Tree Data Structure Introduction

https://www.gatevidyalay.com/tag/trees-in-data-structure/

Tree data structure may be defined as-

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

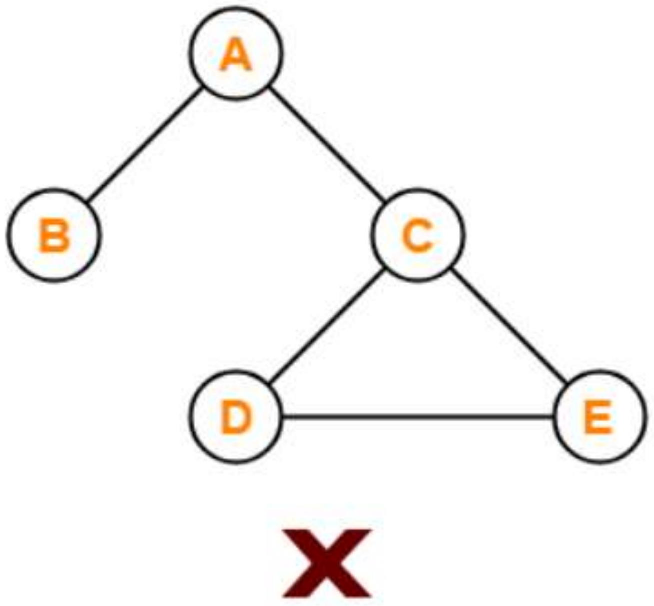
OR

A tree is a connected graph without any circuits.

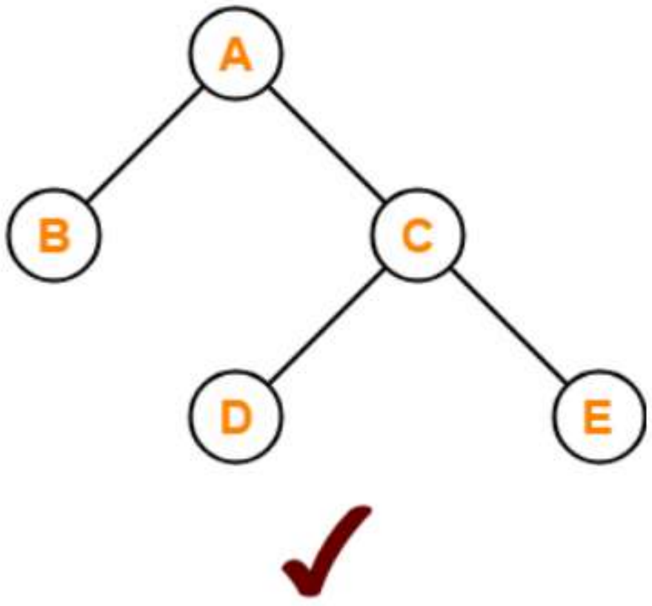
OR

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

Example-



This graph is not a Tree



This graph is a Tree

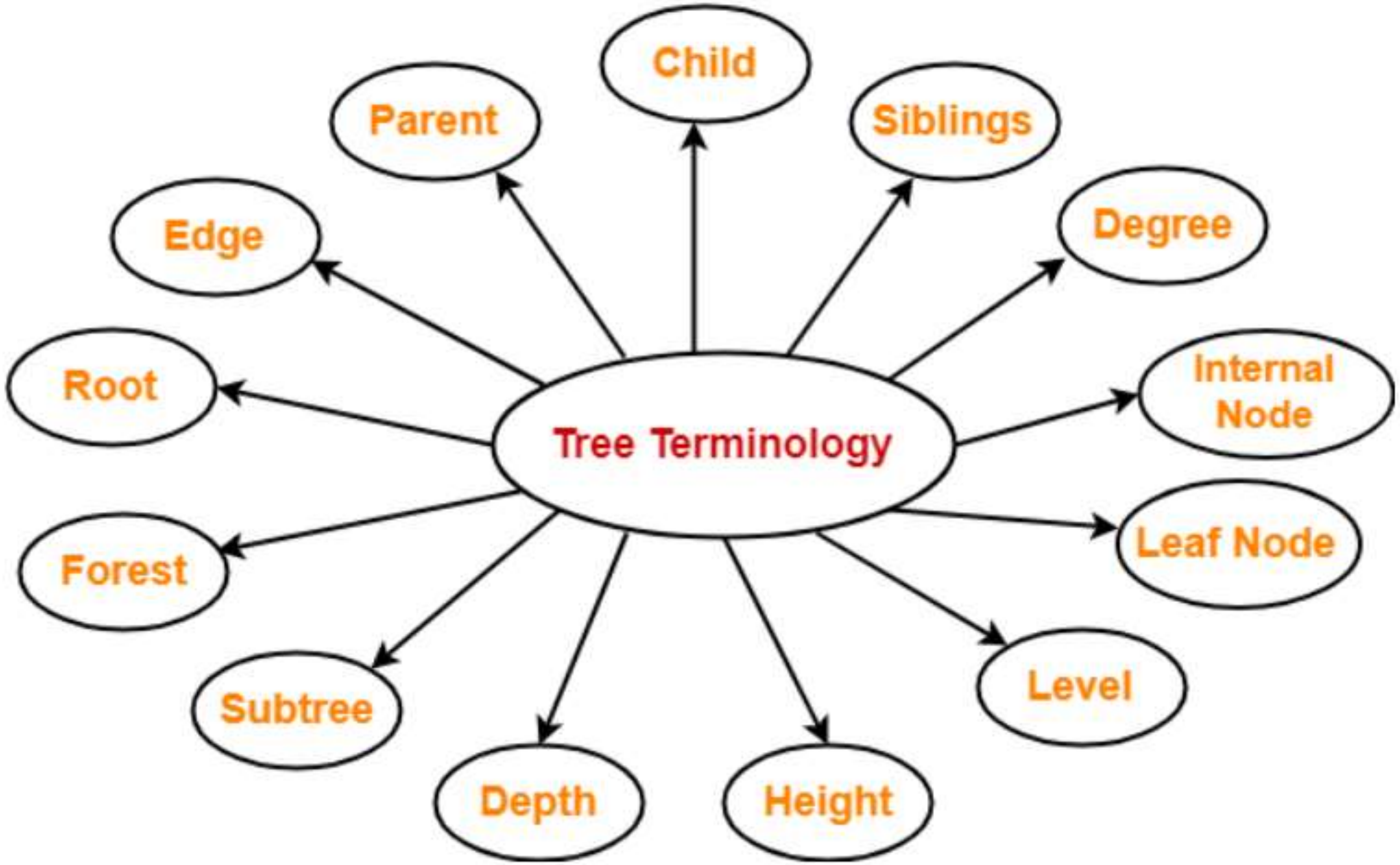
Properties-

The important properties of tree data structure are-

- There is one and only one path between every pair of vertices in a tree.
- A tree with n vertices has exactly (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.

Tree Terminology-

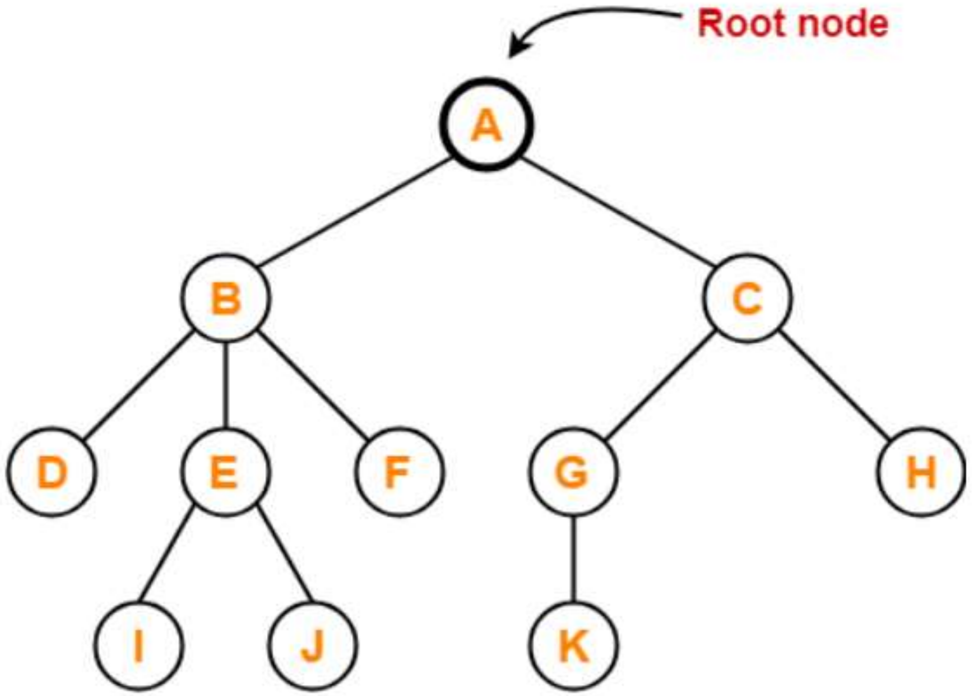
The important terms related to tree data structure are-



1. Root-

- The first node from where the tree originates is called as a **root node**.
- In any tree, there must be only one root node.
- We can never have multiple root nodes in a tree data structure.

Example-

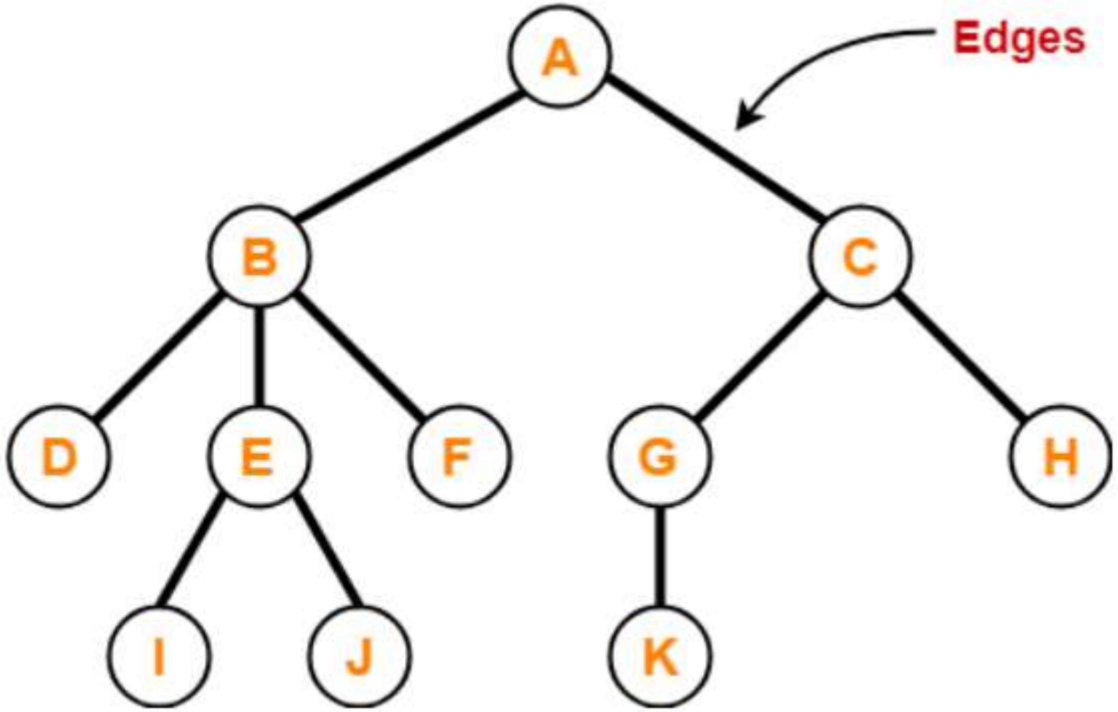


Here, node A is the only root node.

2. Edge-

- The connecting link between any two nodes is called as an **edge**.
- In a tree with n number of nodes, there are exactly (n-1) number of edges.

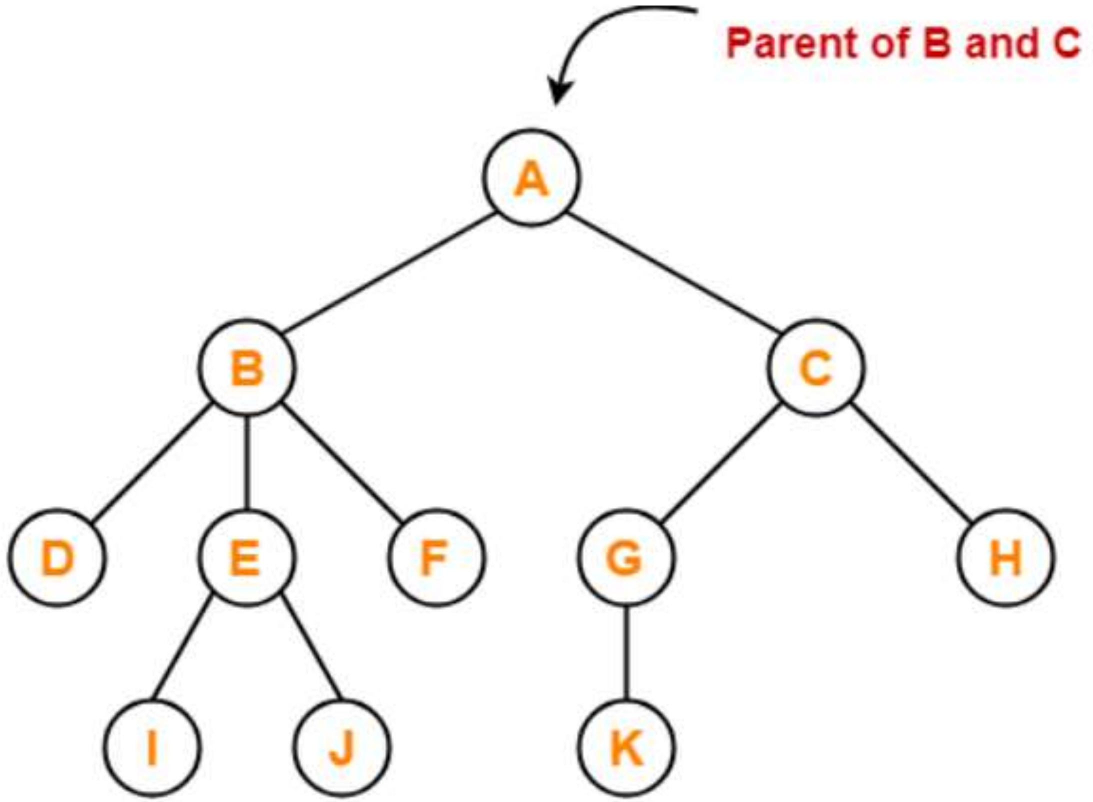
Example-



3. Parent-

- The node which has a branch from it to any other node is called as a **parent node**.
- In other words, the node which has one or more children is called as a parent node.
- In a tree, a parent node can have any number of child nodes.

Example-



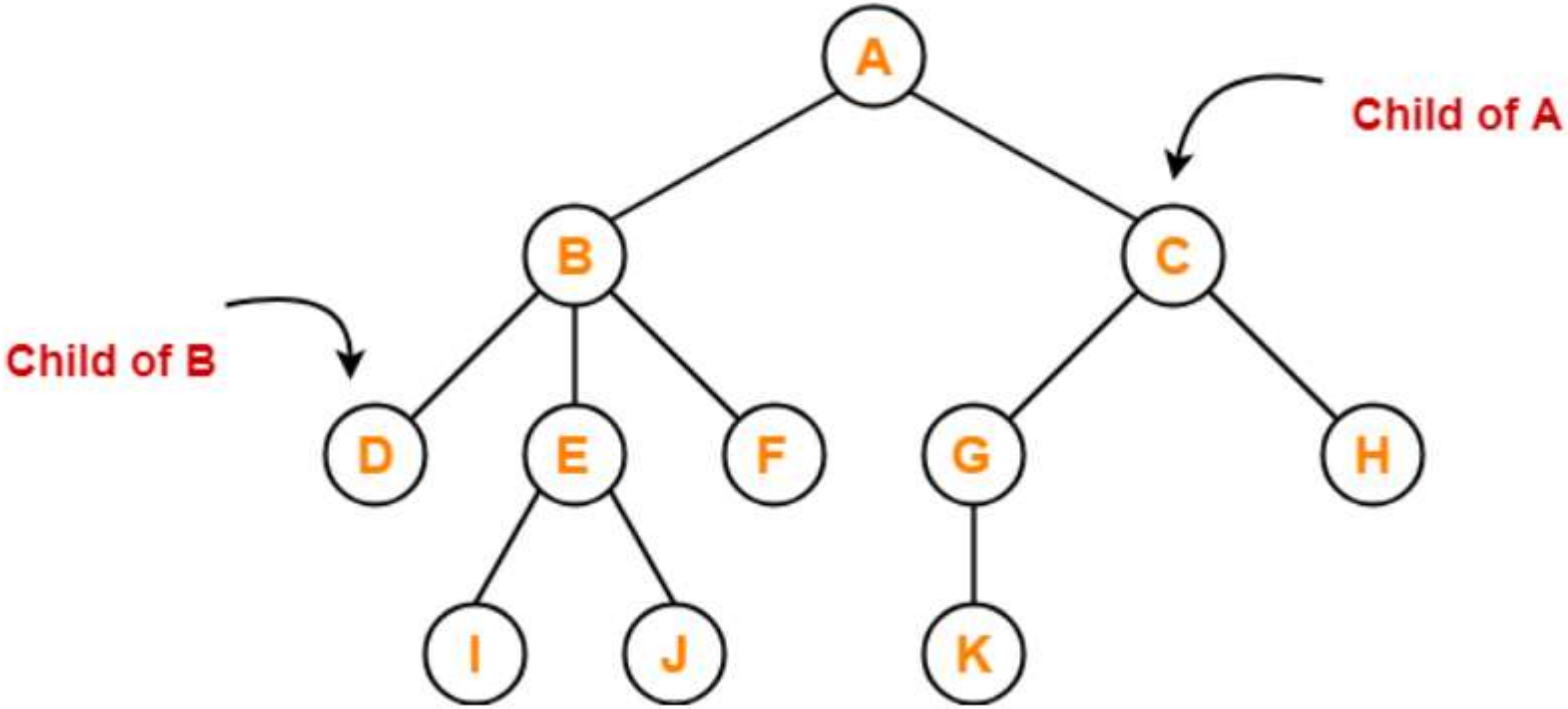
Here,

- Node A is the parent of nodes B and C
- Node B is the parent of nodes D, E and F
- Node C is the parent of nodes G and H
- Node E is the parent of nodes I and J
- Node G is the parent of node K

4. Child-

- The node which is a descendant of some node is called as a **child node**.
- All the nodes except root node are child nodes.

Example-



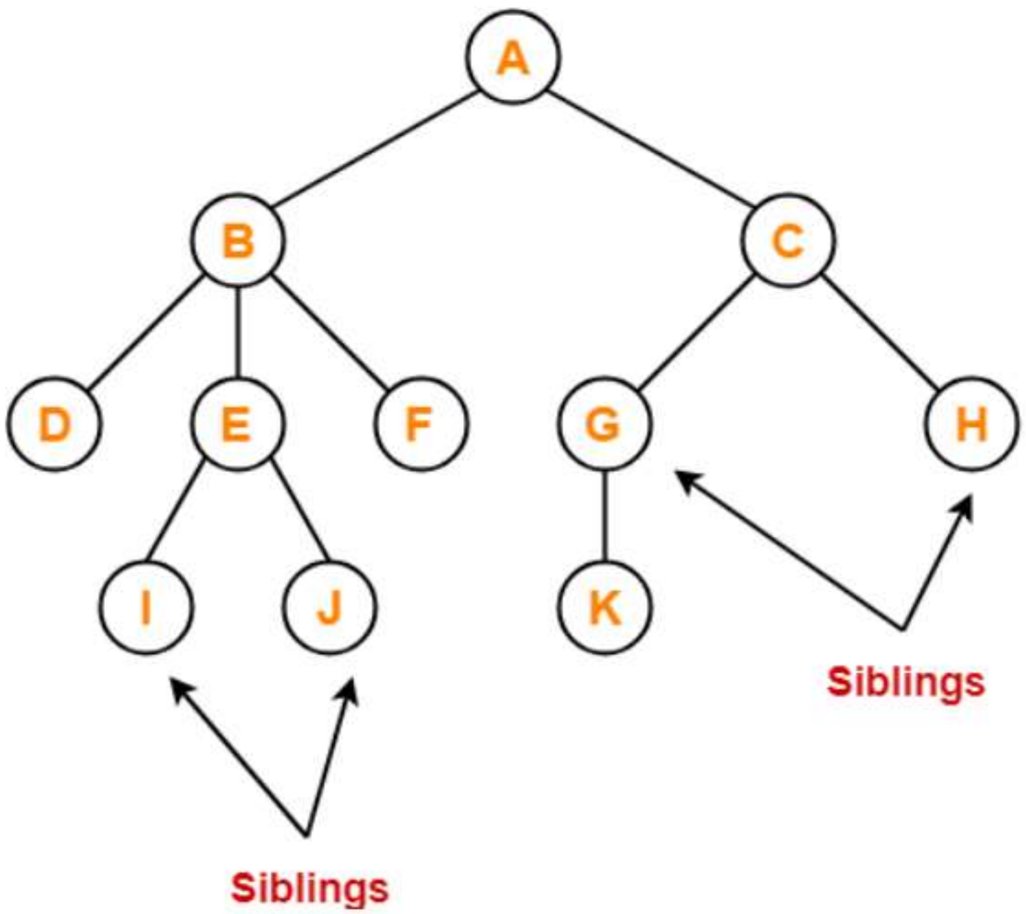
Here,

- Nodes B and C are the children of node A
- Nodes D, E and F are the children of node B
- Nodes G and H are the children of node C
- Nodes I and J are the children of node E
- Node K is the child of node G

5. Siblings-

- Nodes which belong to the same parent are called as **siblings**.
- In other words, nodes with the same parent are sibling nodes.

Example-



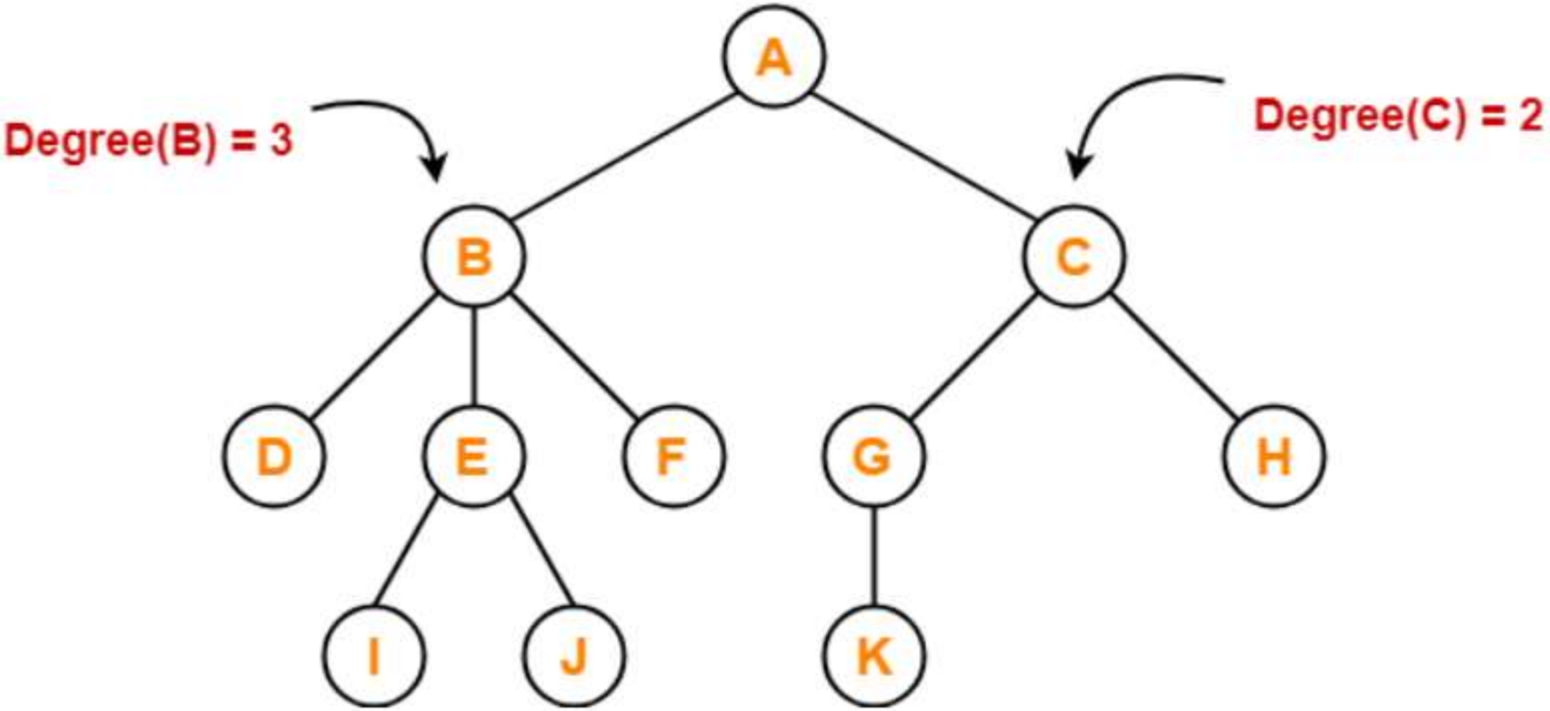
Here,

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

6. Degree-

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

Example-



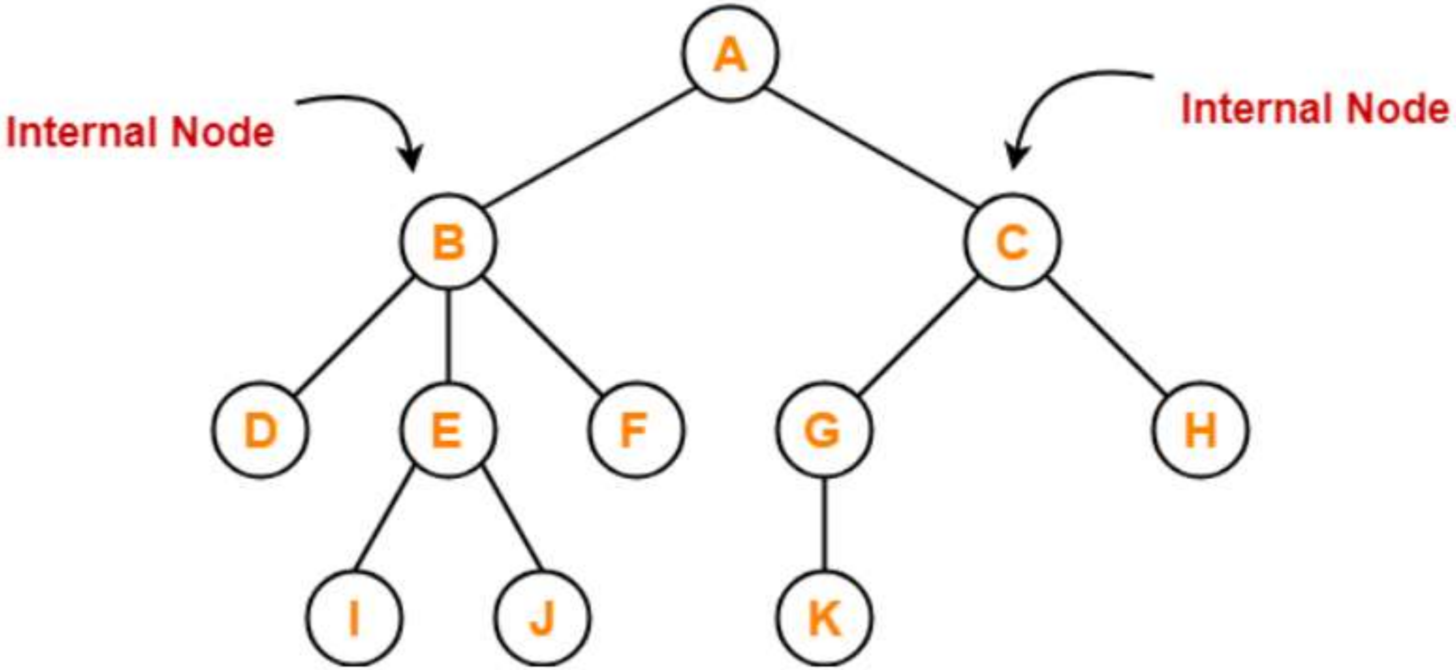
Here,

- Degree of node A = 2
- Degree of node B = 3
- Degree of node C = 2
- Degree of node D = 0
- Degree of node E = 2
- Degree of node F = 0
- Degree of node G = 1
- Degree of node H = 0
- Degree of node I = 0
- Degree of node J = 0
- Degree of node K = 0

7. Internal Node-

- The node which has at least one child is called as an **internal node**.
- Internal nodes are also called as **non-terminal nodes**.
- Every non-leaf node is an internal node.

Example-

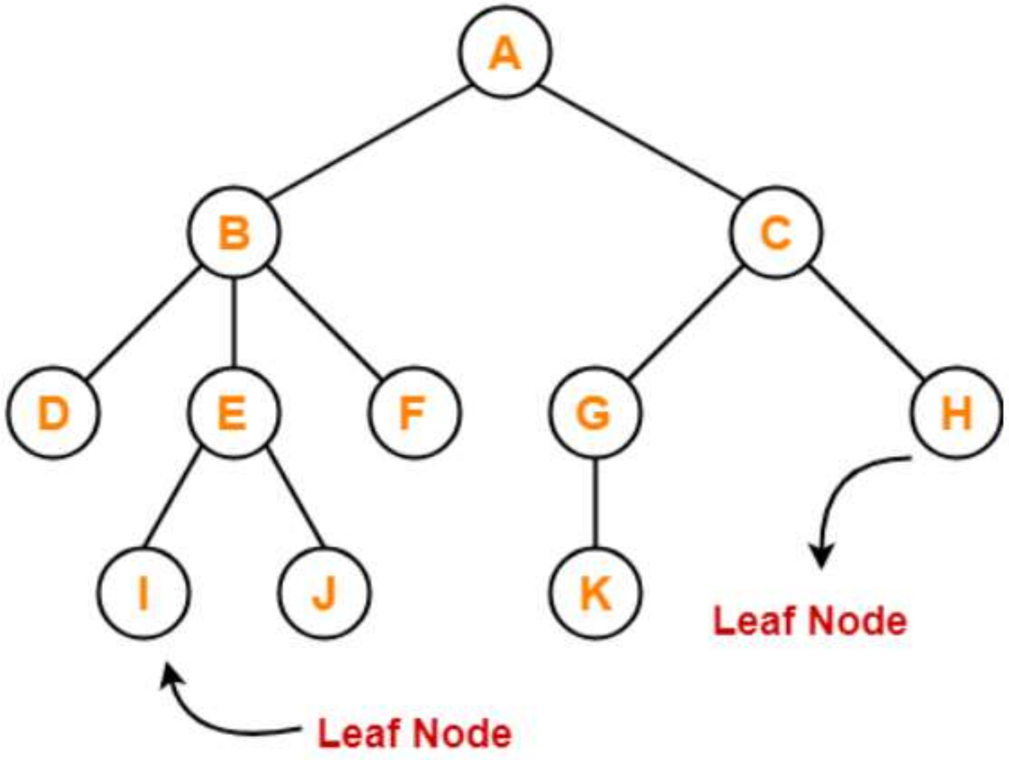


Here, nodes A, B, C, E and G are internal nodes.

8. Leaf Node-

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

Example-

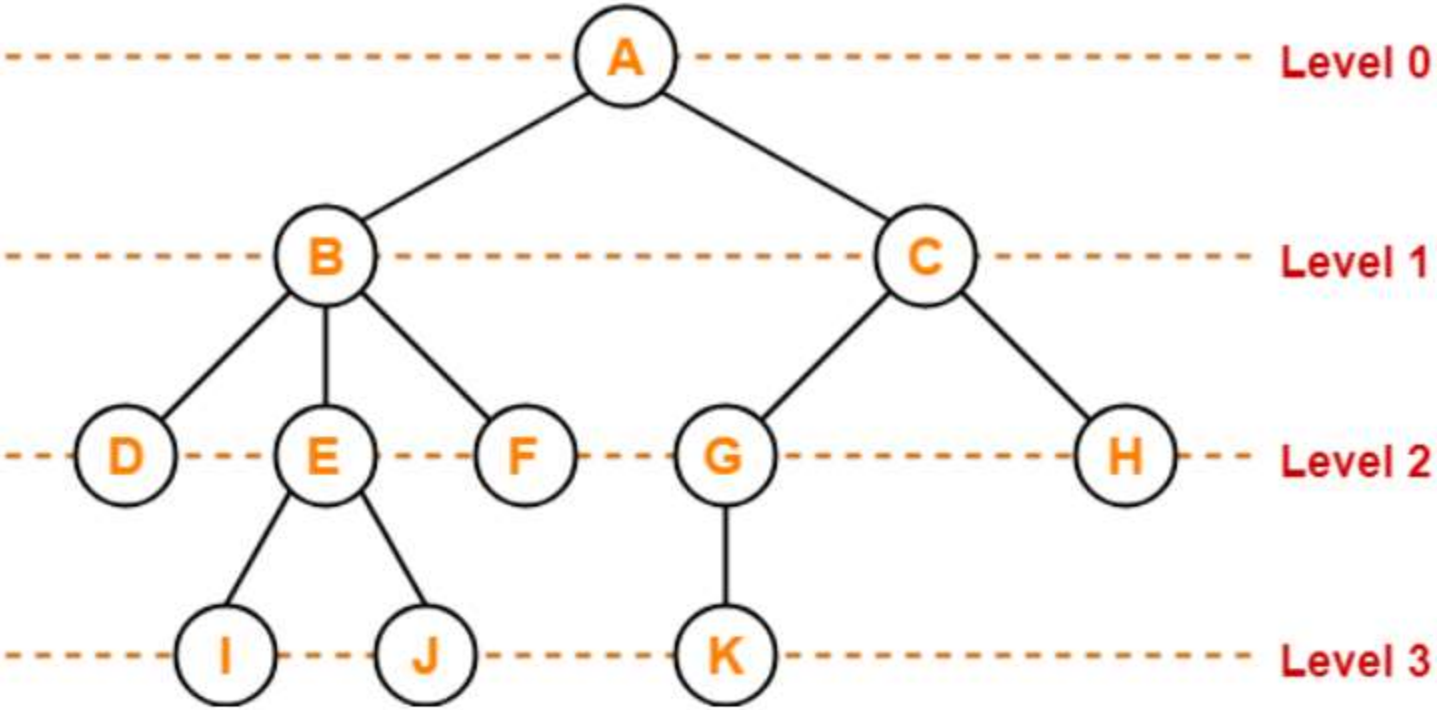


Here, nodes D, I, J, F, K and H are leaf nodes.

9. Level-

- In a tree, each step from top to bottom is called as **level of a tree**.
- The level count starts with 0 and increments by 1 at each level or step.

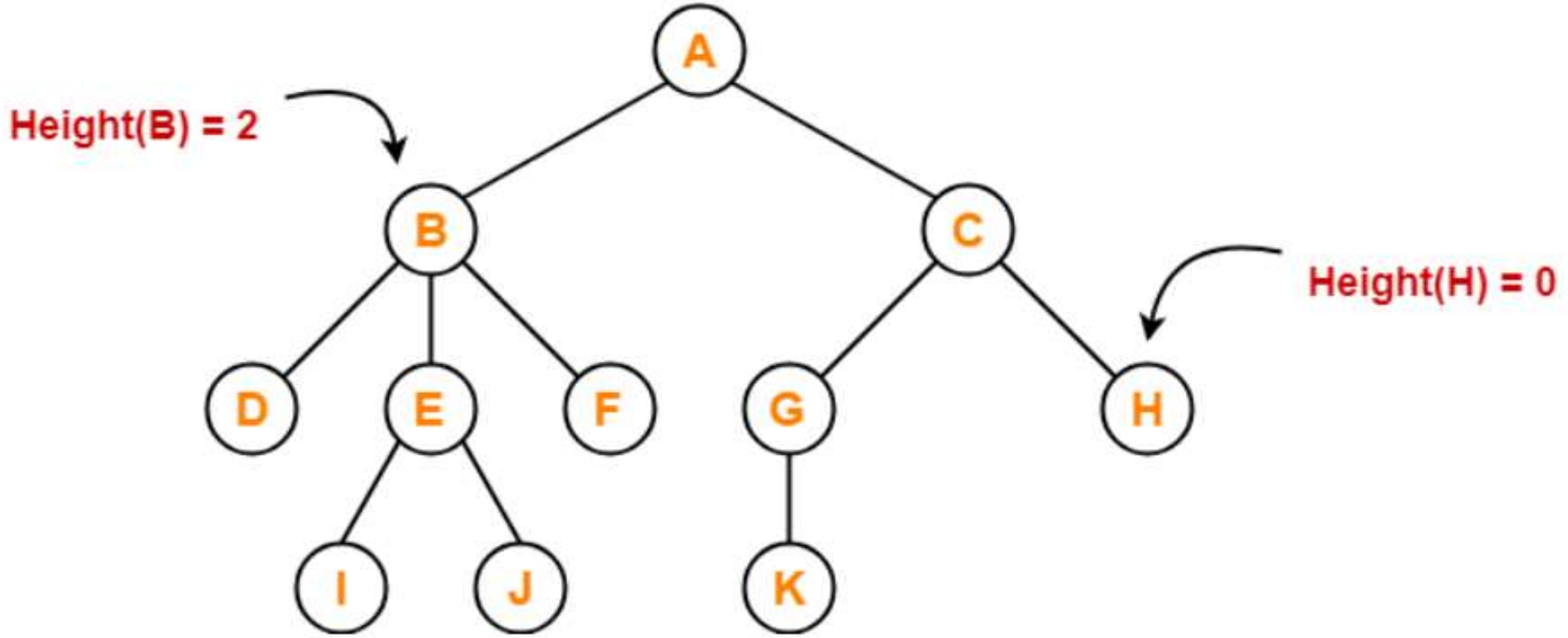
Example-



10. Height-

- Total number of edges that lies on the longest path from any leaf node to a particular node is called as **height of that node**.
- **Height of a tree** is the height of root node.
- Height of all leaf nodes = 0

Example-



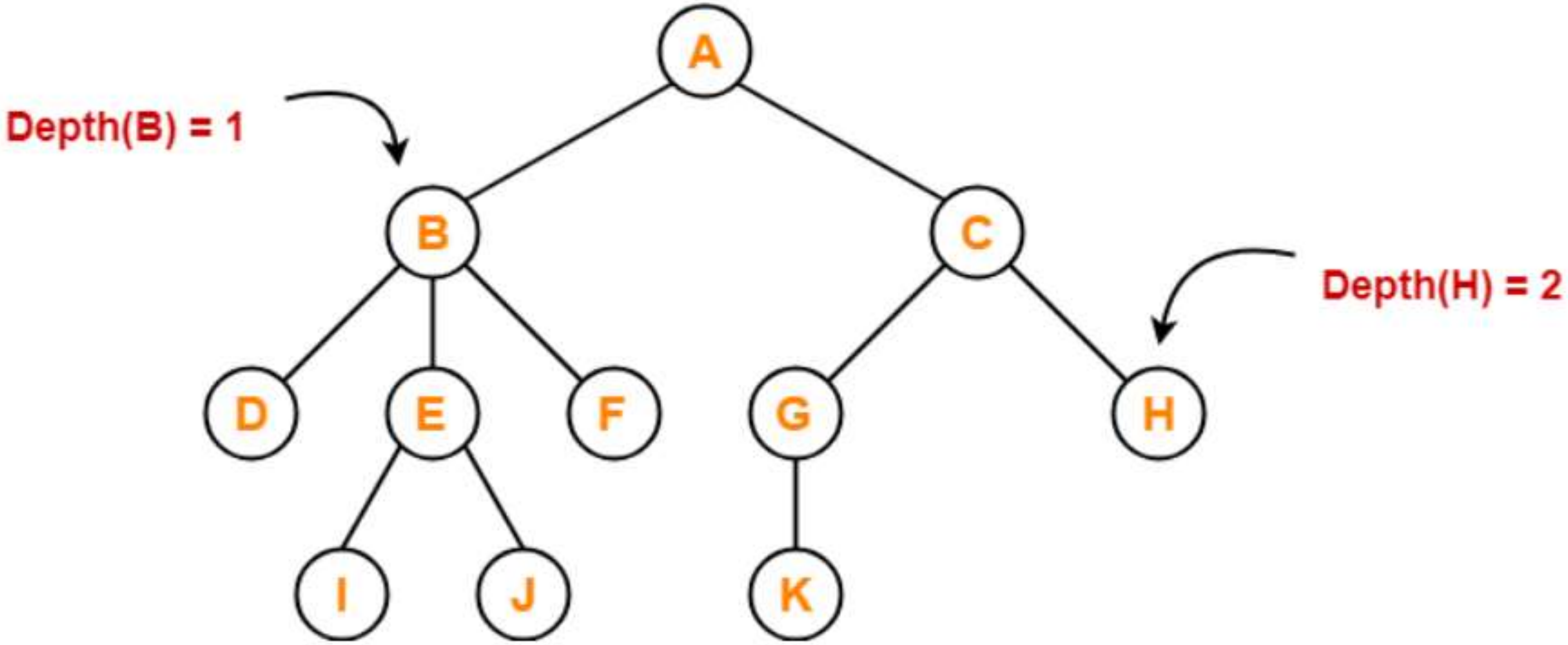
Here,

- Height of node A = 3
- Height of node B = 2
- Height of node C = 2
- Height of node D = 0
- Height of node E = 1
- Height of node F = 0
- Height of node G = 1
- Height of node H = 0
- Height of node I = 0
- Height of node J = 0
- Height of node K = 0

11. Depth-

- Total number of edges from root node to a particular node is called as **depth of that node**.
- **Depth of a tree** is the total number of edges from root node to a leaf node in the longest path.
- Depth of the root node = 0
- The terms "level" and "depth" are used interchangeably.

Example-



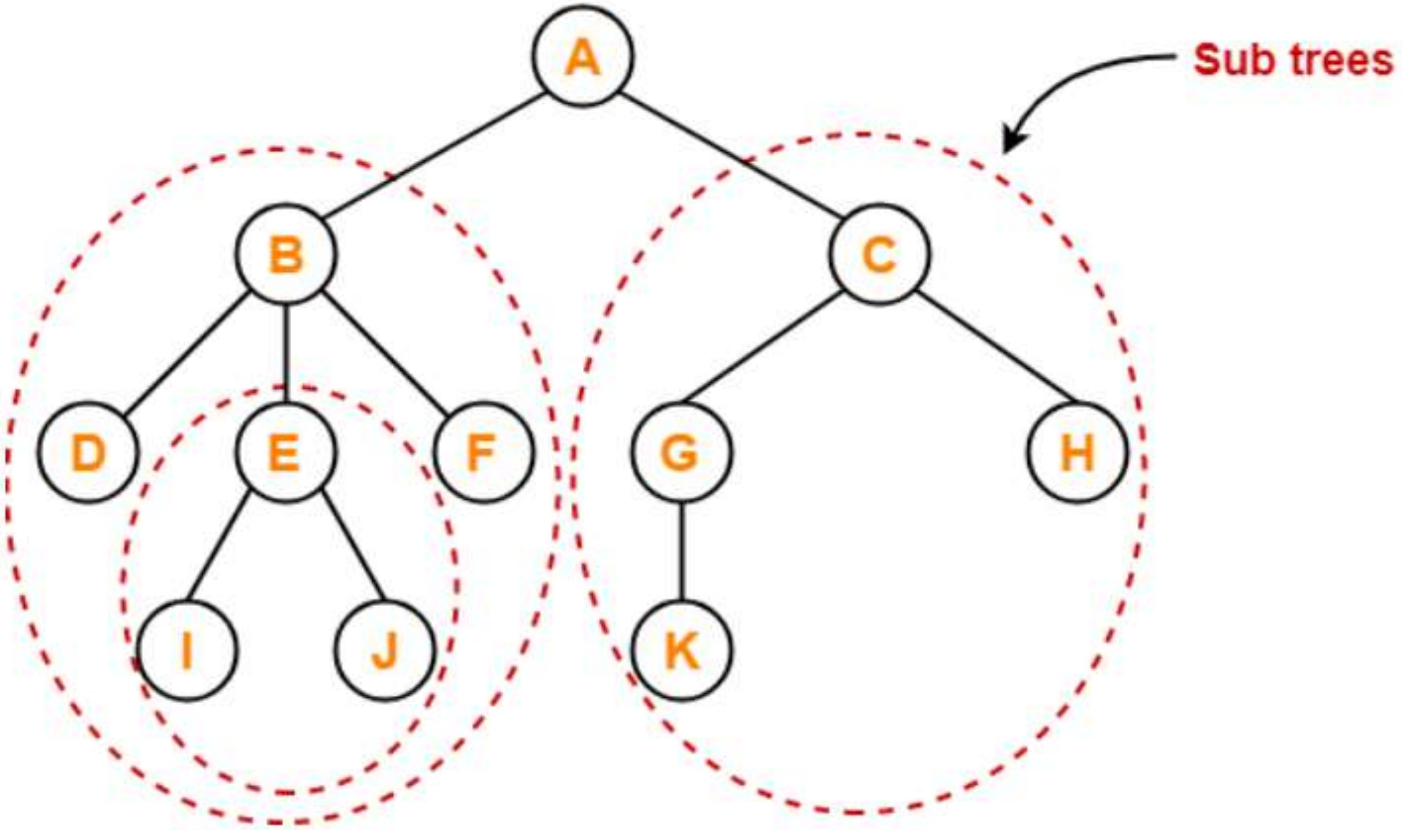
Here,

- Depth of node A = 0
- Depth of node B = 1
- Depth of node C = 1
- Depth of node D = 2
- Depth of node E = 2
- Depth of node F = 2
- Depth of node G = 2
- Depth of node H = 2
- Depth of node I = 3
- Depth of node J = 3
- Depth of node K = 3

12. Subtree-

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

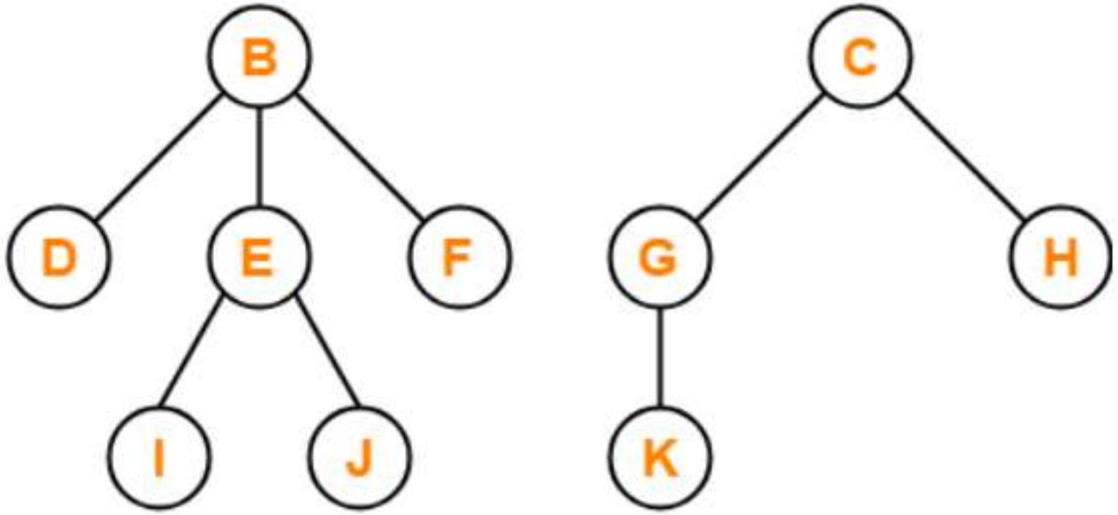
Example-



13. Forest-

A forest is a set of disjoint trees.

Example-



Forest

Binary Tree Data Structure

https://www.gatevidyalay.com/binary-tree-types-of-trees-in-data-structure/

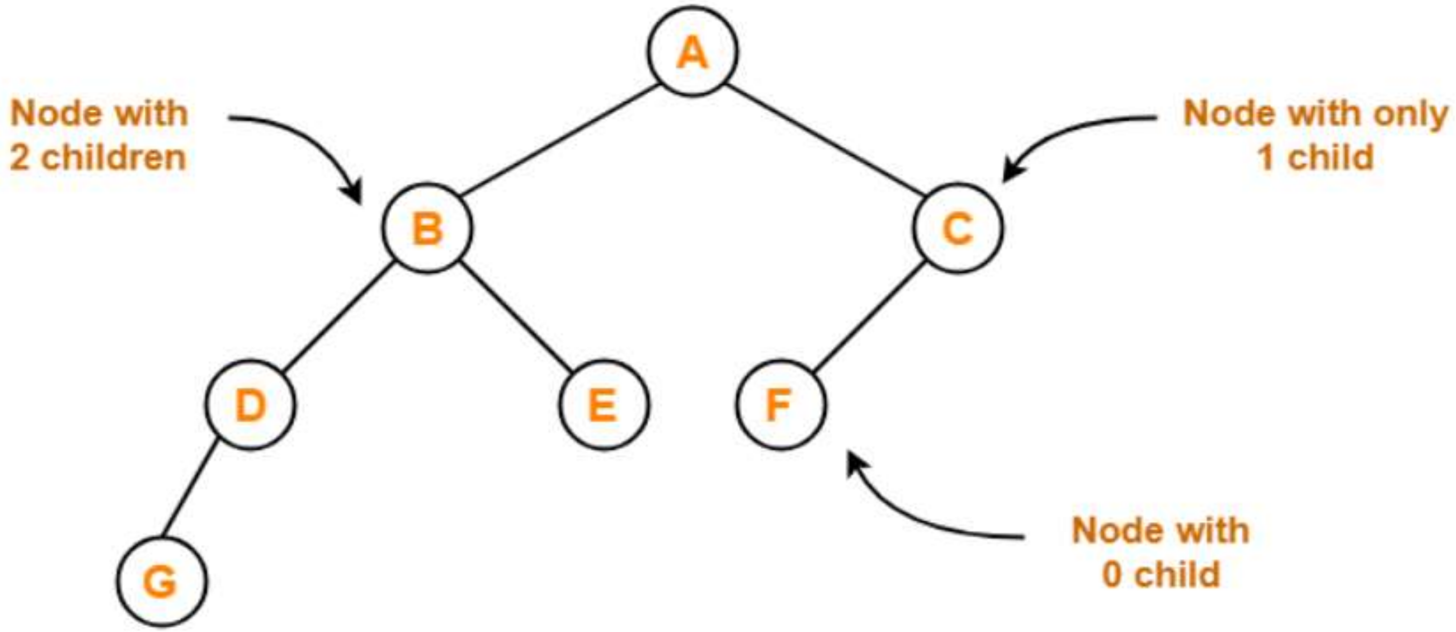
Binary Tree-

Binary tree is a special tree data structure in which each node can have at most 2 children.

Thus, in a binary tree,

Each node has either 0 child or 1 child or 2 children.

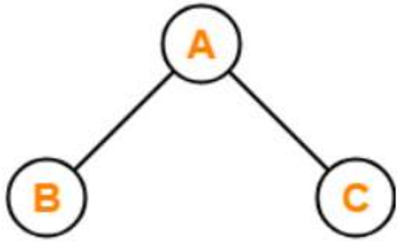
Example- This is a labeled Binary Tree



Binary Tree Example

Labeled Binary Tree-

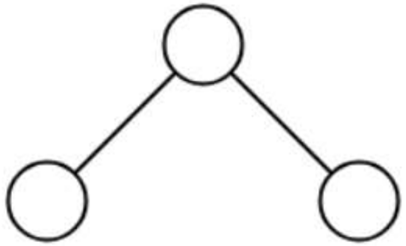
A binary tree is labeled if all its nodes are assigned a label.



Labeled Binary Tree

Unlabeled Binary Tree-

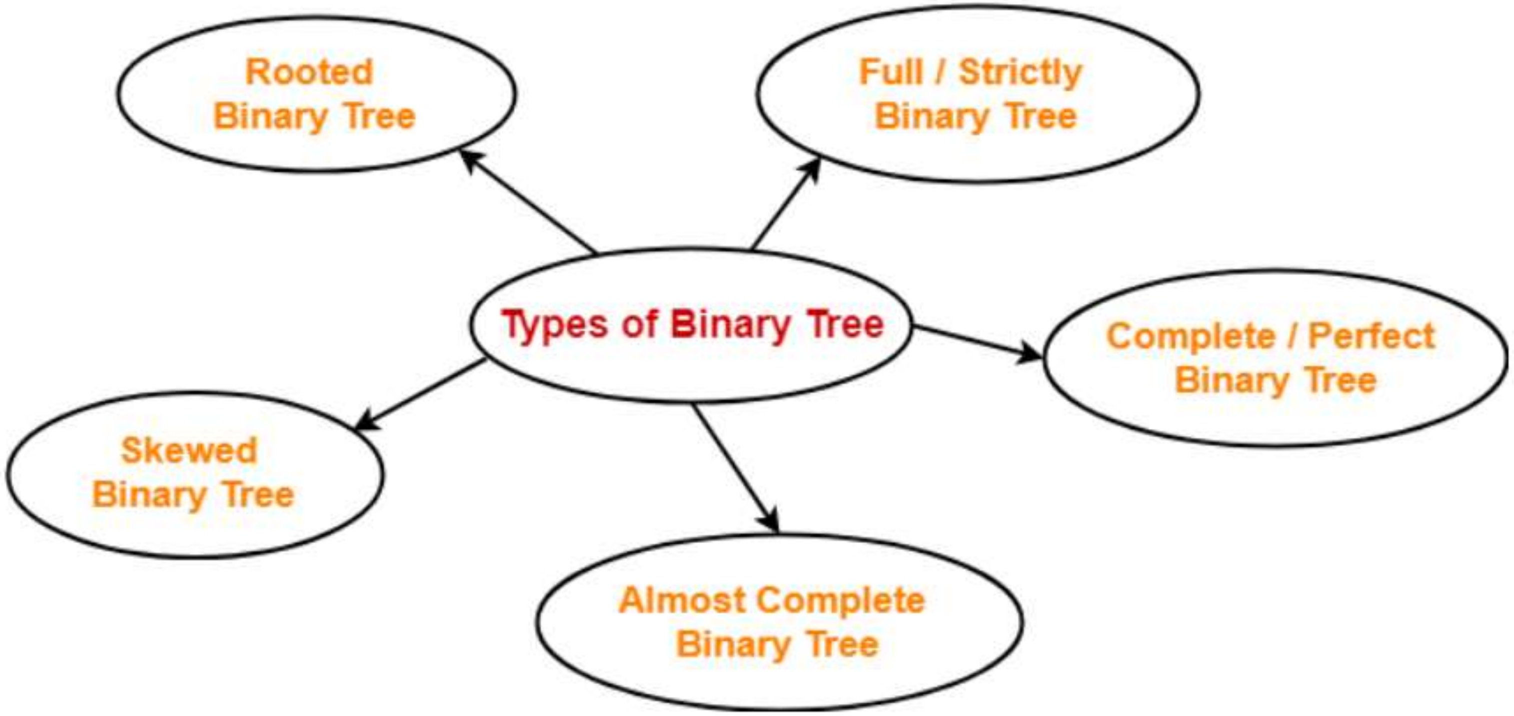
A binary tree is unlabeled if its nodes are not assigned any label.



Unlabeled Binary Tree

Types of Binary Trees-

Binary trees can be of the following types-



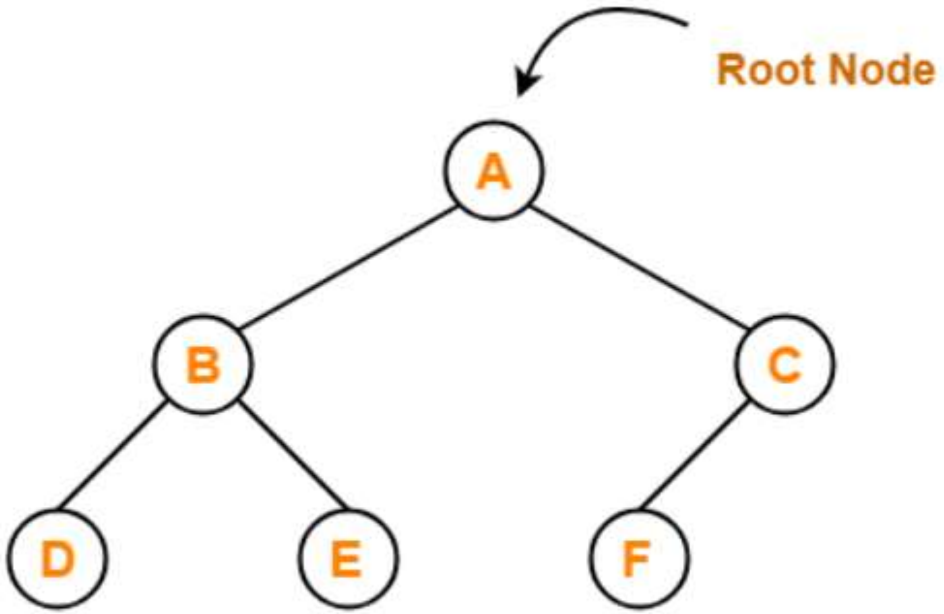
1. Rooted Binary Tree
2. Full / Strictly Binary Tree
3. Complete / Perfect Binary Tree
4. Almost Complete Binary Tree
5. Skewed Binary Tree

1. Rooted Binary Tree-

A **rooted binary tree** is a binary tree that satisfies the following 2 properties-

- It has a root node.
- Each node has at most 2 children.

Example-

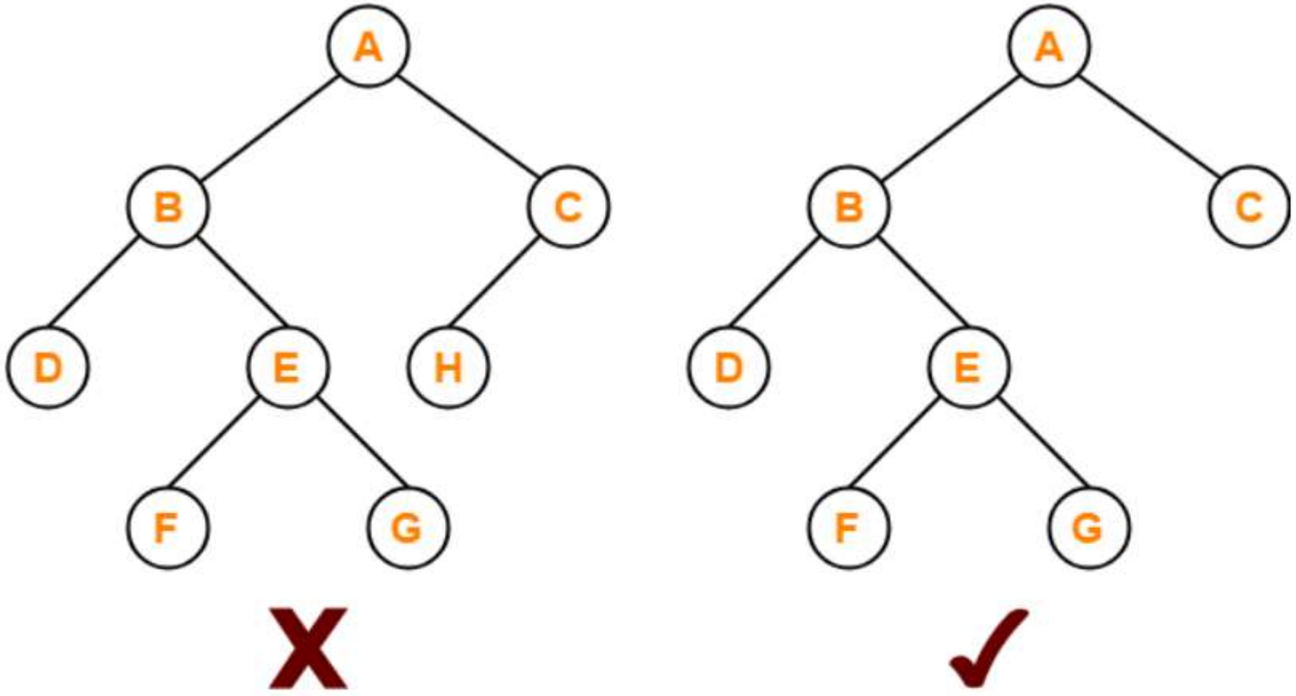


Rooted Binary Tree

2. Full / Strictly Binary Tree-

- A binary tree in which every node has either 0 or 2 children is called as a **Full binary tree**.
- Full binary tree is also called as **Strictly binary tree**.

Example-



Here,

- First binary tree is not a full binary tree.
- This is because node C has only 1 child.

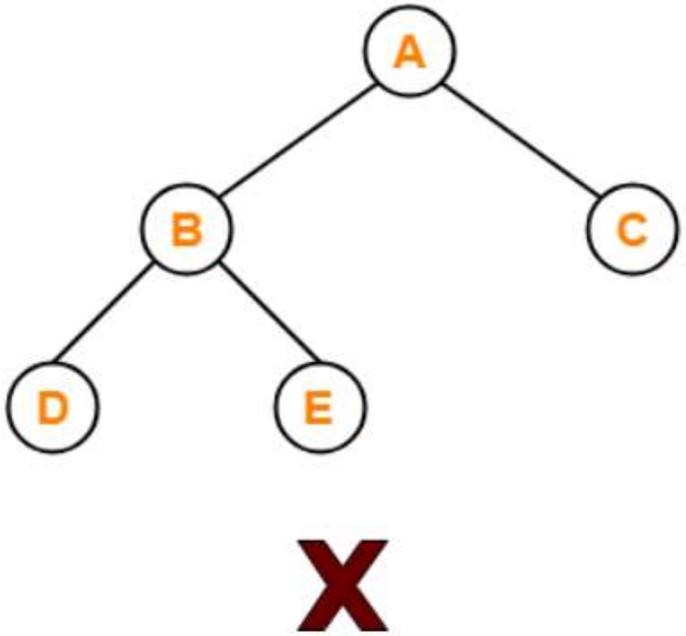
3. Complete / Perfect Binary Tree-

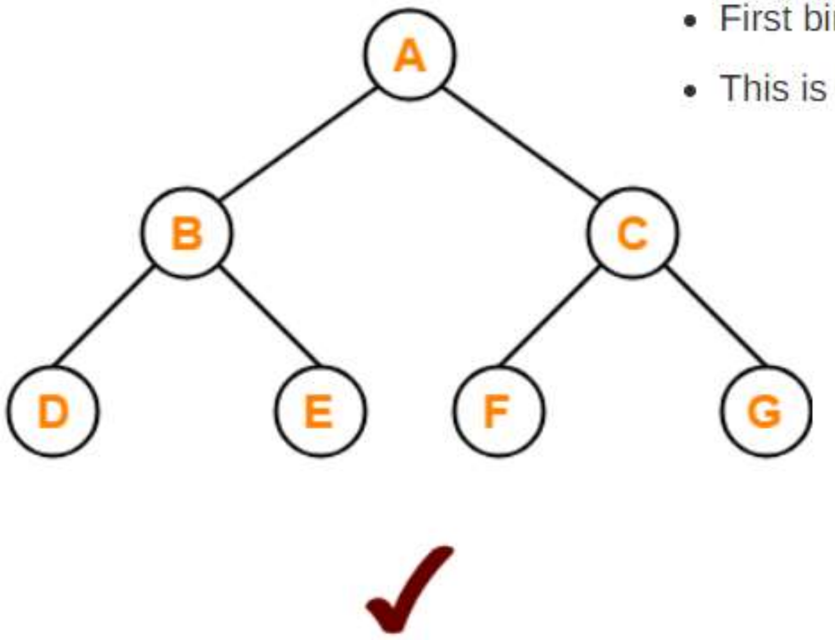
A **complete binary tree** is a binary tree that satisfies the following 2 properties-

- Every internal node has exactly 2 children.
- All the leaf nodes are at the same level.

Complete binary tree is also called as **Perfect binary tree**.

Example-





Here,

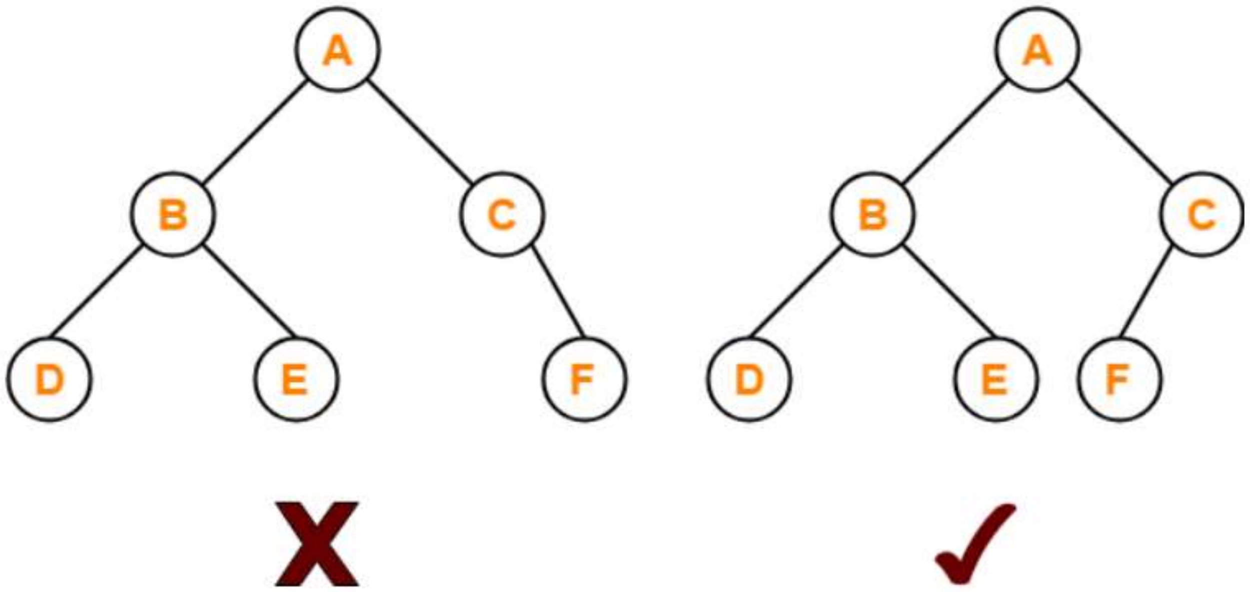
- First binary tree is not a complete binary tree.
- This is because all the leaf nodes are not at the same level.

4. Almost Complete Binary Tree-

An **almost complete binary tree** is a binary tree that satisfies the following 2 properties-

- All the levels are completely filled except possibly the last level.
- The last level must be strictly filled from left to right.

Example-



Here,

- First binary tree is not an almost complete binary tree.
- This is because the last level is not filled from left to right.

5. Skewed Binary Tree-

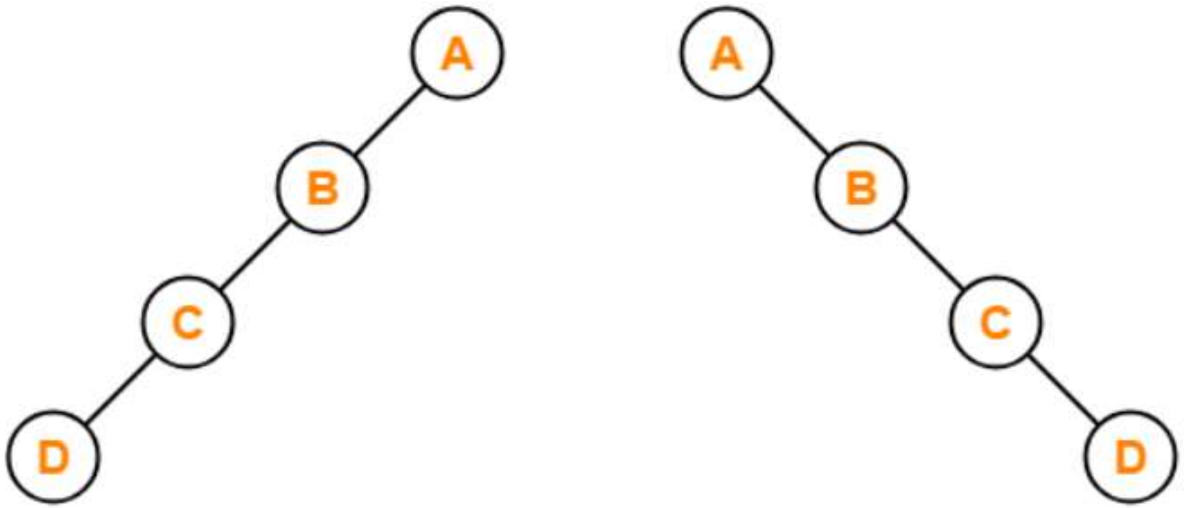
A **skewed binary tree** is a binary tree that satisfies the following 2 properties-

- All the nodes except one node has one and only one child.
- The remaining node has no child.

OR

A **skewed binary tree** is a binary tree of n nodes such that its depth is (n-1).

Example-



Left Skewed Binary Tree   Right Skewed Binary Tree

Binary Tree Properties

https://www.gatevidyalay.com/binary-tree-properties-important-formulas/

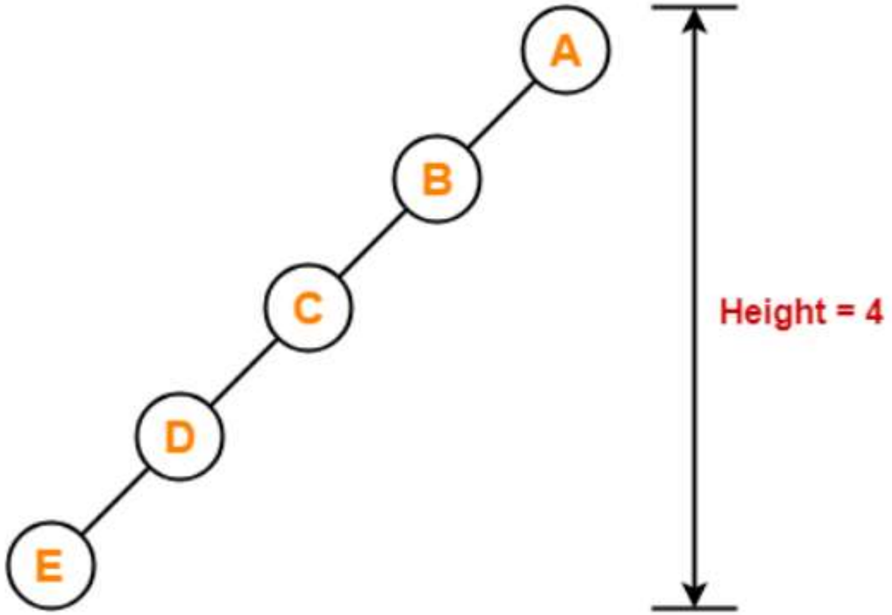
Property-01:

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 4 + 1 = 5 nodes.



Property-02:

Maximum number of nodes in a binary tree of height H

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

Example-

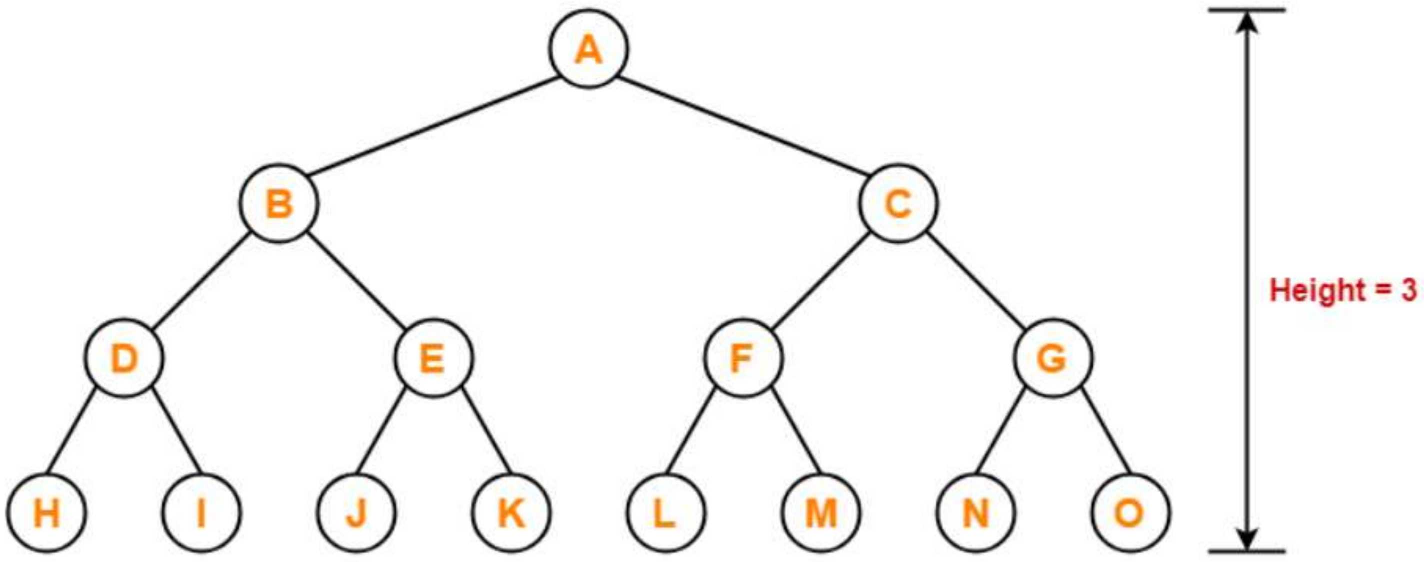
Maximum number of nodes in a binary tree of height 3

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

$= 16 - 1$

= 15 nodes

Thus, in a binary tree of height = 3, maximum number of nodes that can be inserted = 15.



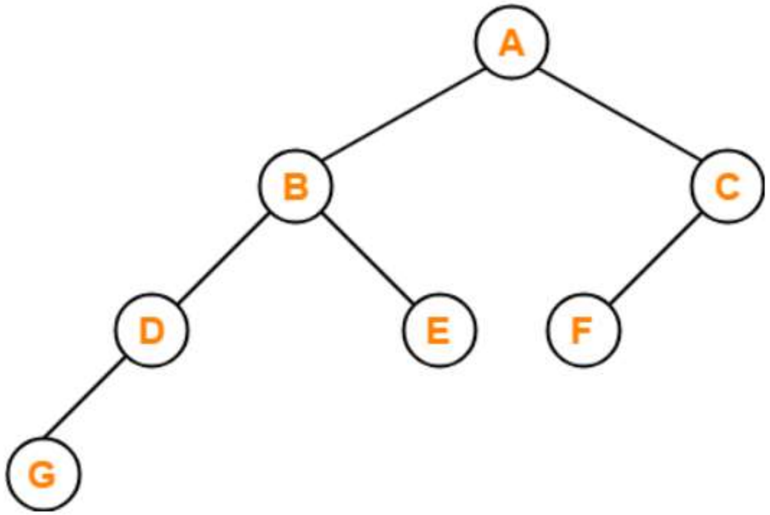
We can not insert more number of nodes in this binary tree.

Property-03:

Total Number of leaf nodes in a Binary Tree **= Total Number of nodes with 2 children + 1**

Example-

Consider the following binary tree-



Here,

- Number of leaf nodes = 3
- Number of nodes with 2 children = 2

Clearly, number of leaf nodes is one greater than number of nodes with 2 children.

This verifies the above relation.

NOTE

It is interesting to note that-

Number of leaf nodes in any binary tree depends only on the number of nodes with 2 children.

Property-04:

Maximum number of nodes at any level 'L' in a binary tree

$= 2^L$

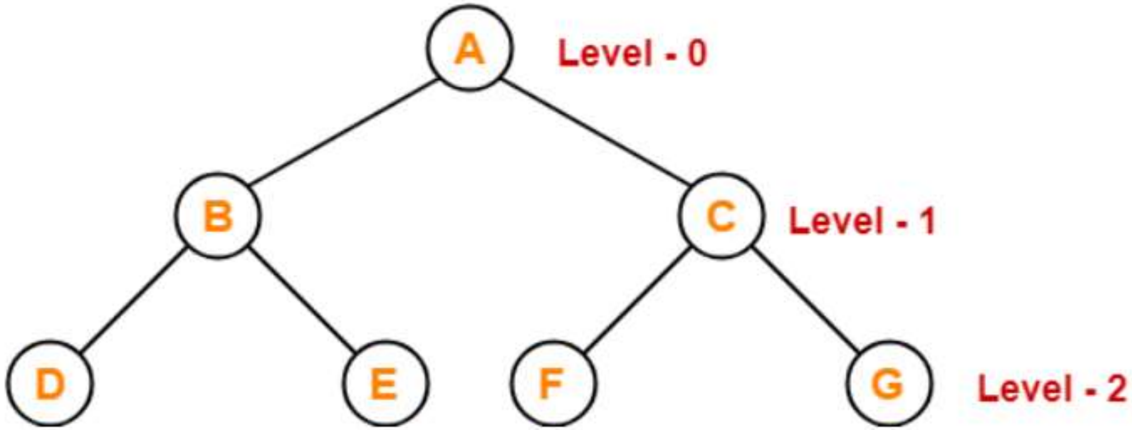
Example-

Maximum number of nodes at level-2 in a binary tree

$= 2^2$

$= 4$

Thus, in a binary tree, maximum number of nodes that can be present at level-2 = 4.



Binary Tree Traversal

https://www.gatevidyalay.com/tree-traversal-binary-tree-traversal/

Tree Traversal-

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

There are 2 main Tree Traversal techniques as,

1) Depth First Traversal
2) Breadth First Traversal

Depth First Traversal-

Following three traversal techniques fall under Depth First Traversal-

1. Preorder Traversal
2. Inorder Traversal
3. Postorder Traversal

1. Preorder Traversal-

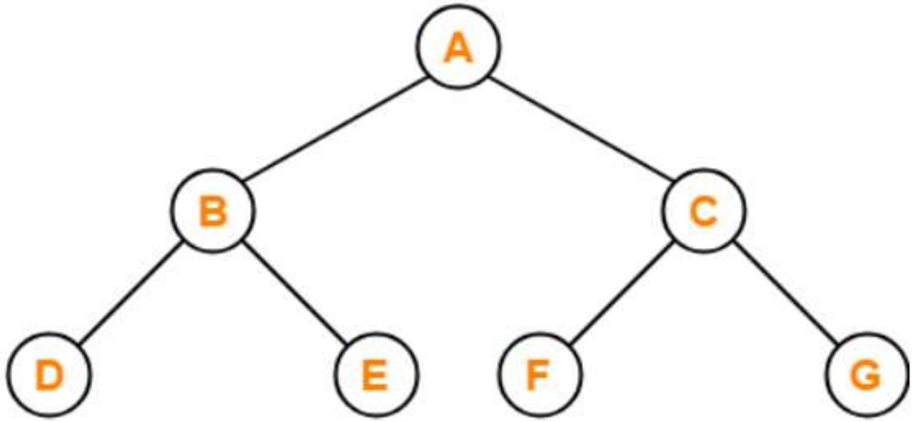
Algorithm-

1. Visit the root
2. Traverse the left sub tree i.e. call Preorder (left sub tree)
3. Traverse the right sub tree i.e. call Preorder (right sub tree)

Root → Left → Right

Example-

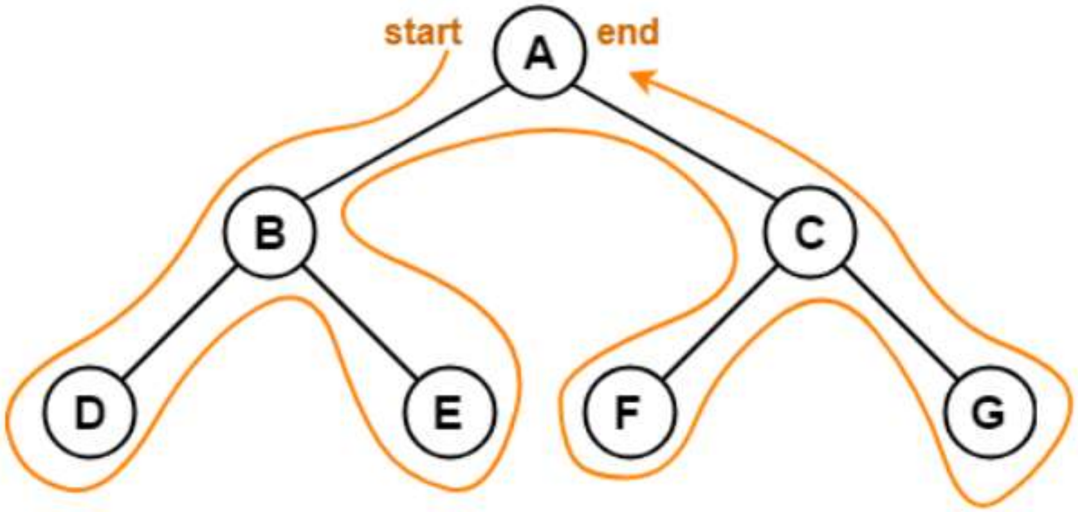
Consider the following example-



Preorder Traversal : A , B , D , E , C , F , G

Preorder Traversal Shortcut

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



Preorder Traversal : A , B , D , E , C , F , G

Applications-

- Preorder traversal is used to get prefix expression of an expression tree.
- Preorder traversal is used to create a copy of the tree.

2. Inorder Traversal-

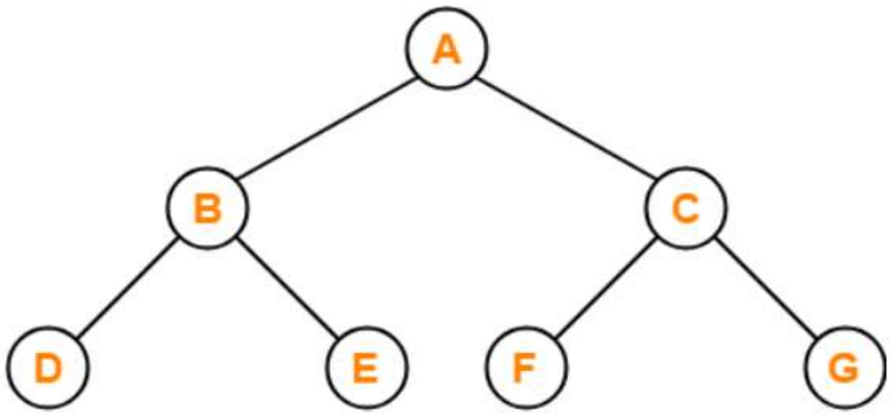
Algorithm-

1. Traverse the left sub tree i.e. call Inorder (left sub tree)
2. Visit the root
3. Traverse the right sub tree i.e. call Inorder (right sub tree)

Left → Root → Right

Example-

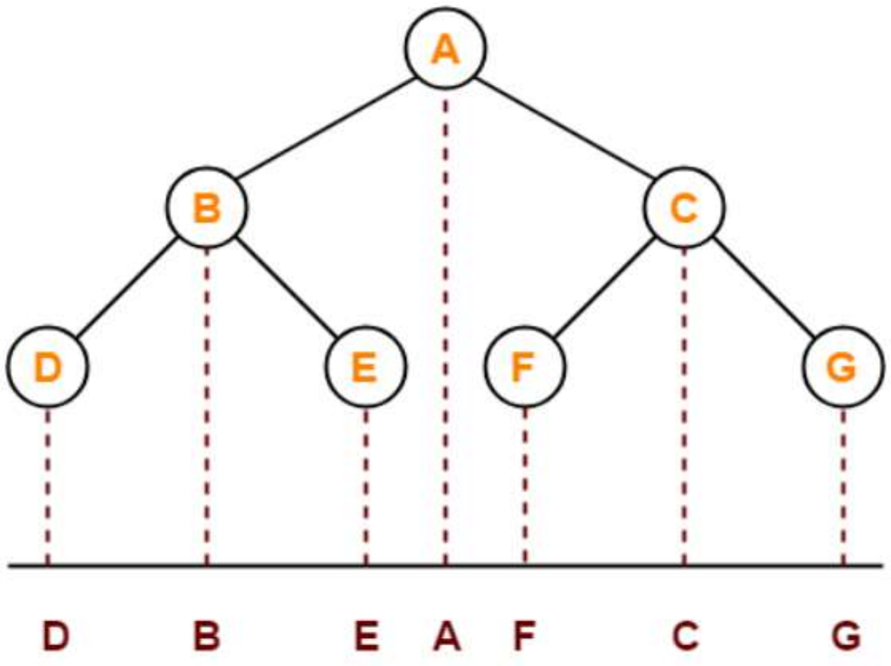
Consider the following example-



Inorder Traversal : D , B , E , A , F , C , G

Inorder Traversal Shortcut

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



Inorder Traversal : D , B , E , A , F , C , G

Application-

- Inorder traversal is used to get infix expression of an expression tree.

3. Postorder Traversal-

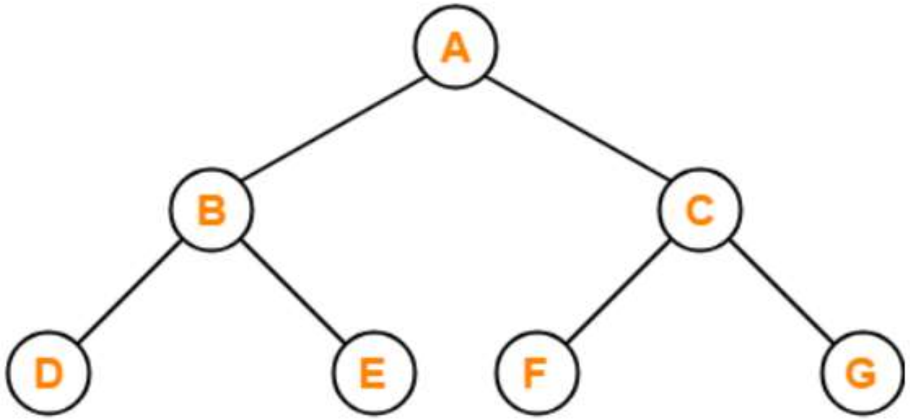
Algorithm-

1. Traverse the left sub tree i.e. call Postorder (left sub tree)
2. Traverse the right sub tree i.e. call Postorder (right sub tree)
3. Visit the root

Left → Right → Root

Example-

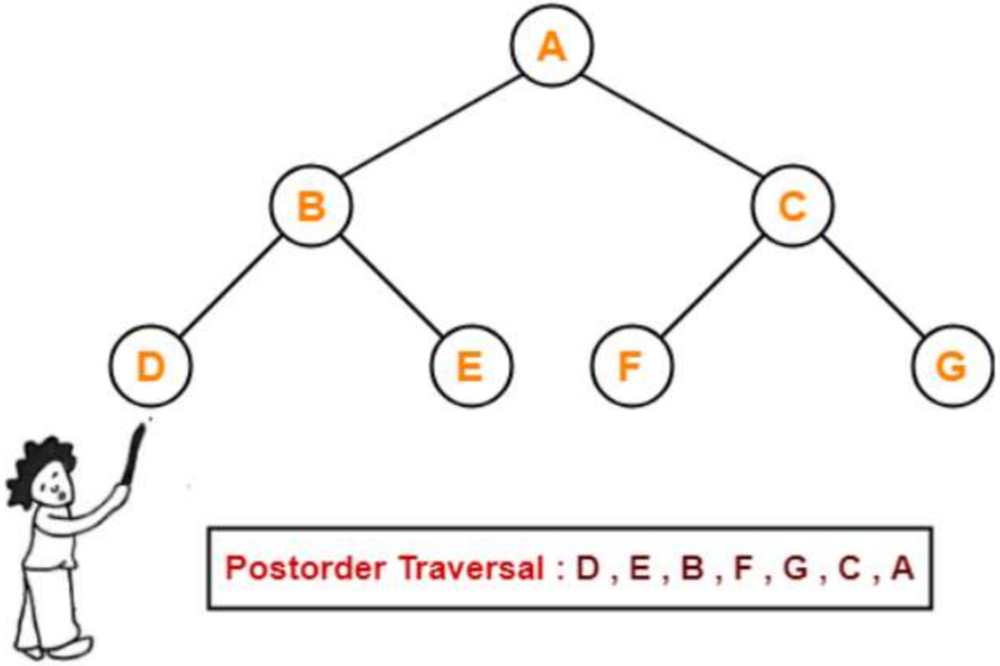
Consider the following example-



Postorder Traversal : D , E , B , F , G , C , A

Postorder Traversal Shortcut

Pluck all the leftmost leaf nodes one by one.



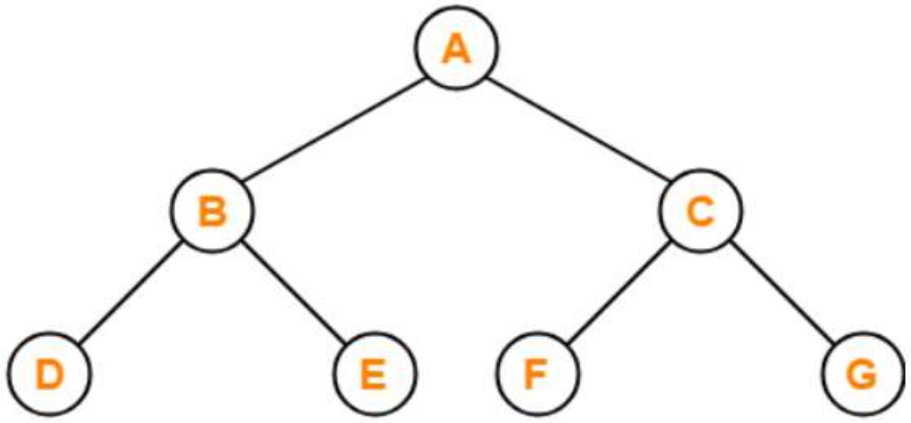
Applications-

- Postorder traversal is used to get postfix expression of an expression tree.
- Postorder traversal is used to delete the tree.
- This is because it deletes the children first and then it deletes the parent.

Breadth First Traversal-

- Breadth First Traversal of a tree prints all the nodes of a tree level by level.
- Breadth First Traversal is also called as **Level Order Traversal**.

Example-



Level Order Traversal : A , B , C , D , E , F , G

Application-

- Level order traversal is used to print the data in the same order as stored in the array representation of a complete binary tree.

Binary Search Tree

https://www.gatevidyalay.com/binary-search-trees-data-structures/

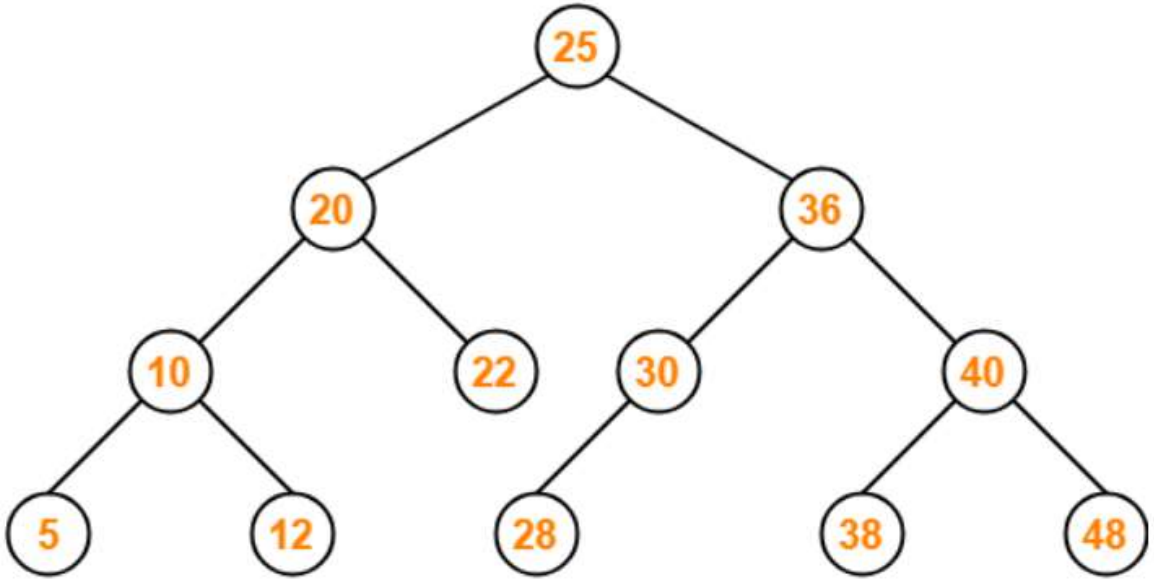
Binary Search Tree- Sorted Binary Tree

> Binary Search Tree is a special kind of binary tree in which nodes are arranged in a specific order.

In a binary search tree (BST), each node contains-

- Only smaller values in its left sub tree
- Only larger values in its right sub tree

Example-



Binary Search Tree

Binary Search Tree Construction-

Let us understand the construction of a binary search tree using the following example-

Example-

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

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

When elements are given in a sequence,

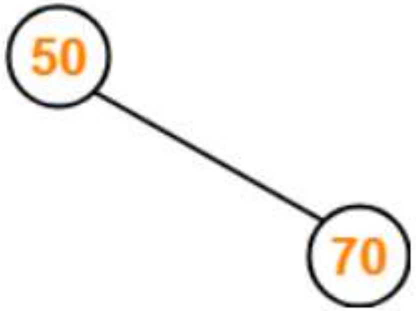
- Always consider the first element as the root node.
- Consider the given elements and insert them in the BST one by one.

The binary search tree will be constructed as explained below-

Insert 50-

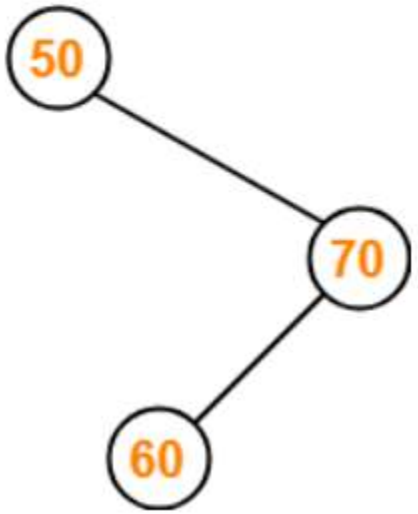
Insert 70-

- As 70 > 50, so insert 70 to the right of 50.



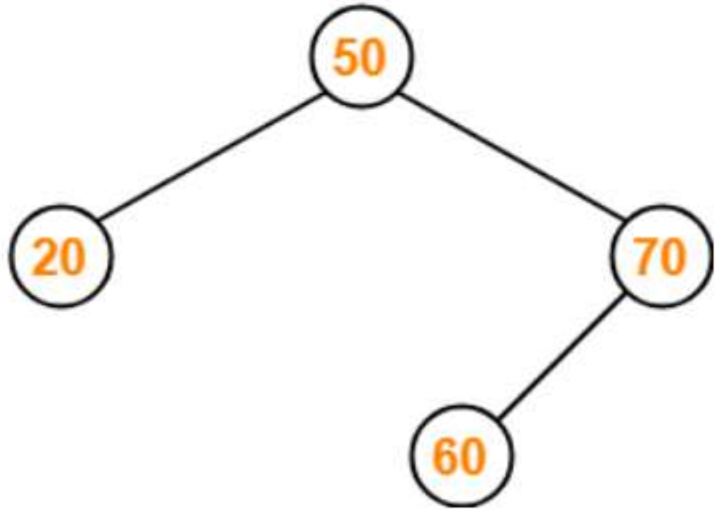
Insert 60-

- As 60 > 50, so insert 60 to the right of 50.
- As 60 < 70, so insert 60 to the left of 70.



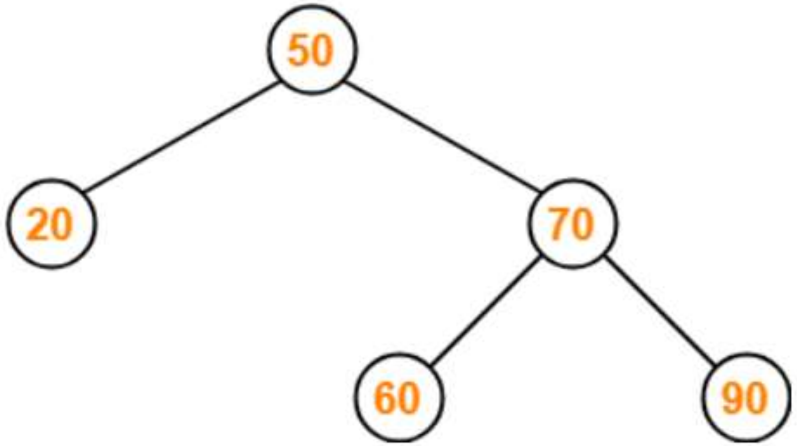
Insert 20-

- As 20 < 50, so insert 20 to the left of 50.



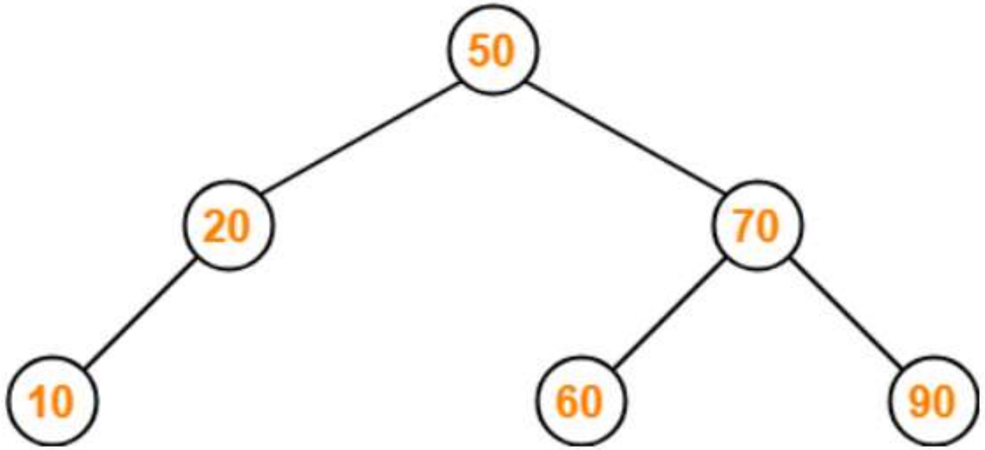
Insert 90-

- As 90 > 50, so insert 90 to the right of 50.
- As 90 > 70, so insert 90 to the right of 70.



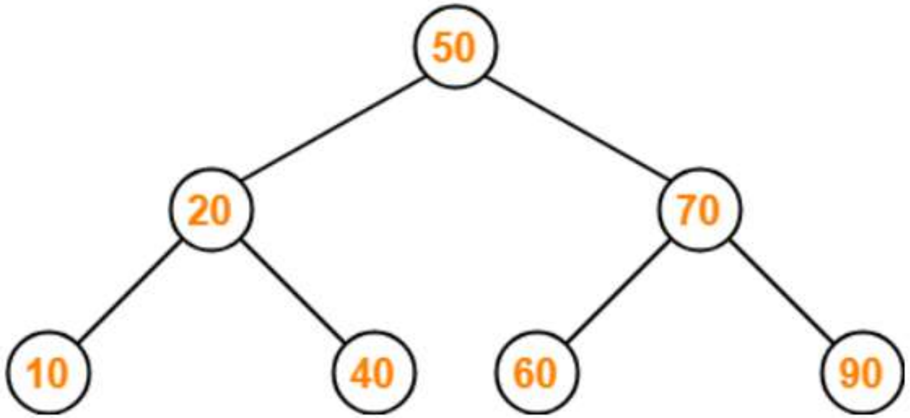
Insert 10-

- As 10 < 50, so insert 10 to the left of 50.
- As 10 < 20, so insert 10 to the left of 20.



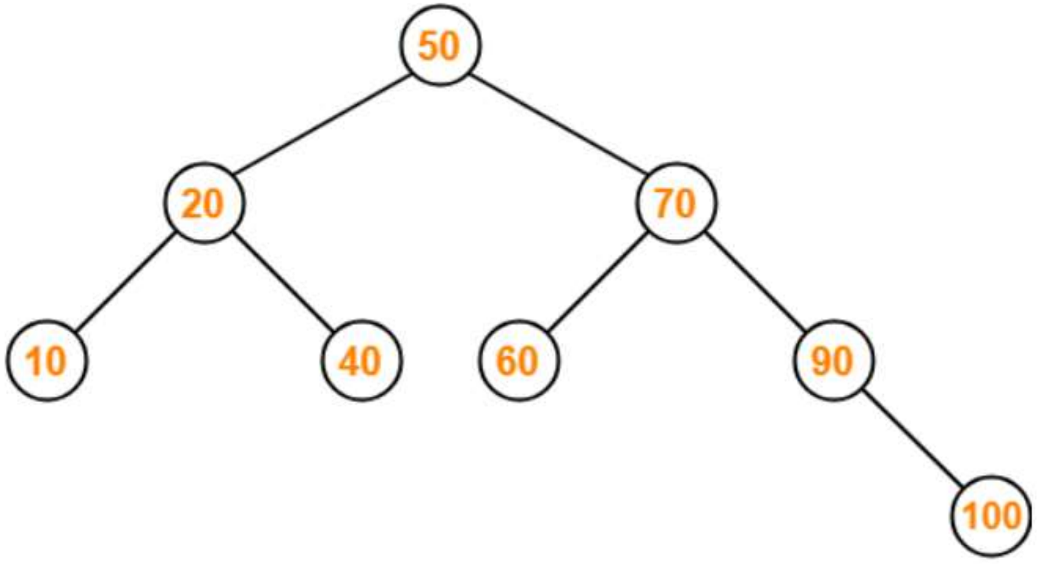
Insert 40-

- As 40 < 50, so insert 40 to the left of 50.
- As 40 > 20, so insert 40 to the right of 20.



Insert 100-

- As 100 > 50, so insert 100 to the right of 50.
- As 100 > 70, so insert 100 to the right of 70.
- As 100 > 90, so insert 100 to the right of 90.

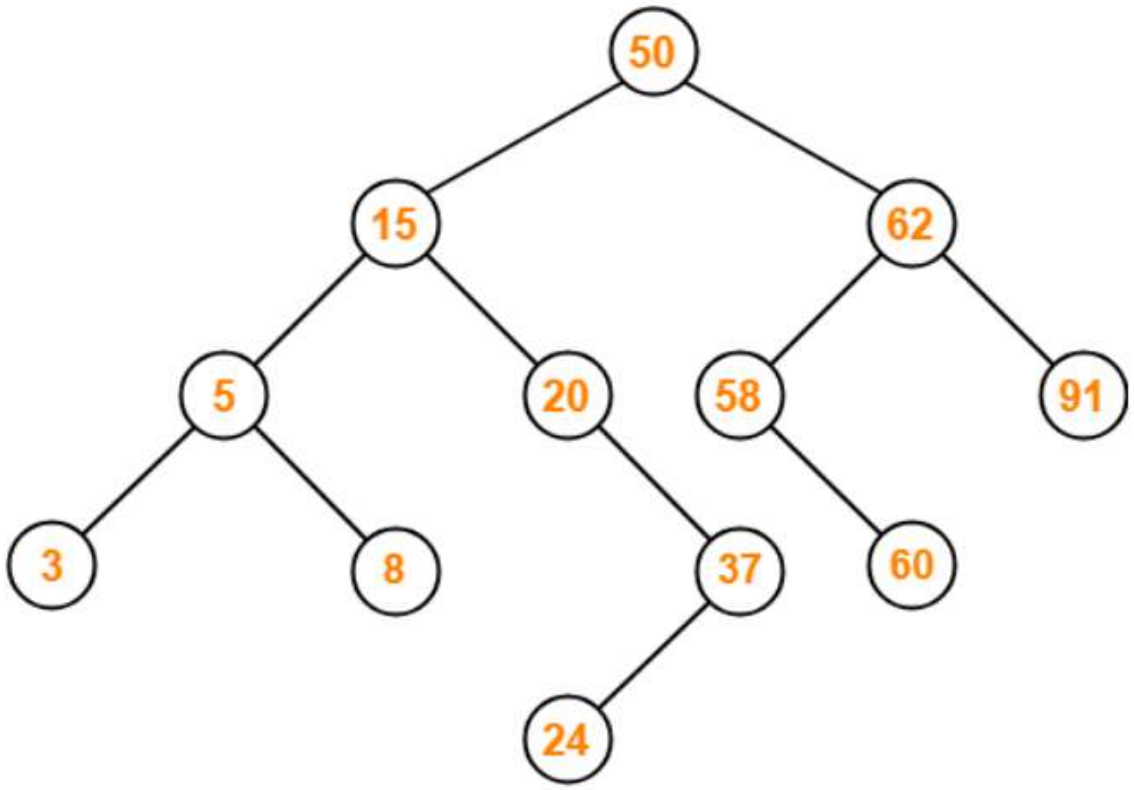


Problem-01:

A binary search tree is generated by inserting in order of the following integers-

50, 15, 62, 5, 20, 58, 91, 3, 8, 37, 60, 24

The resultant binary search tree will be-



Binary Search Tree

Binary Search Tree Traversal

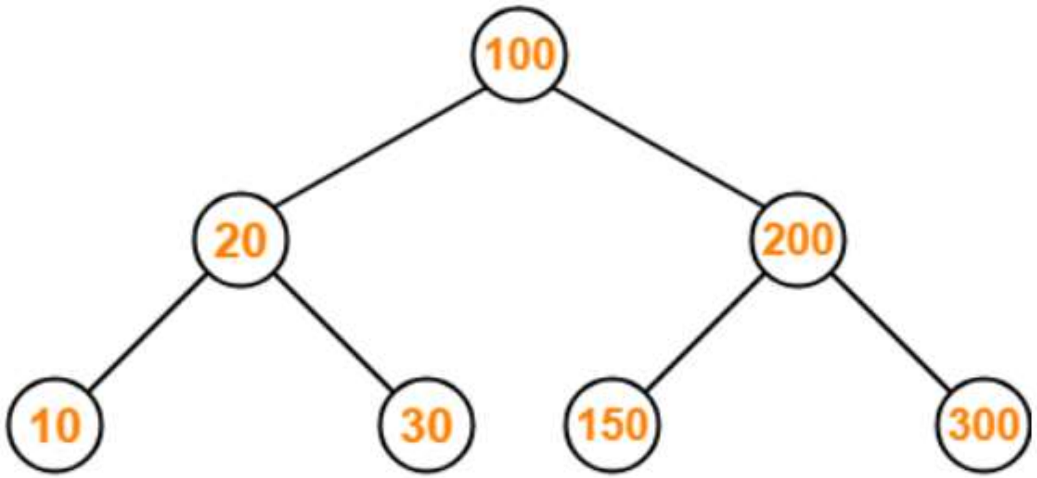
https://www.gatevidyalay.com/binary-search-tree-traversal-bst-traversal/

BST Traversal-

- A binary search tree is traversed in exactly the same way a binary tree is traversed.
- In other words, BST traversal is same as binary tree traversal.

Example-

Consider the following binary search tree-



Binary Search Tree

Now, let us write the traversal sequences for this binary search tree-

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 Notes-

Note-01:

- Inorder traversal of a binary search tree always yields all the nodes in increasing order.

Note-02:

Unlike **Binary Trees**,

- A binary search tree can be constructed using only preorder or only postorder traversal result.
- This is because inorder traversal can be obtained by sorting the given result in increasing order.
  (gives a right skewed binary tree every time).

Binary Search Tree Operations

https://www.gatevidyalay.com/binary-search-tree-insertion-bst-deletion/

Binary Search Tree Operations-

Commonly performed operations on binary search tree are-

1. Search Operation
2. Insertion Operation
3. Deletion Operation

1. Search Operation-

Search Operation is performed to search a particular element in the Binary Search Tree.

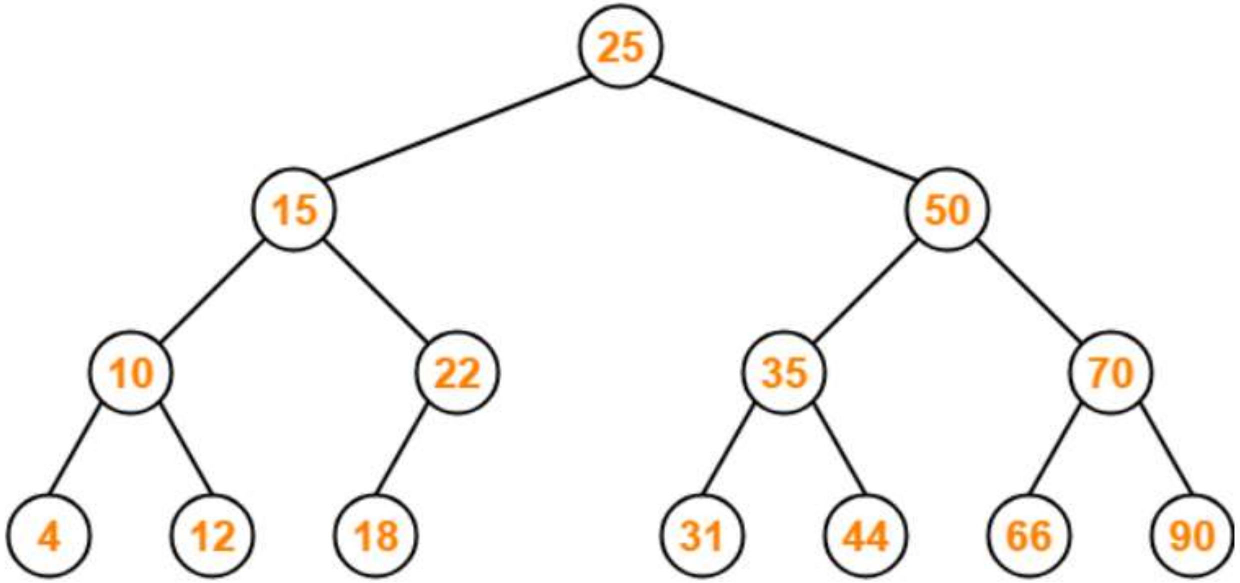
Rules-

For searching a given key in the BST,

- Compare the key with the value of root node.
- If the key is present at the root node, then return the root node.
- If the key is greater than the root node value, then recur for the root node's right subtree.
- If the key is smaller than the root node value, then recur for the root node's left subtree.

Example-

Consider key = 45 has to be searched in the given BST-



Binary Search Tree

- We start our search from the root node 25.
- As 45 > 25, so we search in 25's right subtree.
- As 45 < 50, so we search in 50's left subtree.
- As 45 > 35, so we search in 35's right subtree.
- As 45 > 44, so we search in 44's right subtree but 44 has no subtrees.
- So, we conclude that 45 is not present in the above BST.

2. Insertion Operation-

Insertion Operation is performed to insert an element in the Binary Search Tree.

Rules-

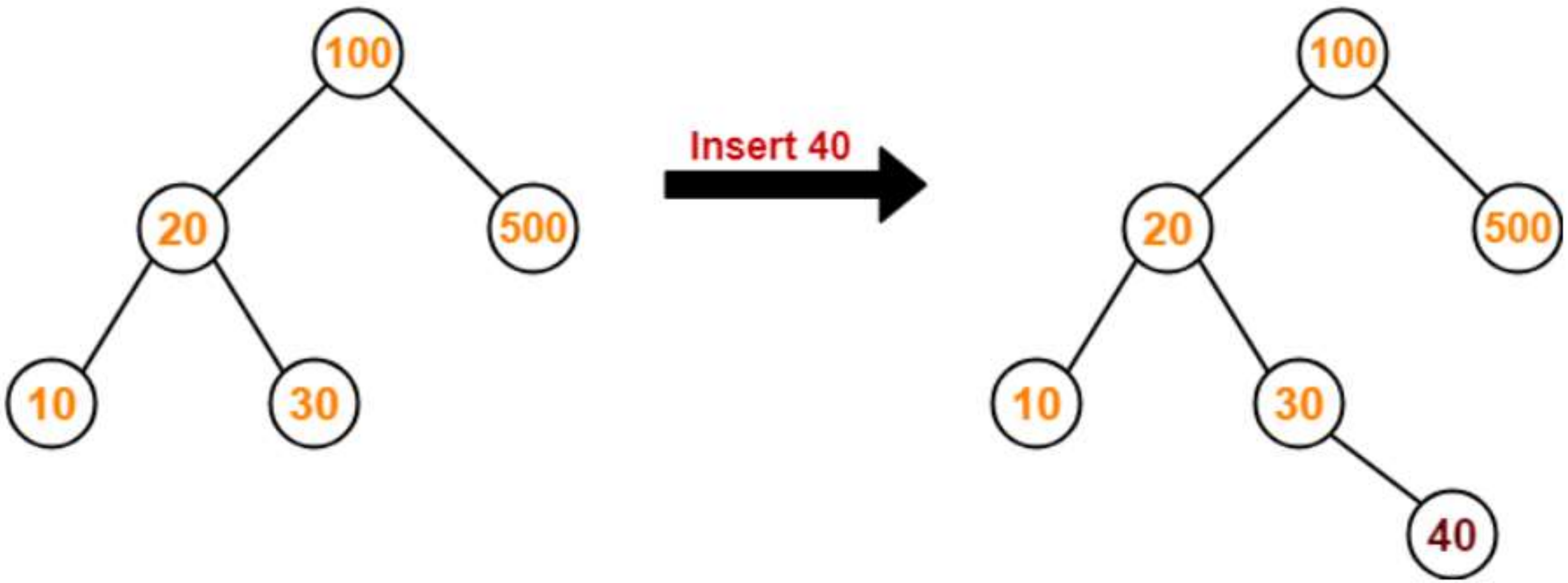
The insertion of a new key always takes place as the child of some leaf node.

For finding out the suitable leaf node,

- Search the key to be inserted from the root node till some leaf node is reached.
- Once a leaf node is reached, insert the key as child of that leaf node.

Example-

Consider the following example where key = 40 is inserted in the given BST-



- We start searching for value 40 from the root node 100.
- As 40 < 100, so we search in 100's left subtree.
- As 40 > 20, so we search in 20's right subtree.
- As 40 > 30, so we add 40 to 30's right subtree.

3. Deletion Operation-

Deletion Operation is performed to delete a particular element from the Binary Search Tree.

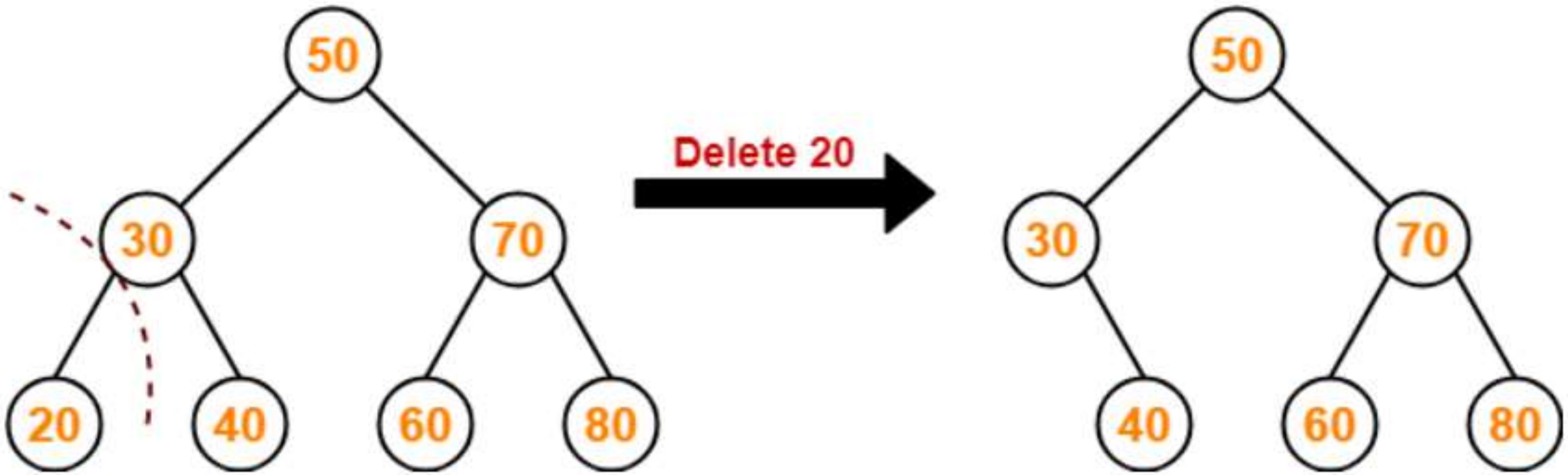
When it comes to deleting a node from the binary search tree, following three cases are possible-

Case-01: Deletion Of A Node Having No Child (Leaf Node)-

Just remove / disconnect the leaf node that is to deleted from the tree.

Example-

Consider the following example where node with value = 20 is deleted from the BST-

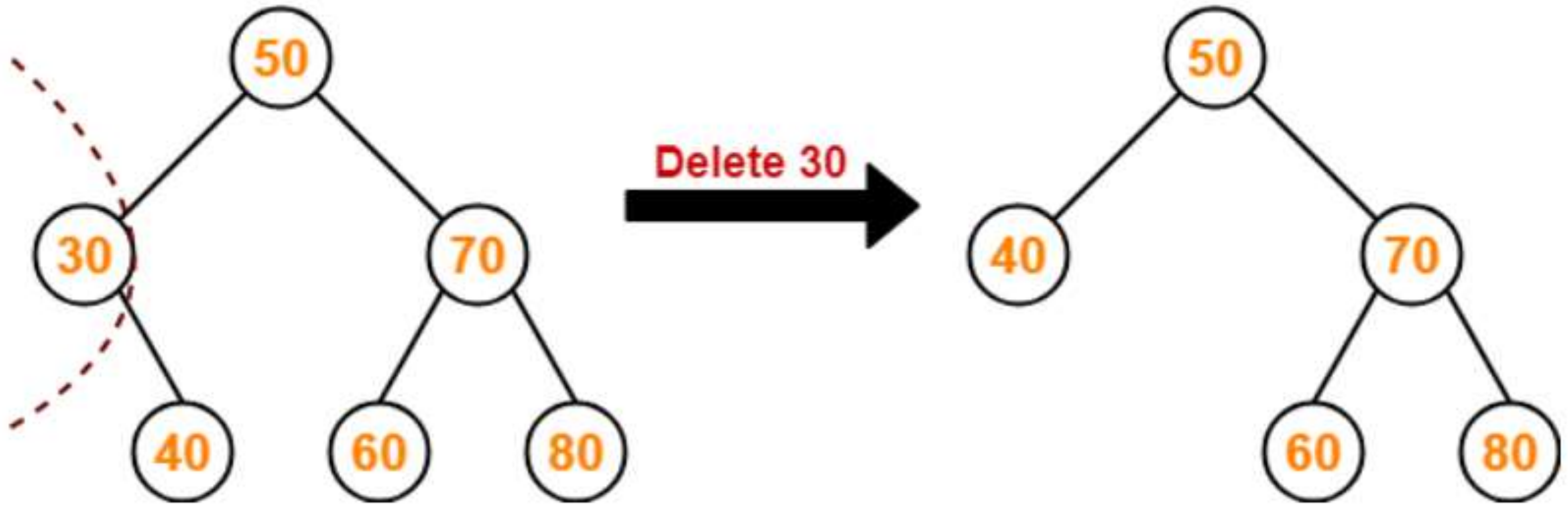


Case-02: Deletion Of A Node Having Only One Child-

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

Example-

Consider the following example where node with value = 30 is deleted from the BST-



Case-02: Deletion Of A Node Having Two Children-

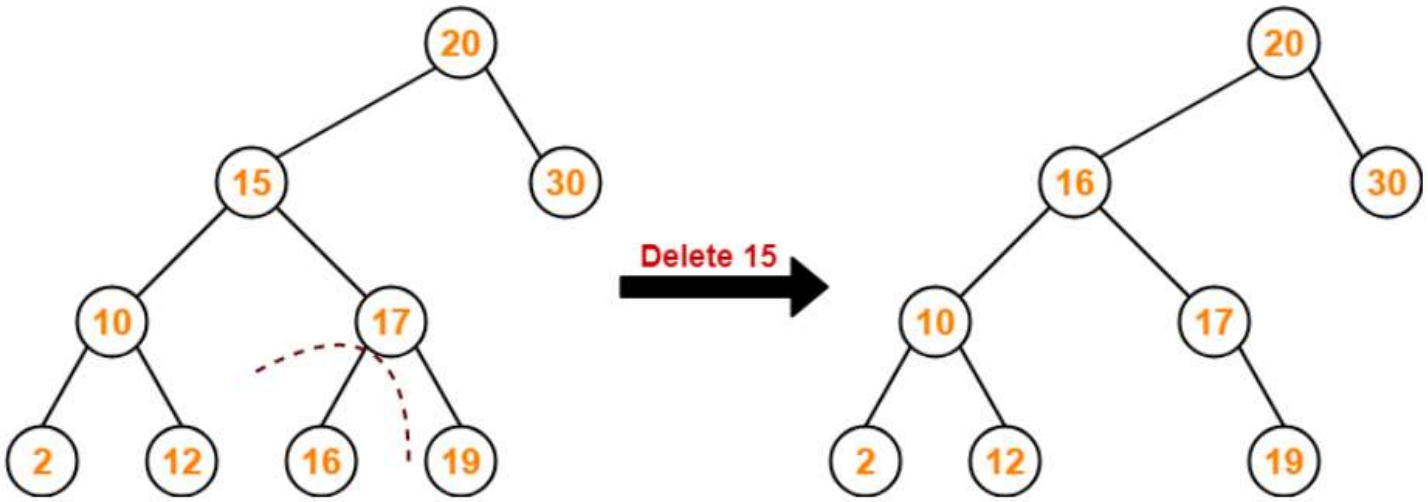
A node with two children may be deleted from the BST in the following two ways-

Method-01:

- Visit to the right subtree of the deleting node.
- Pluck the least value element called as inorder successor.
- Replace the deleting element with its inorder successor.

Example-

Consider the following example where node with value = 15 is deleted from the BST-

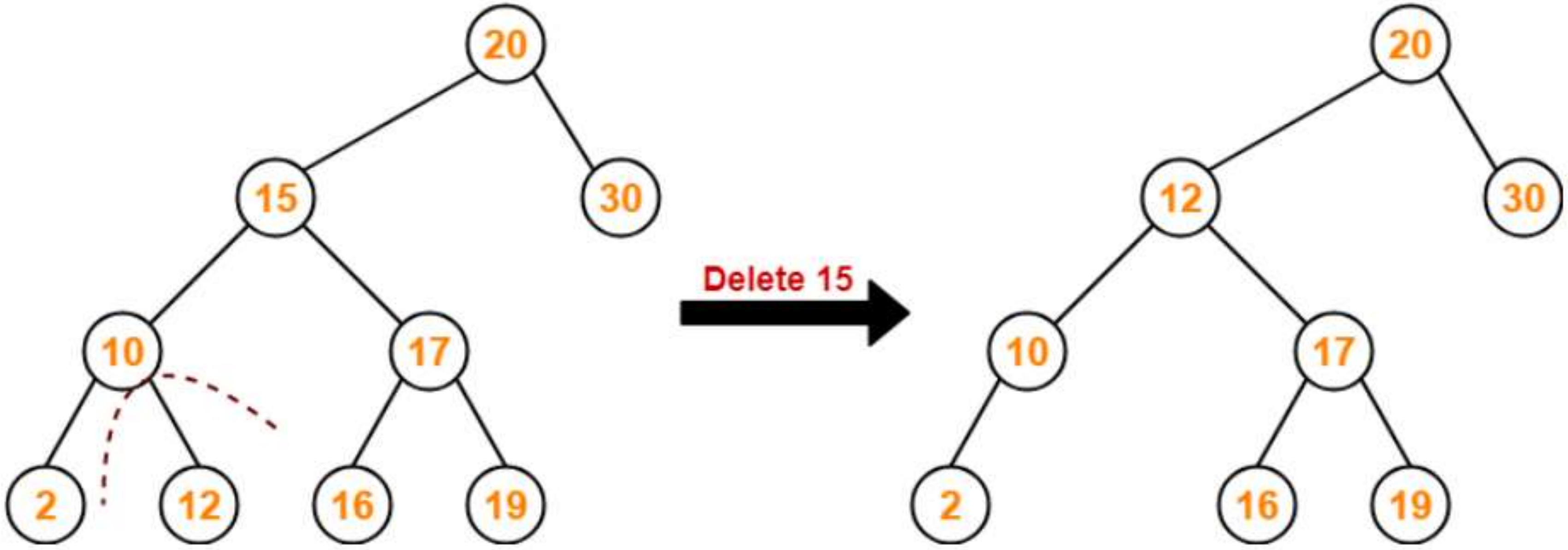


Method-02:

- Visit to the left subtree of the deleting node.
- Pluck the greatest value element called as inorder predecessor.
- Replace the deleting element with its inorder predecessor.

Example-

Consider the following example where node with value = 15 is deleted from the BST-



The End