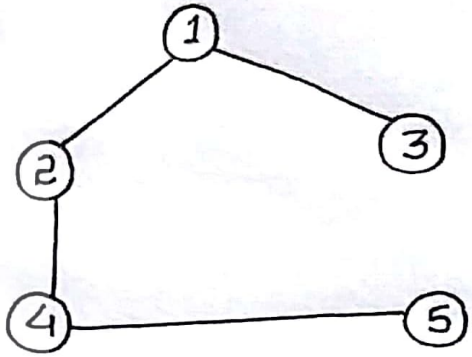
Question:

Which of the following graphs are trees ?

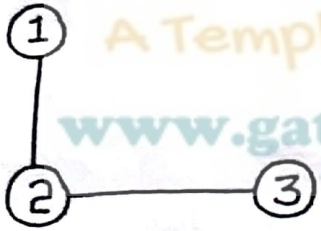
(A)



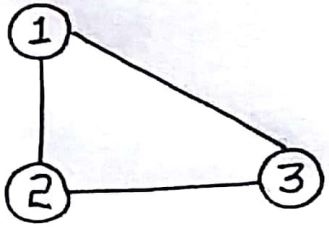
(B)



(C)



(D)



Solution: Only (B) and (C) are trees

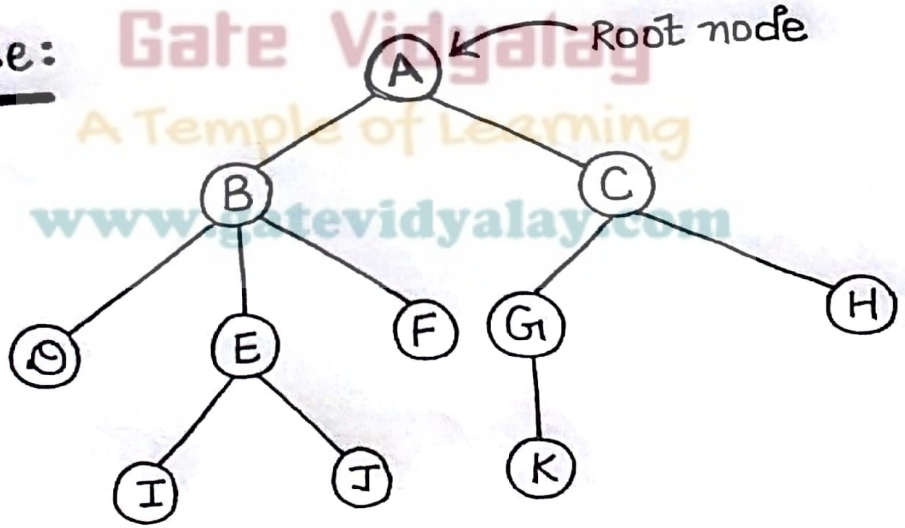
Properties of Trees:

- There is one and only one path between every pair of vertices in a tree.
- A tree with n vertices has $n-1$ edges.
- A graph is a tree if and only if it is minimally connected.
- Any connected graph with n vertices and $n-1$ edges is a tree.

Basic Terminology...:

1) Root:

- Root node is the origin of tree data structure. It is the first node.
- In any tree, there must be only one root node and we can never have multiple root nodes in a tree.
- **Example:**

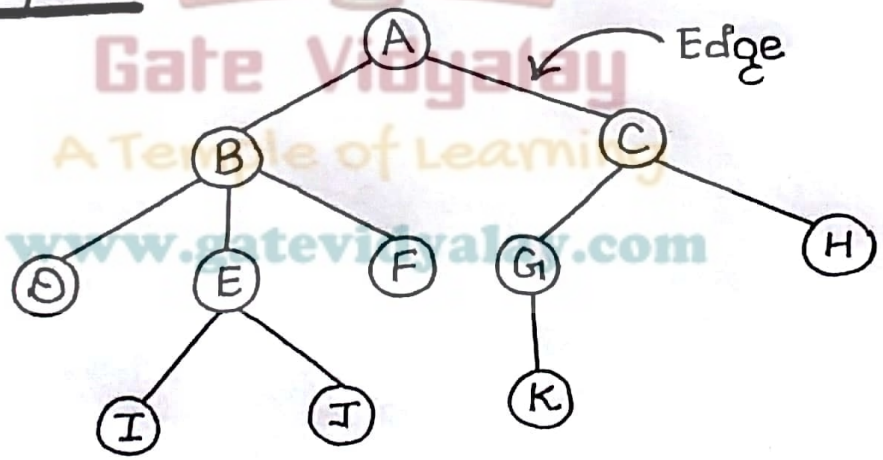


2) Edge:

- The connecting link between any two nodes is called as edge.

Remember: In a tree with 'n' number of nodes, there will be exactly 'n-1' number of edges

- **Example:**

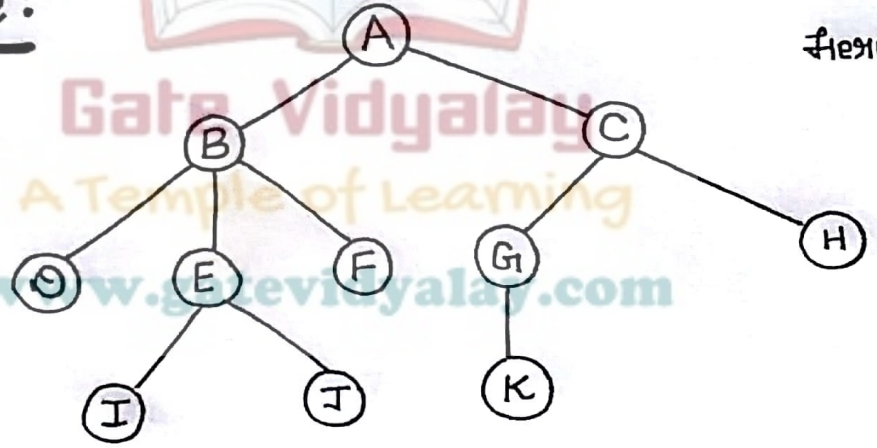


3. Parent:

- The node which has a branch from it to any other node is called as parent node.

  In other words, the node which has child/children is called as parent node.

- **Example:**

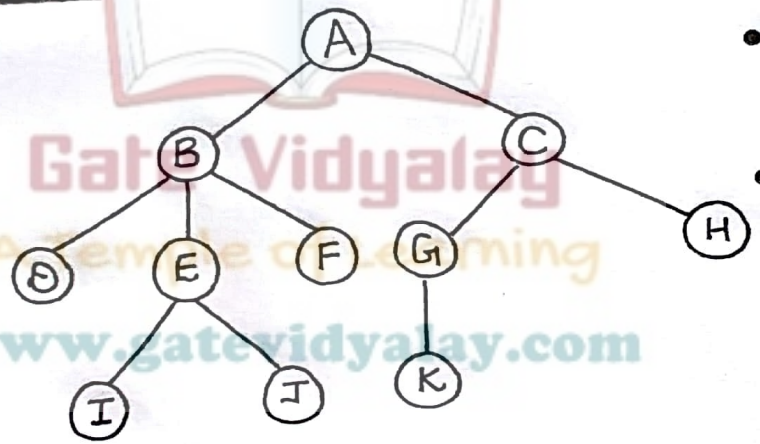


Here, A, B, C, E and G are parent nodes

4. Child :

- The node which is descendant of any node is called as child node . So, all the nodes except root node are child nodes.

- **Example:**



- Here, B and C are children of A
- G and H are children of C
- K is the only child of G etc

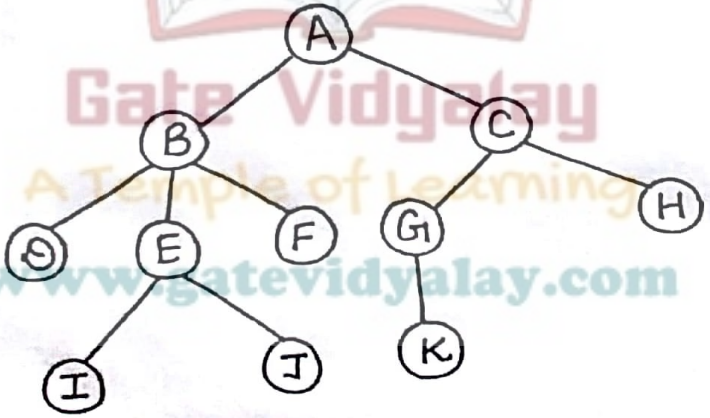
Note: In a tree, any parent node can have any number of child nodes.

5. Siblings :

- Nodes which belong to the same parent are called as siblings.

  In other words, nodes with the same parent are called as sibling nodes.

- **Example :**

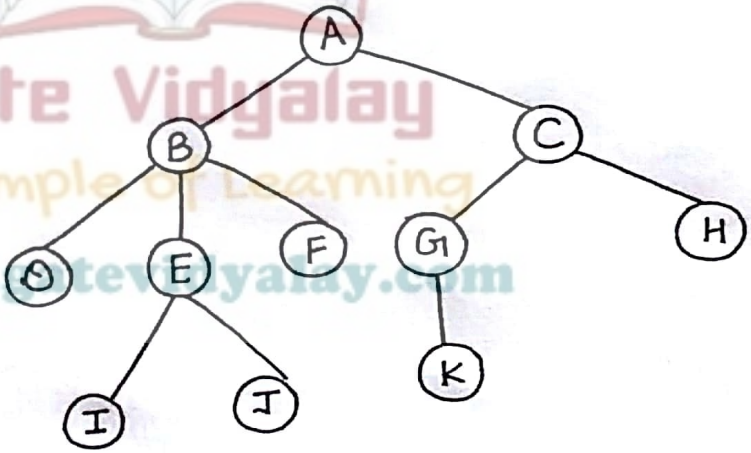


Here,

- B and C are siblings
- D, E and F are siblings
- G and H are siblings
- I and J are siblings

6. Degree:

- The total number of children of a node is called as degree of that node.
- Degree of a tree is the highest degree of a node among all the nodes in the tree.
- **Example:**



Here,

- Degree of B = 3
- Degree of A = 2
- Degree of F = 0

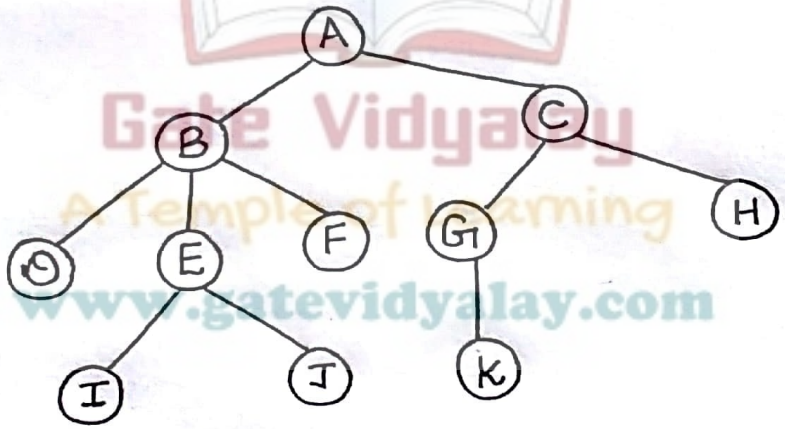
etc

7. Internal Nodes:

- The node which has atleast one child is called as internal node

  They are also called as non-terminal nodes.

- **Example:**



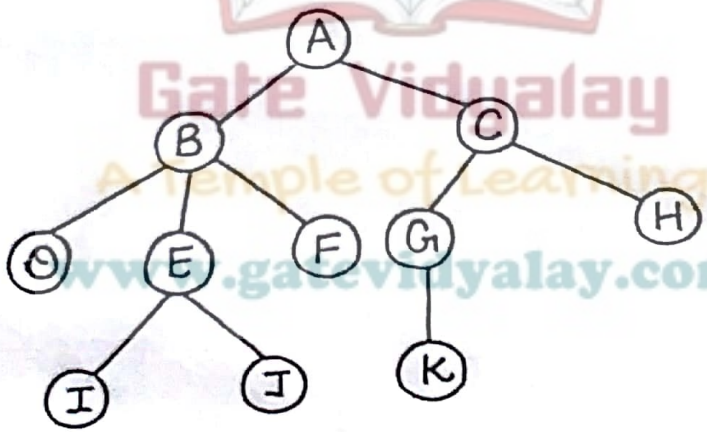
Here,

A, B, C, E and G are internal nodes

Note: Every non-leaf node is an internal node.

8. Leaf Nodes :

- The node which does not have a child is called as leaf node.
- The leaf nodes are also called as external nodes or terminal nodes.
- **Example:**



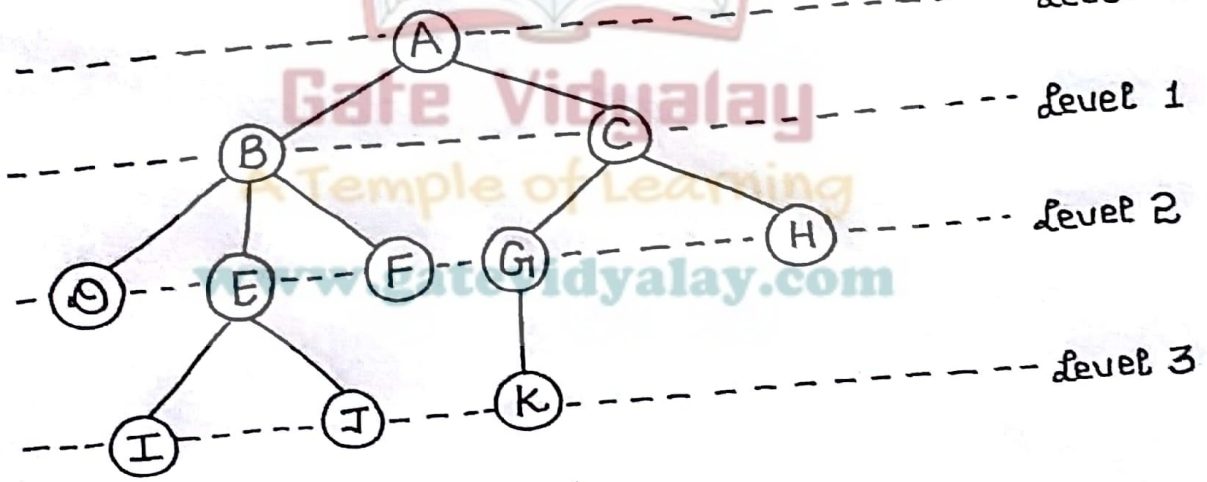
Here,

D, I, J, F, K and H are leaf nodes

9. Level:

- In a tree, each step from top to bottom is called as level and the level count starts with '0' and incremented by one at each level or step.

- **Example:**



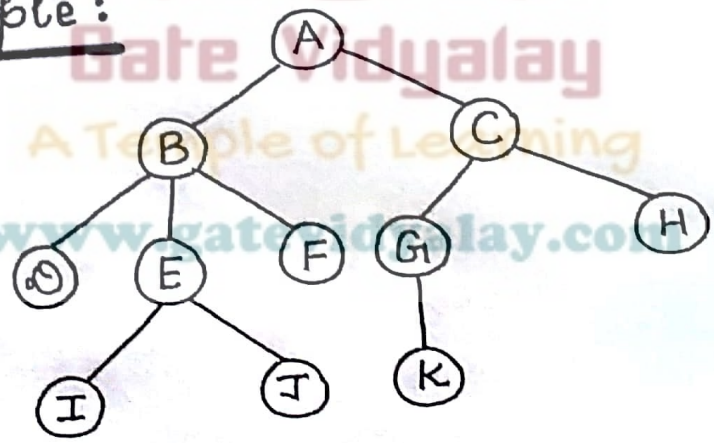
10. Height:

- The total number of edges from leaf node to a particular node in the longest path is called as height of that node.

Note:

- Height of the tree = Height of the root node
- Height of all leaf nodes = 0

- **Example:**



Here,

- Height of K = 0
- Height of B = 2
- Height of A = 3
- Height of G = 1
- Height of H = 0

etc

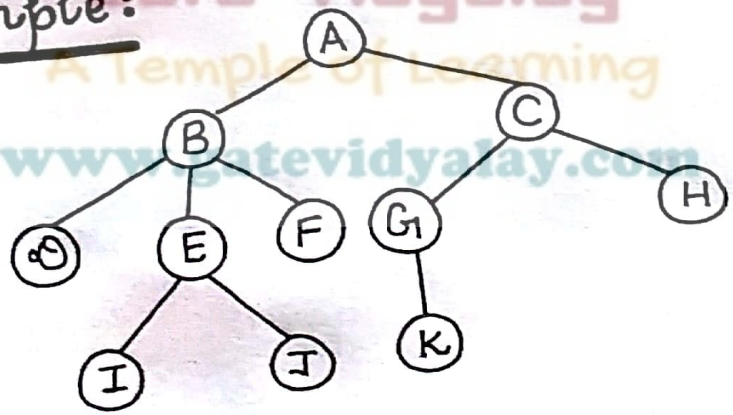
11. Depth:

- The total number of edges from root node to a particular node is called as depth of that node.

Note:

- Depth of the tree = Total number of edges from root node to a leaf node in the longest path
- Depth of the root node = 0

- **Example:**



Here,

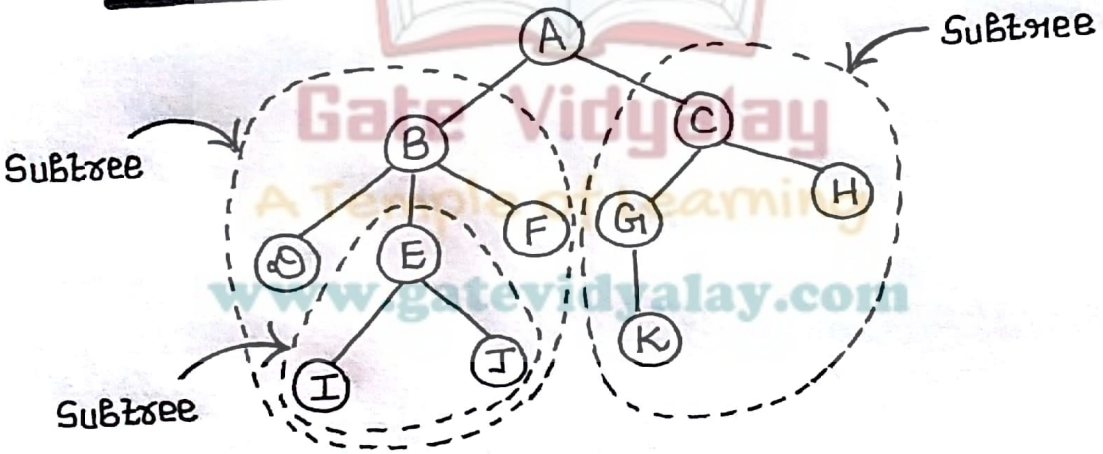
- Depth of A = 0
- Depth of B = 1
- Depth of K = 3

etc.

12. Subtree :

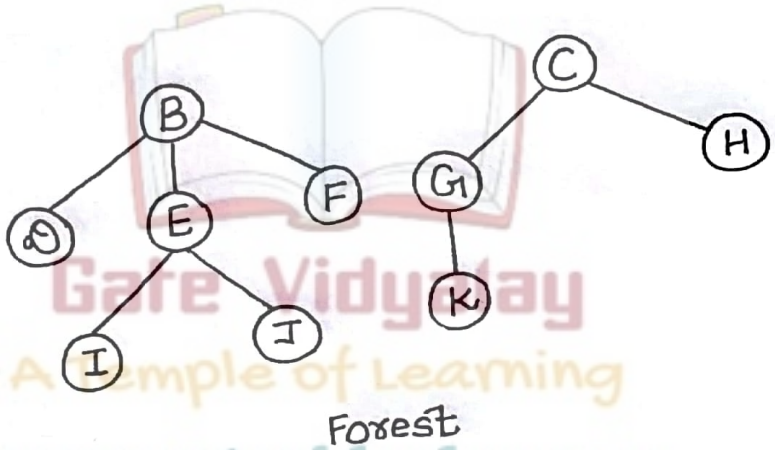
- In a tree, each child from a node forms a subtree recursively. Every child node will form a subtree on its parent node.

- **Example :**



13. Forest:

- It is a set of disjoint trees.
- **Example:**



Forest

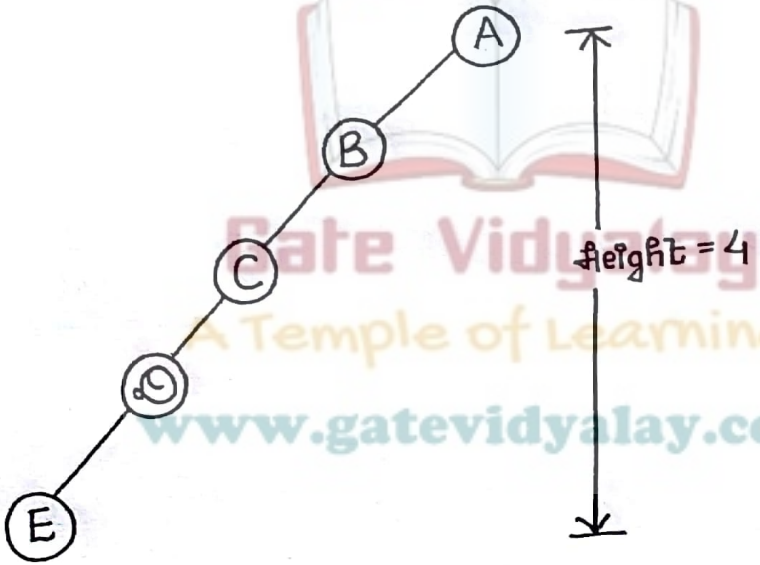
Important Formulas :-

Formula-1:

Minimum number of nodes in a binary tree of height `H' = H+1

Example:

To construct a binary tree of height '4' we need at least 5 nodes.



Formula-2:

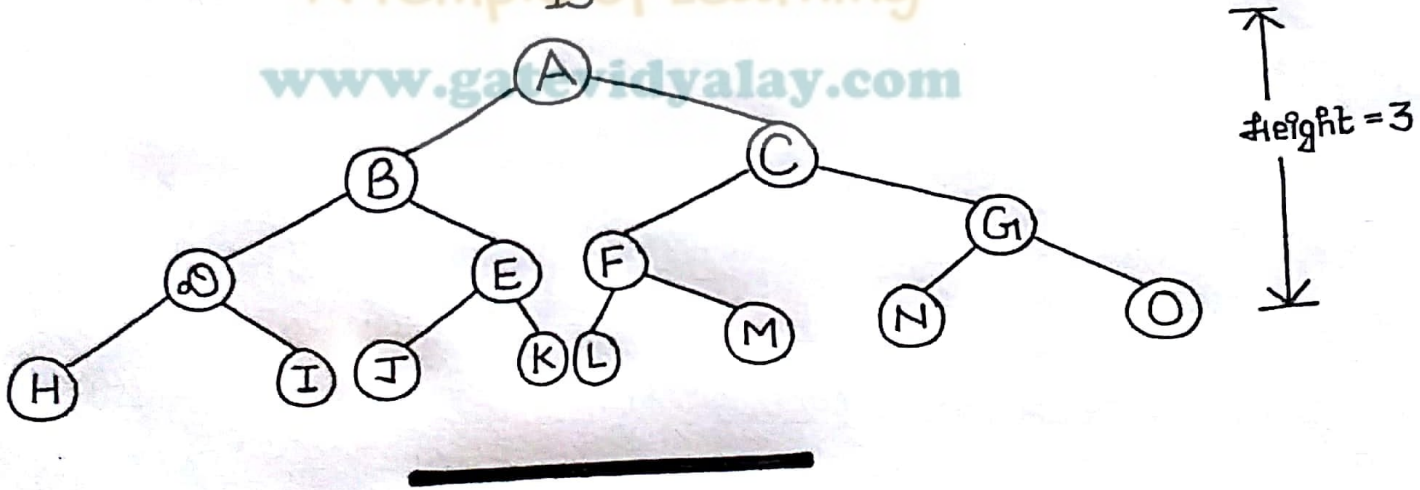
> Maximum number of nodes in a binary tree of height 'h'
>
> $= 2^{h+1} - 1$

Example:

Maximum number of nodes in a binary tree of height '3'

$= 2^{3+1} - 1$

$= 15$

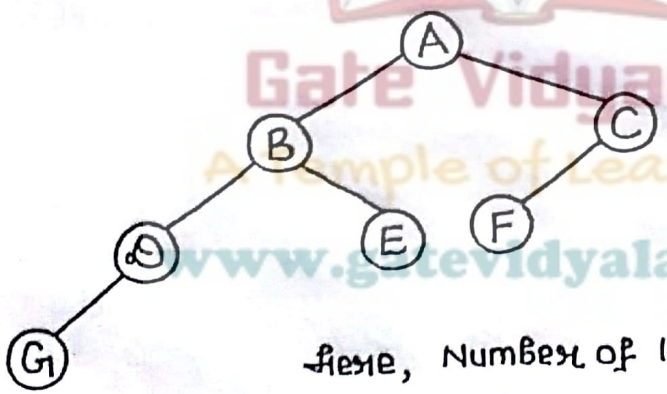


Formula-3:

Total number of leaf nodes in binary tree	=	Total number of degree-2 nodes	+ 1

Example:

Consider the following binary tree-



Here, Number of leaf nodes = 3

Number of degree-2 nodes = 2

This verifies the above relation.

Formula-4:

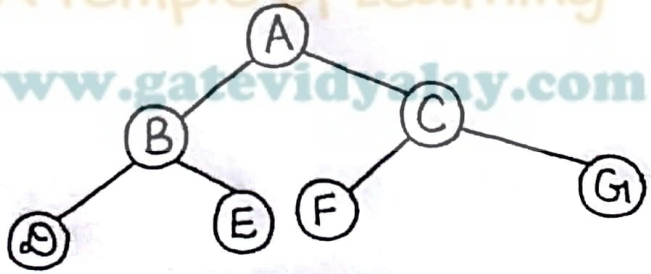
> Maximum number of nodes at any level 'L'
>
> $= 2^L$

Example:

Maximum number of nodes at level '2'

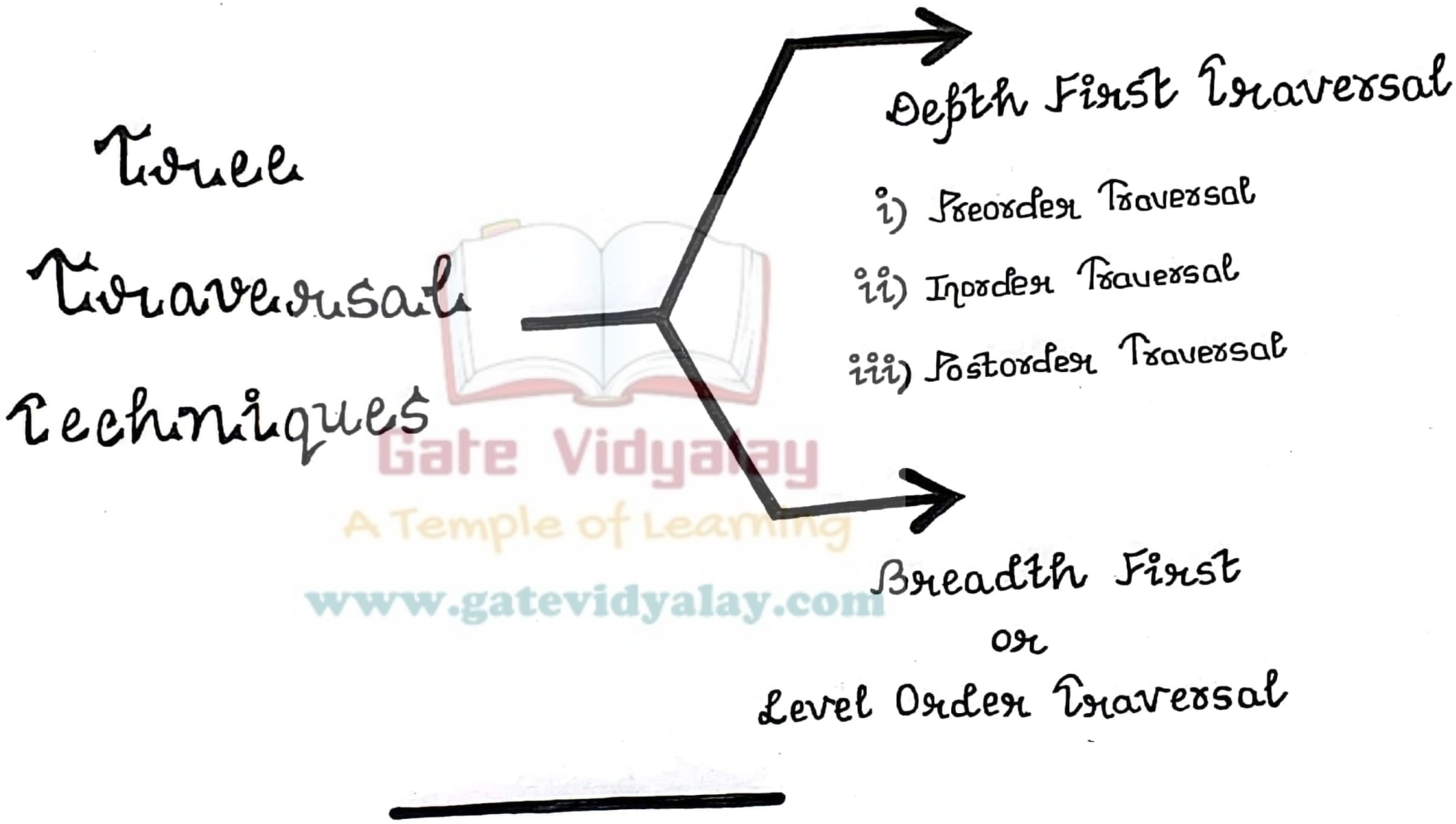
$= 2^2$

$= 4$



Tree Traversal:

Tree traversal refers to the process of visiting each node in a tree data structure exactly once.



I) Preorder Traversal :

Algorithm:

i) Visit the root

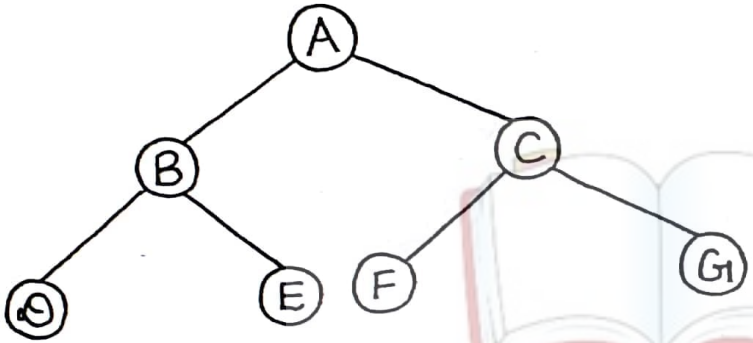
ii) Traverse the left subtree i.e. call Preorder (left subtree)

iii) Traverse the right subtree i.e. call Preorder (Right subtree)

Remember:

Root → Left → Right

Example:

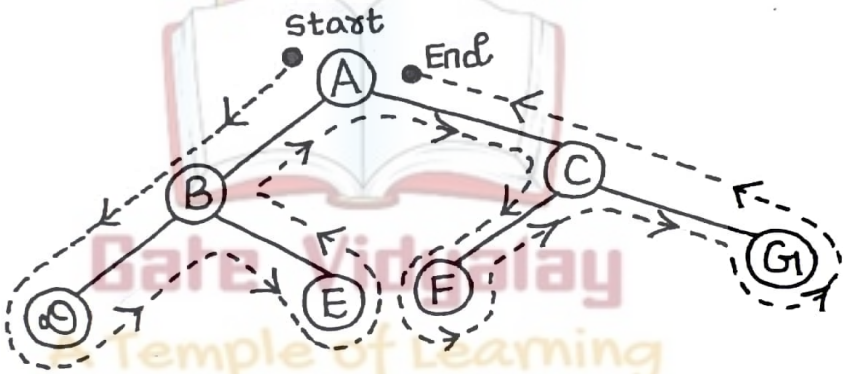


Binary Tree

Preorder Traversal: A B D E C F G1

Shortcut for Preorder Traversal :

Just traverse the entire tree starting from the root node keeping yourself to the left.



∴ Preorder Traversal = A B D E C F G1

II) In order Traversal:

Algorithm:

i) Traverse the left subtree i.e. call Inorder (left-subtree)

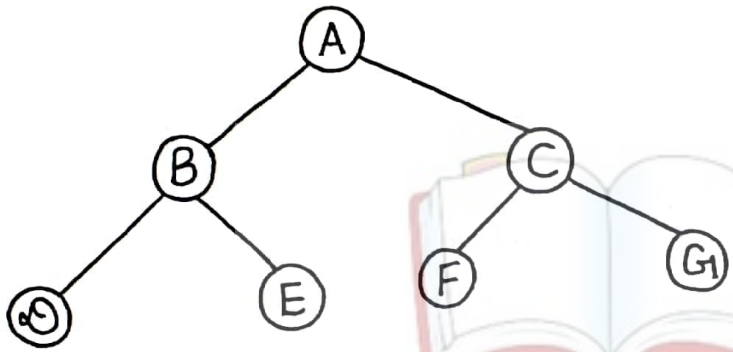
ii) visit the root

iii) Traverse the right subtree i.e. call Inorder (right-subtree)

Remember:

Left → Root → Right

Example :

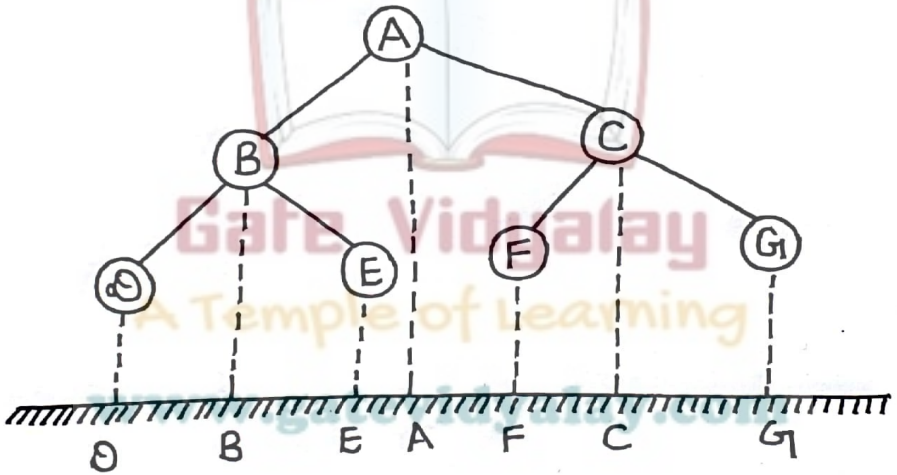


Binary Tree

Inorder Traversal : D B E A F C G

Shortcut for Inorder Traversal :

Just keep a plane mirror horizontally at the bottom of the tree and take the projection of all nodes.



∴ Inorder Traversal = D B E A F C G

iii) Postorder Traversal:

Algorithm:

i) Traverse the left subtree i.e. call Postorder (Left subtree)

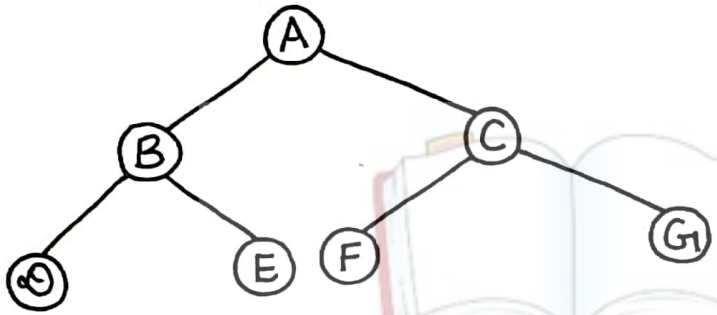
ii) Traverse the right subtree i.e. call Postorder (Right subtree)

iii) visit the root

Remember:

Left → Right → Root

Example :

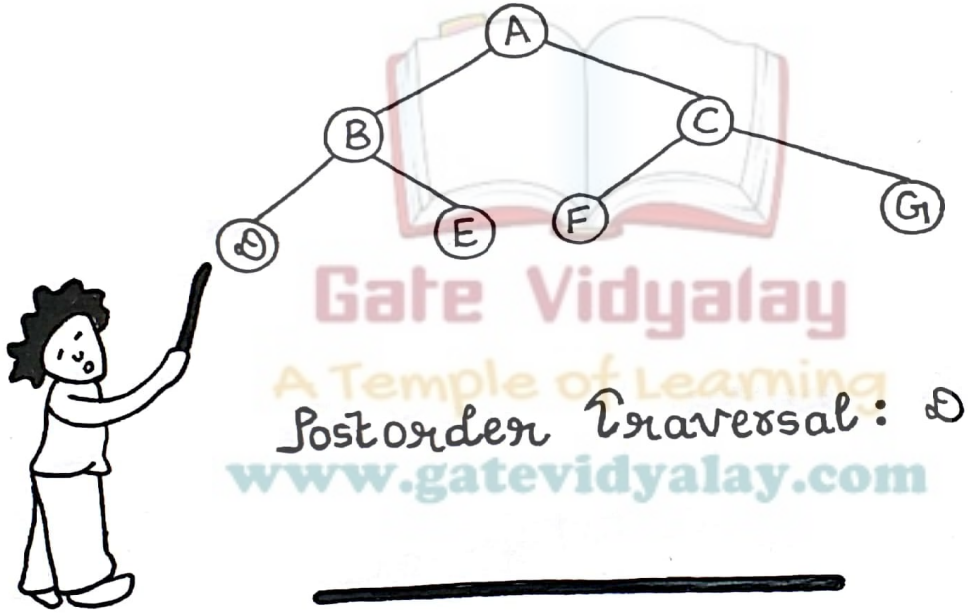


Binary Tree

Postorder Traversal : D E B F G C A

Shortcut for Postorder Traversal:

Just pluck the leftmost leaf nodes one by one.

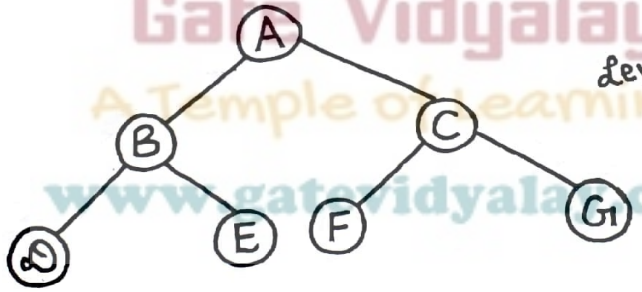


Postorder Traversal: D E B F G C A

iv) Level Order Traversal:

- Level Order Traversal of a tree is the breadth first traversal of a tree which prints all the nodes of a tree level by level.

- **Example:**



Level order Traversal :

A B C D E F G1

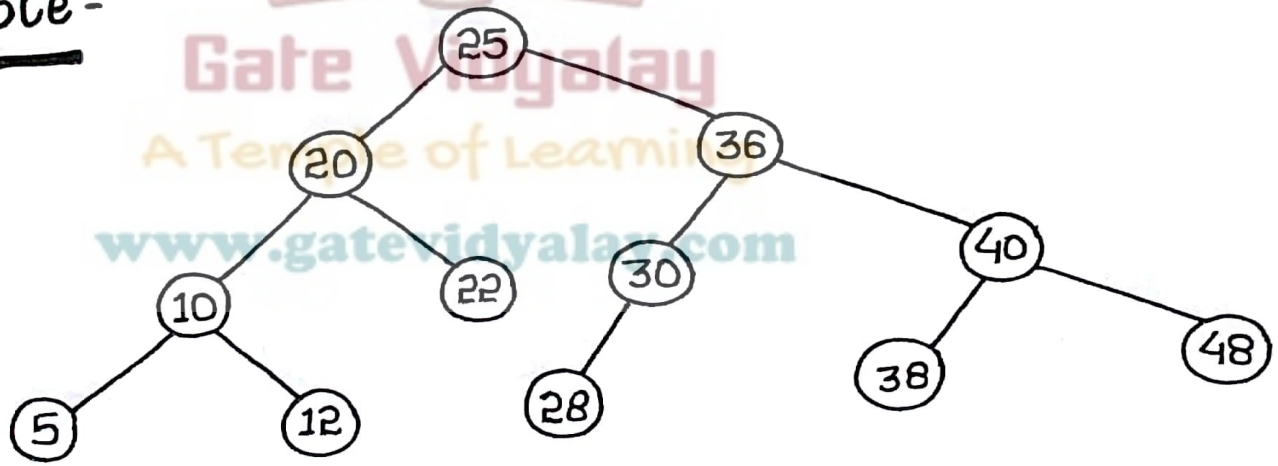
Important Points for exam:

- Preorder traversal is used to get prefix expression of an expression tree.
- Inorder traversal is used to get infix expression of an expression tree.
- Postorder traversal is used to get postfix expression of an expression tree.
- Preorder traversal is used to create a copy of the tree.
- Postorder traversal is used to delete the tree.
- Level order traversal prints the data in the same order as it is stored in the array representation of complete binary tree.

Definition-

Binary Search Tree (BST) is a special kind of binary tree in which every node contains smaller values only in its left subtree and only larger values in its right subtree.

Example-



Number of distinct BSTs with 'n' distinct keys -

$$\frac{{}^{2n}C_n}{n+1}$$

Example-

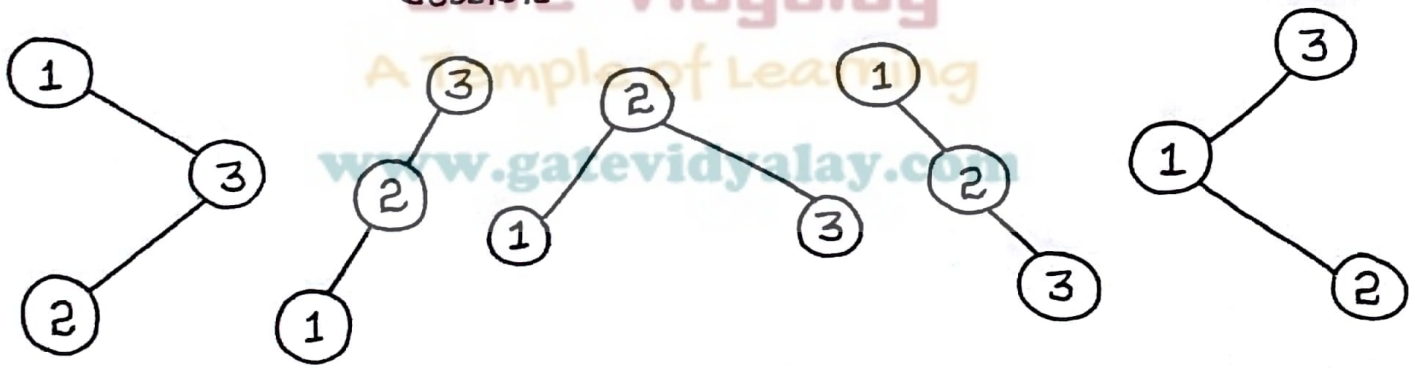
Number of distinct Binary search Trees with 3 distinct keys

$$= {}^{2\times3}C_3 \,/\, 3+1$$

$$= 5$$

Consider three distinct keys are - 1 , 2 , 3

Possible BSTs are -



Construction of BST -

Question -

Construct a Binary Search Tree (BST) for the following sequence of numbers -

50 , 70 , 60 , 20 , 90 , 10 , 40 , 100

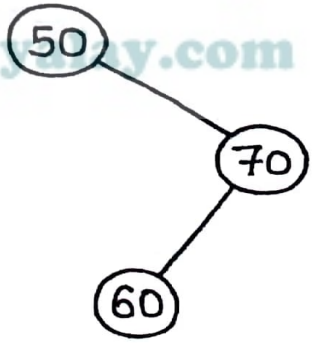
Solution -

When elements are given in a sequence, we consider the first element as the root node.

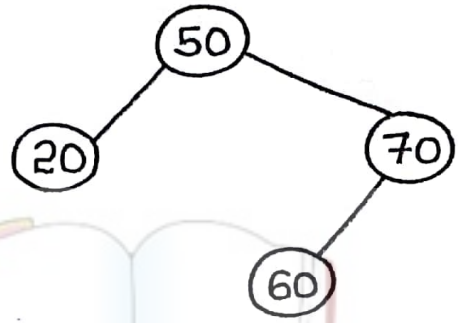
- **Insert 50 -**

- **Insert 70 -**

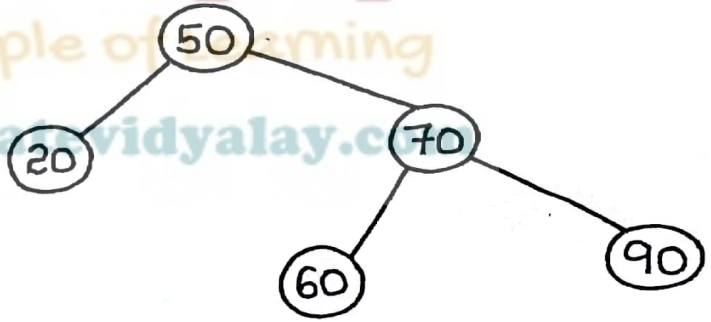
- **Insert 60 -**



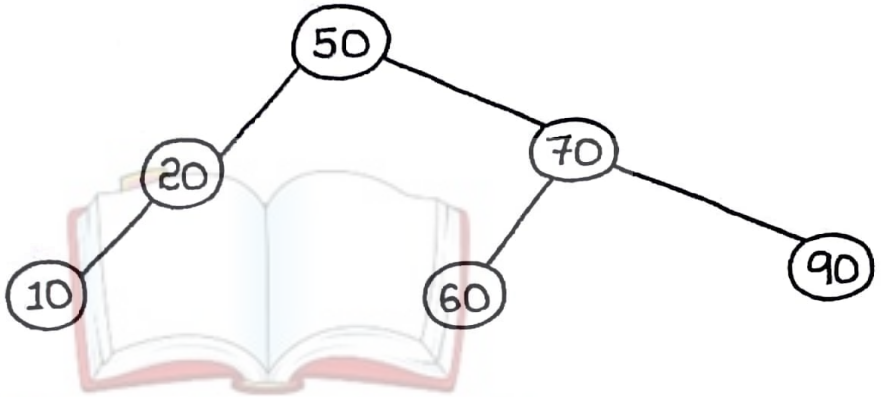
- Insert 20 -



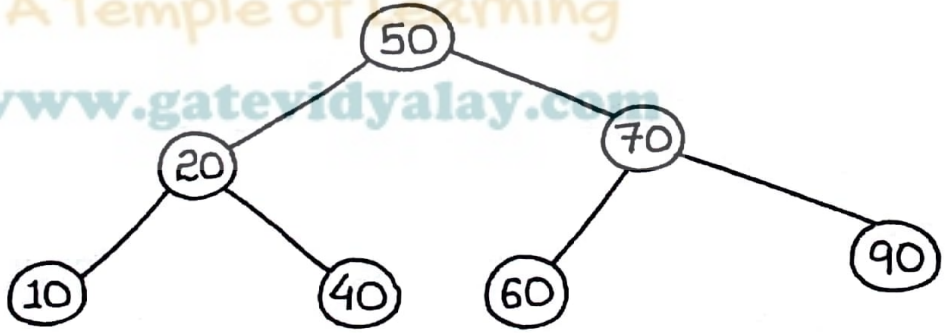
- Insert 90 -



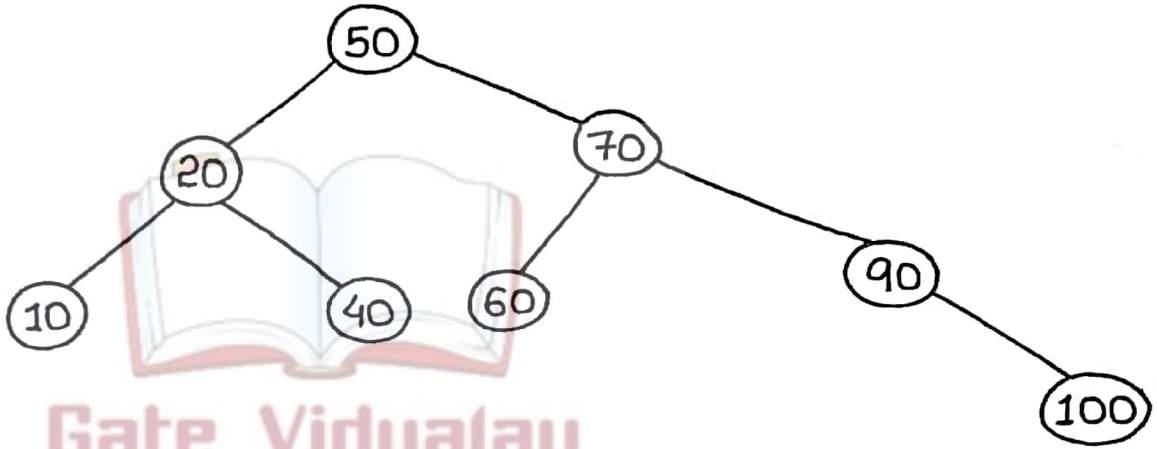
- Insert 10 -



- Insert 40 -



- **Insert 100 -**



This is the required Binary Search Tree.

Operations on BST -

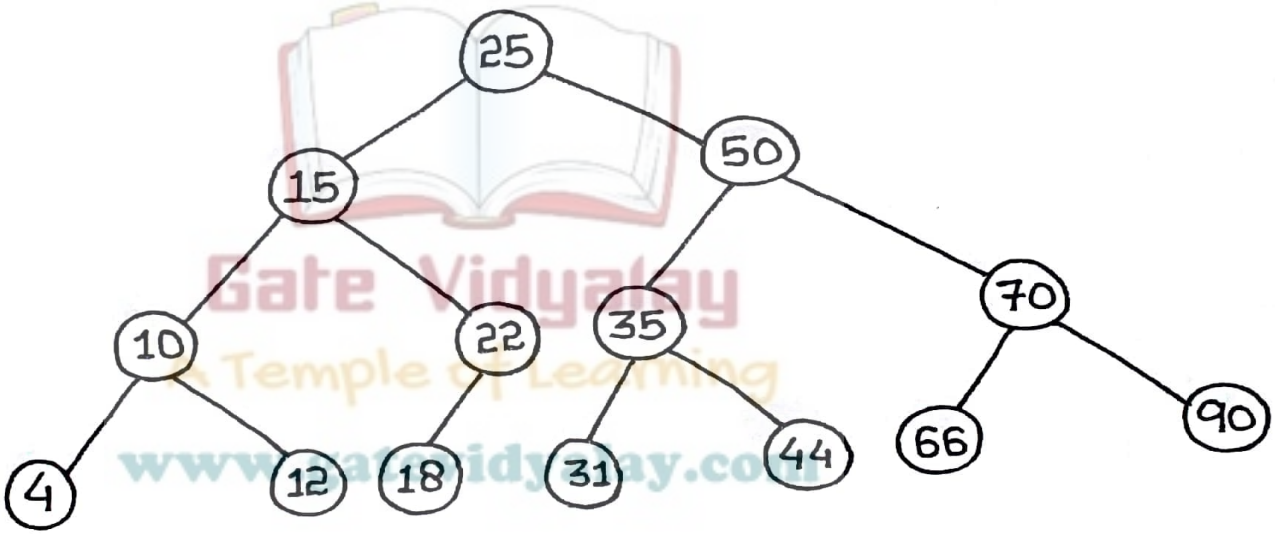
The following operations are performed on a Binary Search Tree -

Search Operation -

To search a given key in Binary Search Tree, we first compare it with root. If the key is present at root, we return root. If the key is greater than root's key, we recur for right subtree of root node otherwise we recur for left subtree.

Example-

Search for 45 in the BST-



Step-01: Start at the root. As $45 > 25$, so search in right subtree.

Step-02: As $45 < 50$, so search in 50's left subtree.

Step-03: As $45 > 35$, so search in 35's right subtree.

Step-04: As $45 > 44$, so search in 44's right subtree.

But 44 has no subtrees. So, 44 is not present in the BST.

Insertion Operation -

A new key is always inserted at leaf. We start searching a key from root till we hit a leaf node. Once a leaf node is found, the new node is added as a child of the leaf node.

Example -

Step-01: Start at root node 100. As $40 < 100$, so search in 100's right subtree.

Step-02: As $40 > 20$, so search in 20's right subtree.

Step-03: As $40 > 30$ (leaf node), so add 40 to 30's right subtree.

Deletion Operation-

Deleting a node from Binary Search Tree gives rise to following 3 cases-

Case-I: Deleting a node with no child (leaf node)

Case-II: Deleting a node with one child

Case-III: Deleting a node with two children

Case-I: Deleting a leaf node-

It is very simple. Just remove the leaf node from the tree.

Example-

Case-II: Deleting a node with one child-

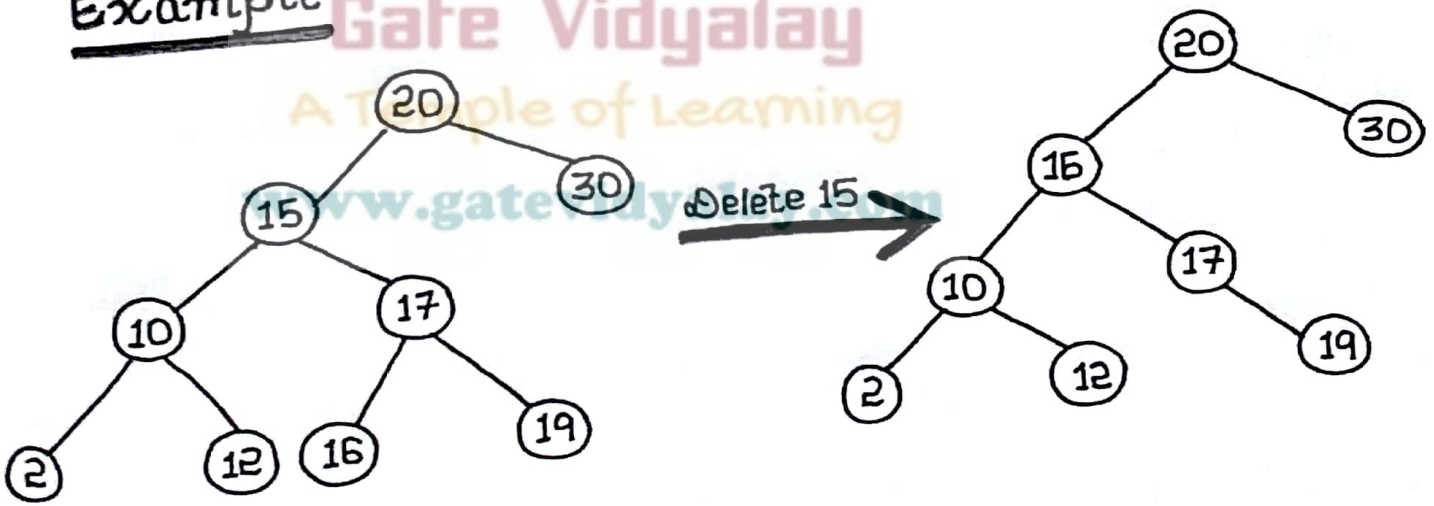
Just make the child of the deleting node, the child of the grandparent.

Example-

Case-III: Deleting a node with 2 children-

Method-I: Go to the right subtree of the deleting node, pluck the least element called inorder successor and replace with the deleting node.

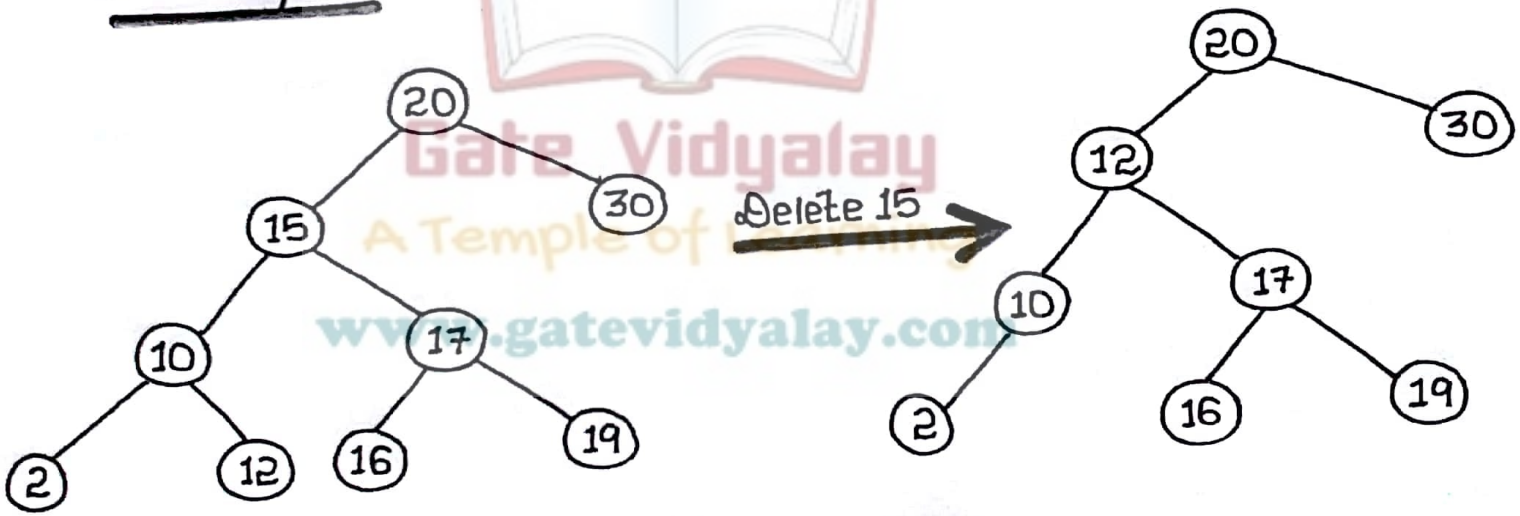
Example-



Method-II:

Go to the left subtree of the deleting node, pluck the greatest element called inorder predecessor and replace with the deleting node.

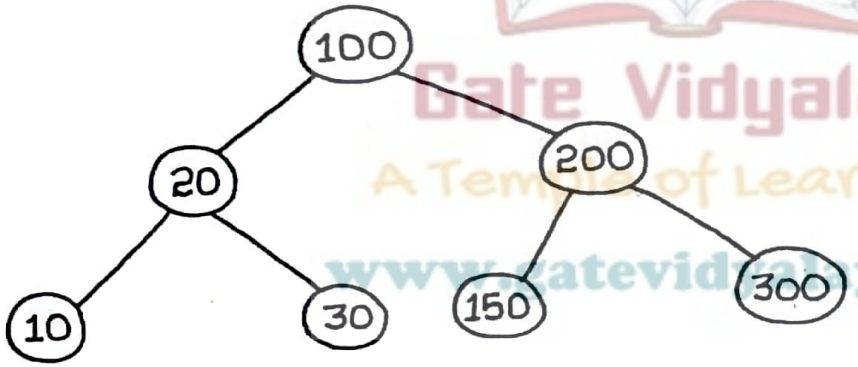
Example-



Traversal of a Binary Search Tree -

Traversal of a Binary Search Tree is exactly same as that of a Binary Tree.

Example -



- Preorder Traversal

100, 20, 10, 30, 200, 150, 300

- Inorder Traversal

10, 20, 30, 100, 150, 200, 300

- Postorder Traversal

10, 30, 20, 150, 300, 200, 100

Important Points -

- Inorder Traversal of a Binary Search Tree (BST) always yields all the nodes in increasing order.
- We can construct a Binary Search Tree with only preorder or postorder traversal because you can always get the inorder traversal by sorting the given traversal result in increasing order.

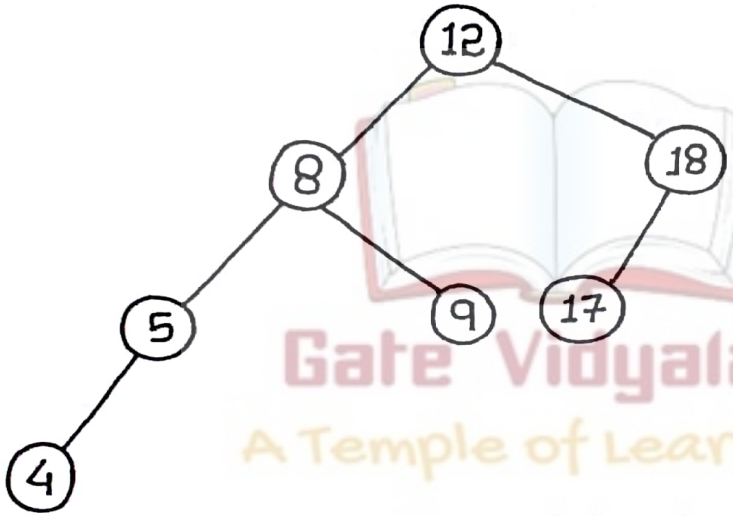
Definitions-

AVL trees are self-balancing Binary Search Trees where the difference between heights of left and right subtrees cannot be more than one for all nodes.

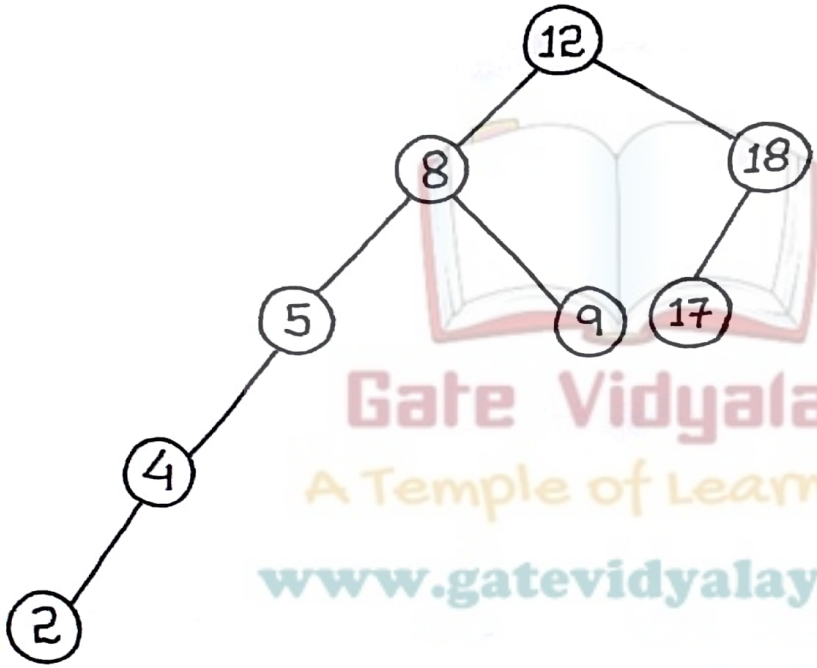
OR

AVL trees are Binary Search Trees in which the heights of the left and right subtrees of every node differs by at most one.

Example of AVL Tree -



Example of not an AVL Tree -



Balance Factor -

Balance Factor for any node is defined as -

> Balance Factor = Height of left subtree − Height of Right subtree

For a node to be balanced, the value of balance factor has to be either 0 or 1 or -1.

How balancing is done in AVL Trees?

In AVL Tree, after performing every operation like insertion and deletion, we check the balance factor of every node in the tree.

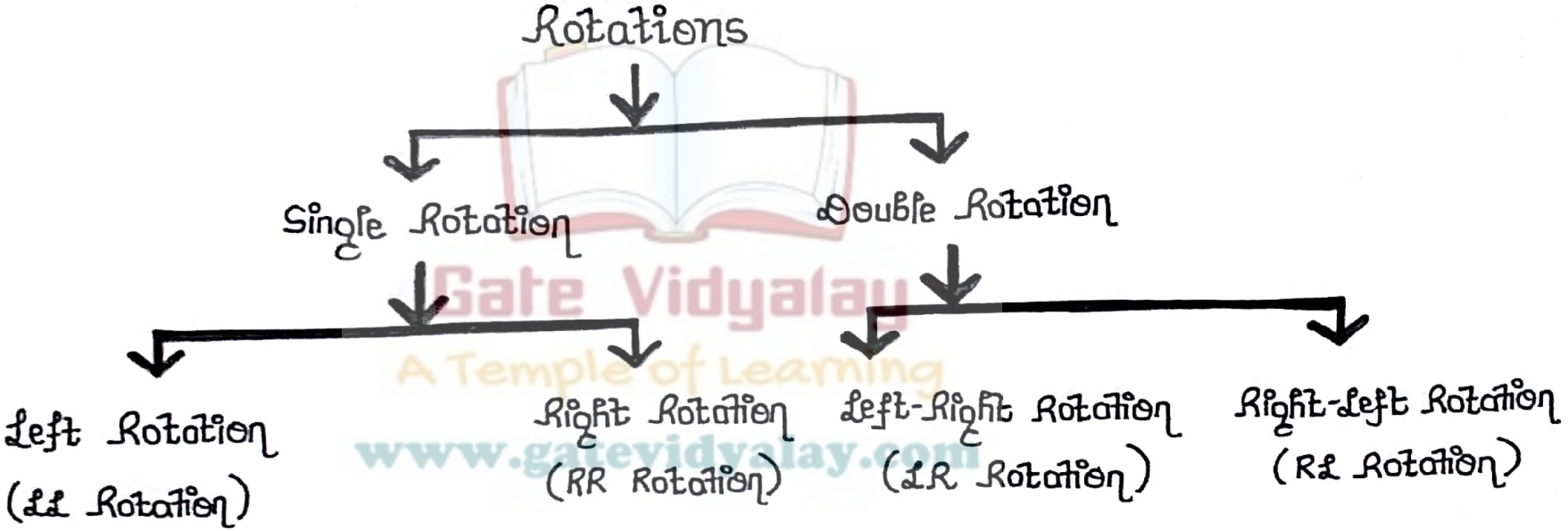
If every node satisfies the balance factor condition, then the operation is concluded otherwise we must make it balanced.

We use rotation operations to make the tree balanced when the tree becomes unbalanced due to any operation.

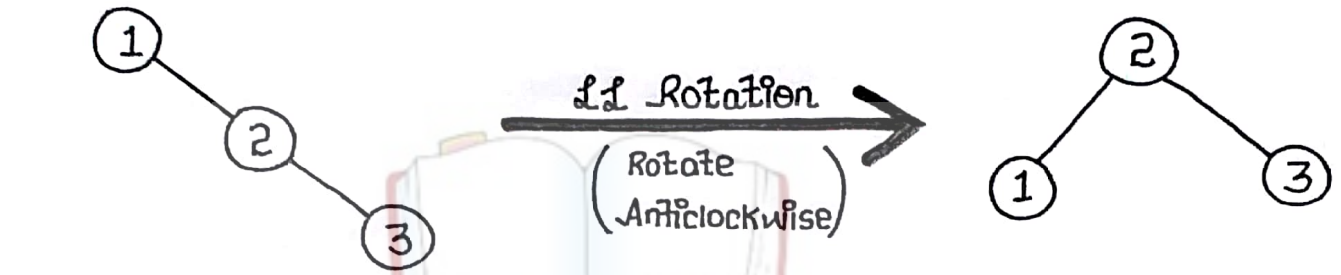
Thus,

> Rotation is the process of moving the nodes to make tree balanced.

Kinds of Rotations-



Case-I:

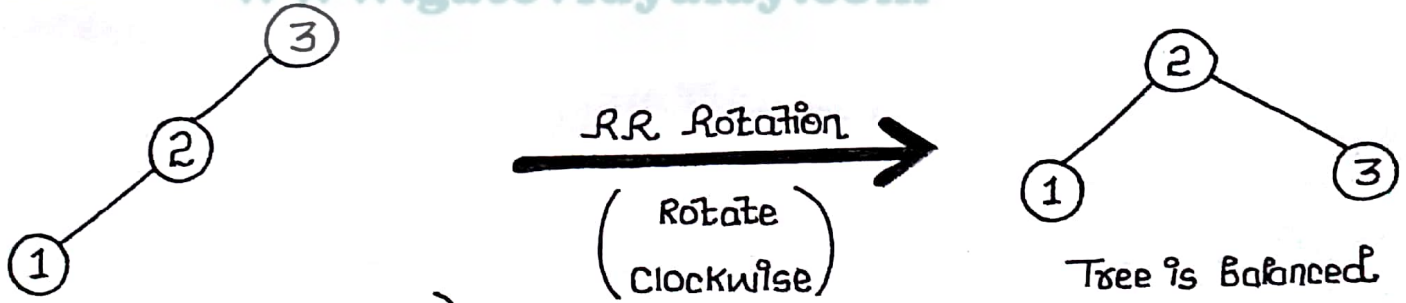


(Insertion order : 1,2,3)

Tree is Imbalanced

Tree is Balanced

Case-II:

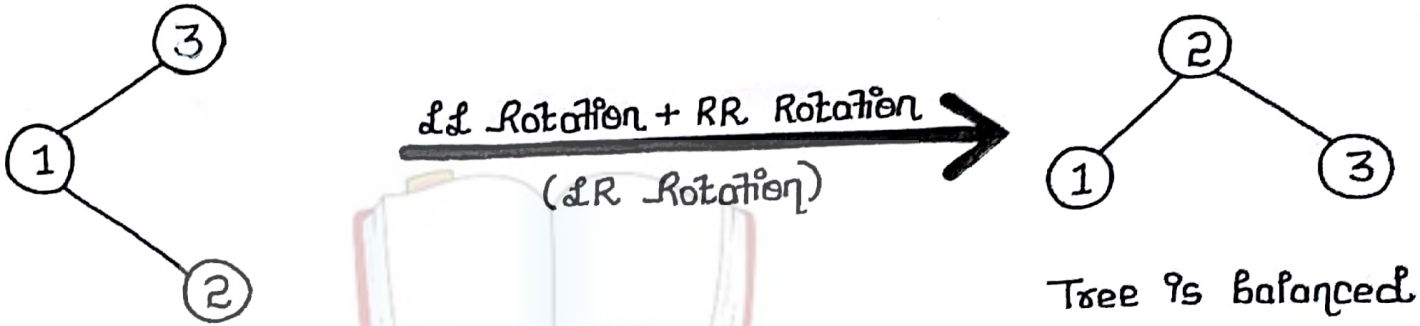


(Insertion order : 3,2,1)

Tree is Imbalanced

Tree is Balanced

Case-III:

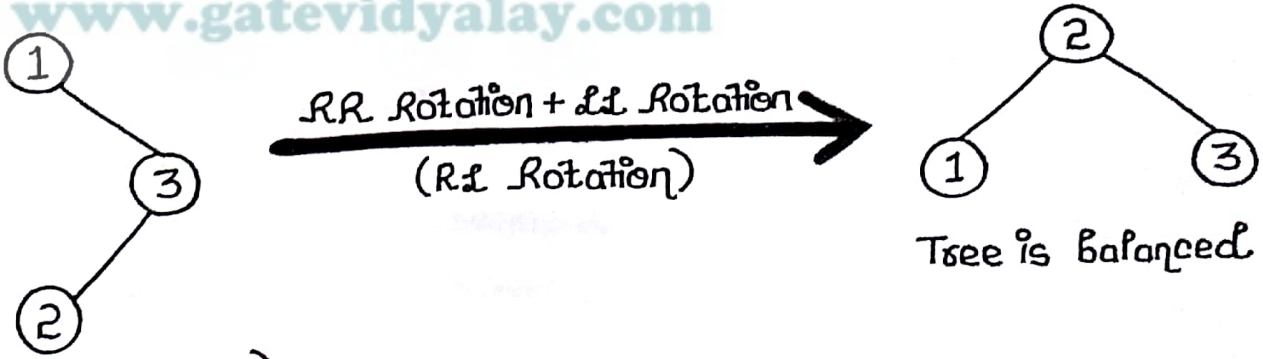


(Insertion Order : 3,1,2)

Tree is Imbalanced

Tree is balanced

Case-IV:



(Insertion Order : 1,3,2)

Tree is Imbalanced

Tree is balanced

Operations on AVL Trees-

The following operations are performed on AVL trees-

i) Search

ii) Insertion

iii) Deletion