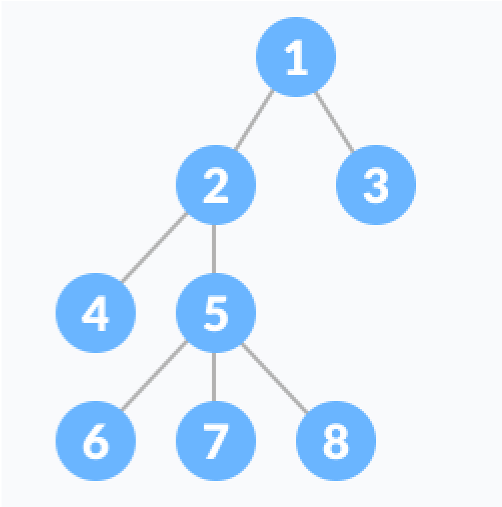
Trees - Introduction

- A tree is a nonlinear hierarchical data structure that consists of nodes connected by edges.
- Different tree data structures allow quicker and easier access to the data as it is a non-linear data structure.



Properties of Trees

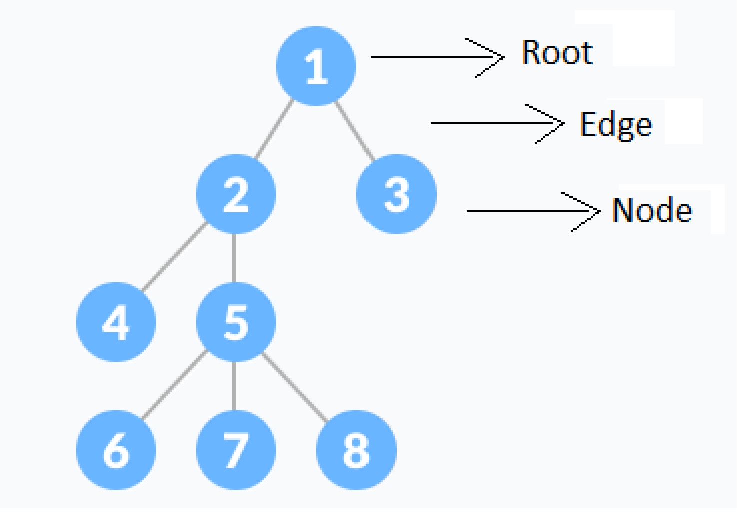
- There is one and only one path between every pair of vertices in a tree.
- A tree with n vertices has n-1 edges.
- A graph is a tree if and if only if it is minimally connected.
- Any connected graph with n vertices and n-1 edges is a tree.

Tree Applications

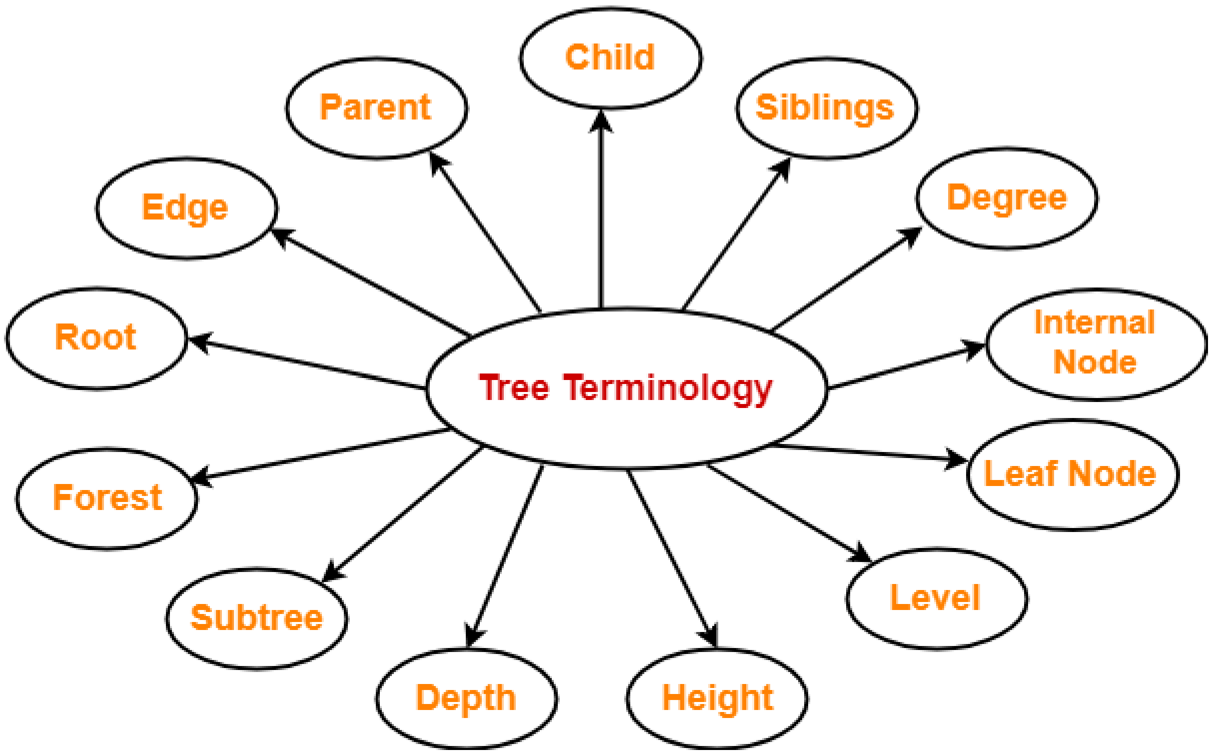
- Binary Search Trees(BSTs) are used to quickly check whether an element is present in a set or not.
- Heap is a kind of tree that is used for heap sort.
- A modified version of a tree called Tries is used in modern routers to store routing information.
- Most popular databases use B-Trees and T-Trees, which are variants of the tree structure we learned above to store their data
- Compilers use a syntax tree to validate the syntax of every program you write.

Tree terminologies

- **Node** - A node is an entity that contains a key or value and pointers to its child nodes.
  - The last nodes of each path are called **leaf nodes or external nodes** that do not contain a link/pointer to child nodes.
  - The node having at least a child node is called an **internal node**.
- **Edge -**It is the link between any two nodes.
- **Root -** It is the topmost node of a tree.

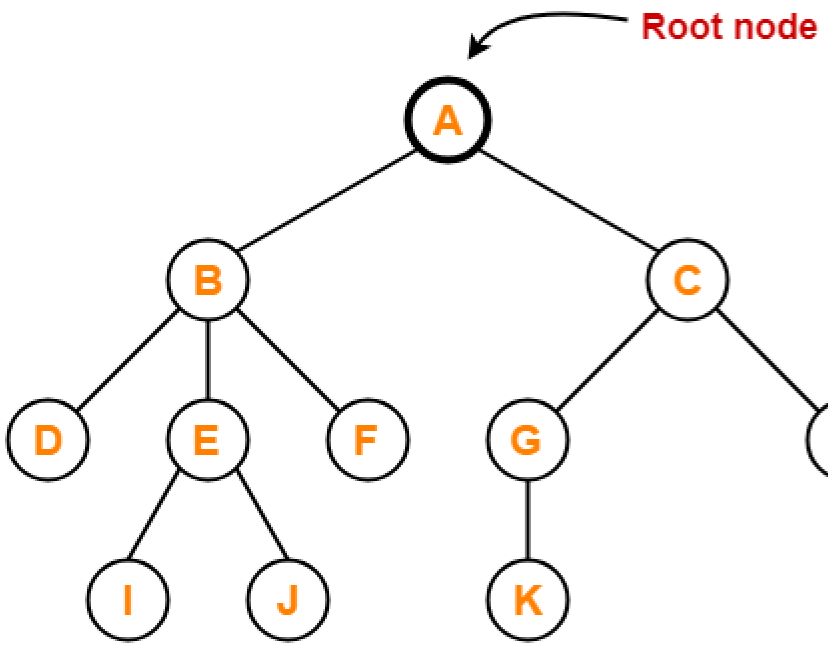


Tree terminologies



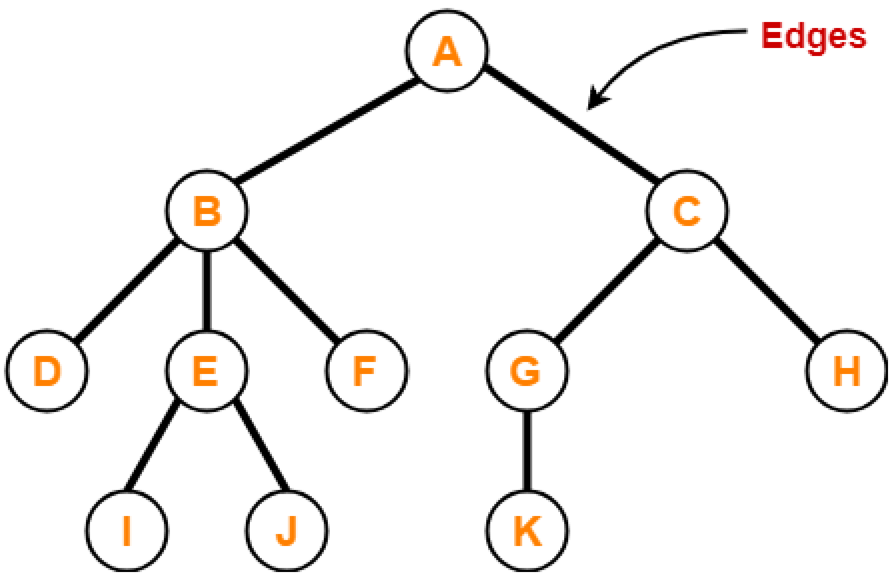
Root

- 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.



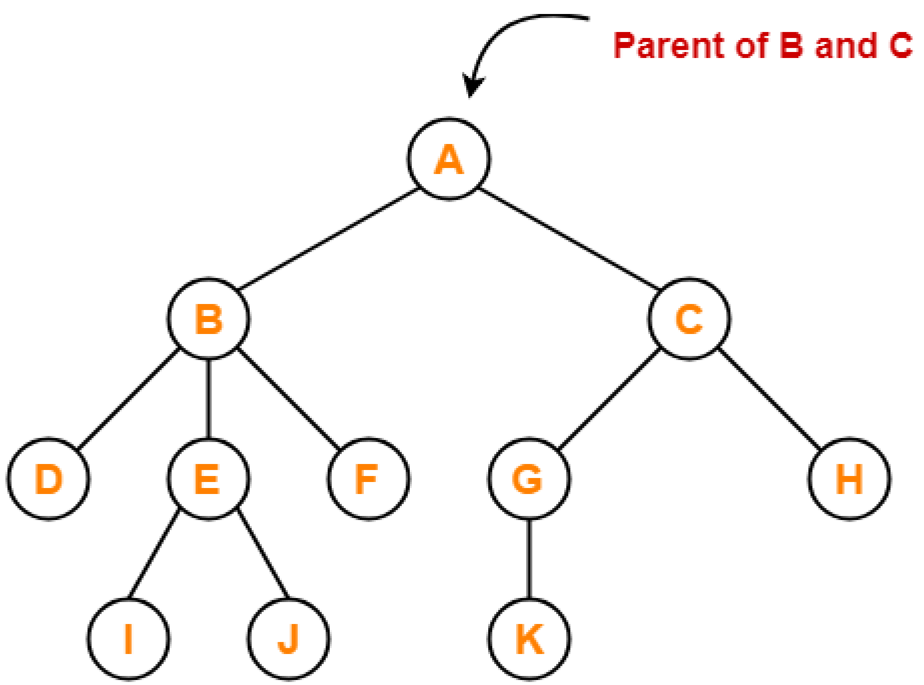
Edge

- 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.



Parent

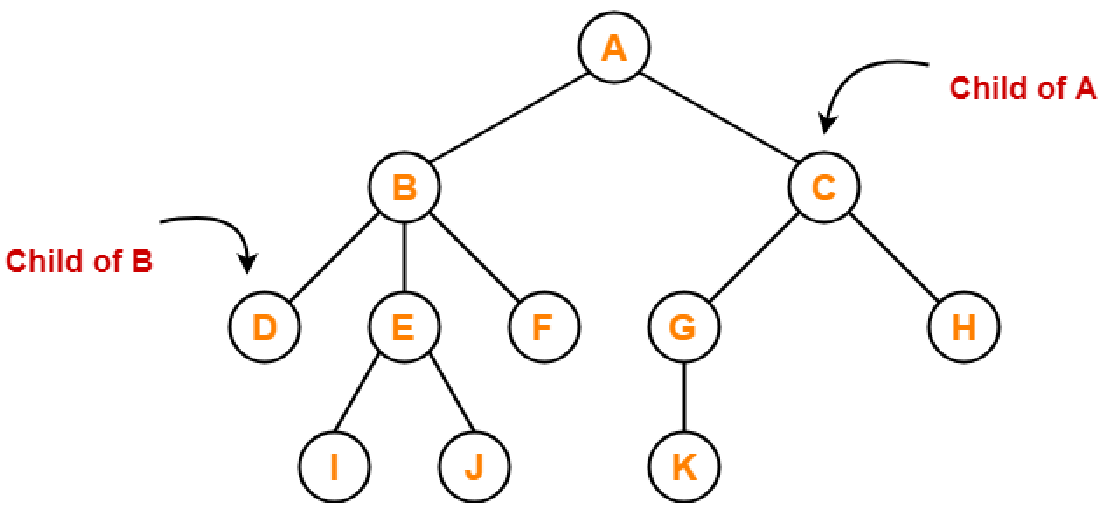
- The node which has a branch from it to any other node is called as a **parent node**.
- In other words, the node which has one or more children is called as a parent node.
- In a tree, a parent node can have any number of child nodes.



- 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

Child

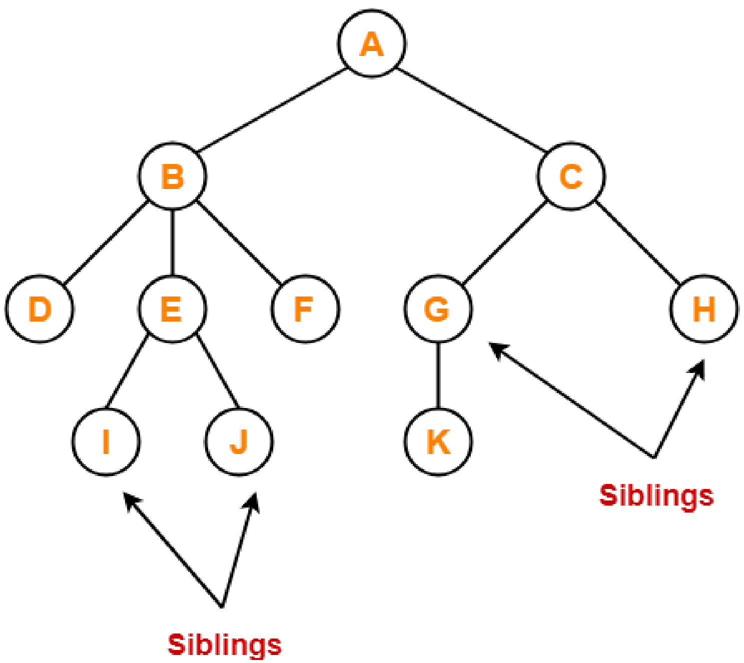
- The node which is a descendant of some node is called as a **child node**.
- All the nodes except root node are child nodes.



- 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

Siblings

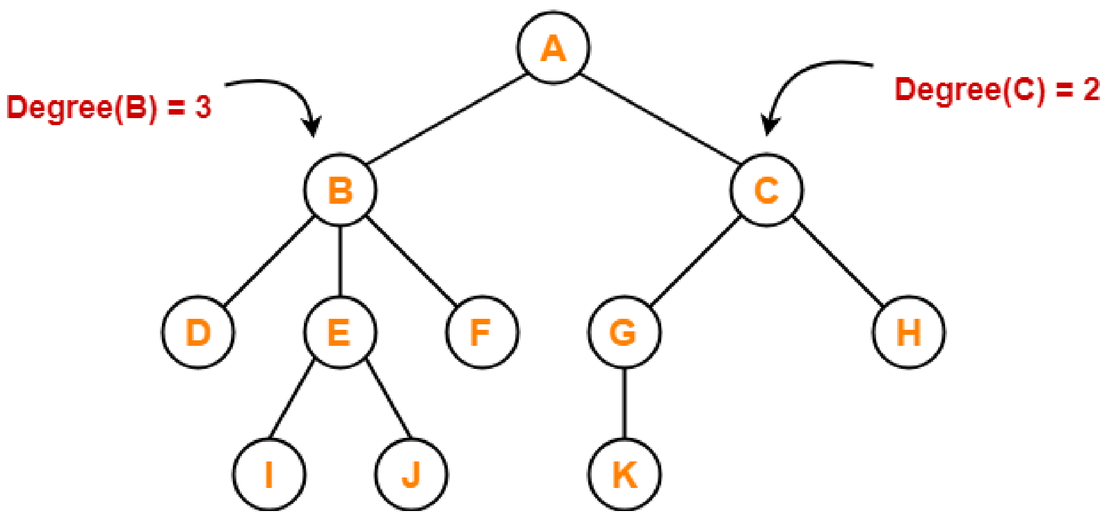
- Nodes which belong to the same parent are called as **siblings**.
- In other words, nodes with the same parent are sibling nodes.



- 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

Degree

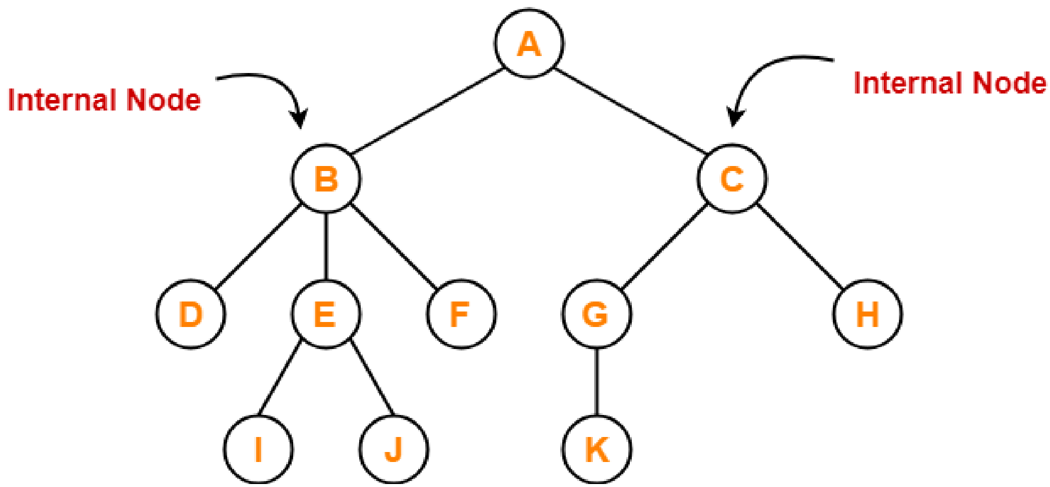
- **Degree of a node** is the total number of children of that node.
- **Degree of a tree** is the highest degree of a node among all the nodes in the tree.



- 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

Internal node

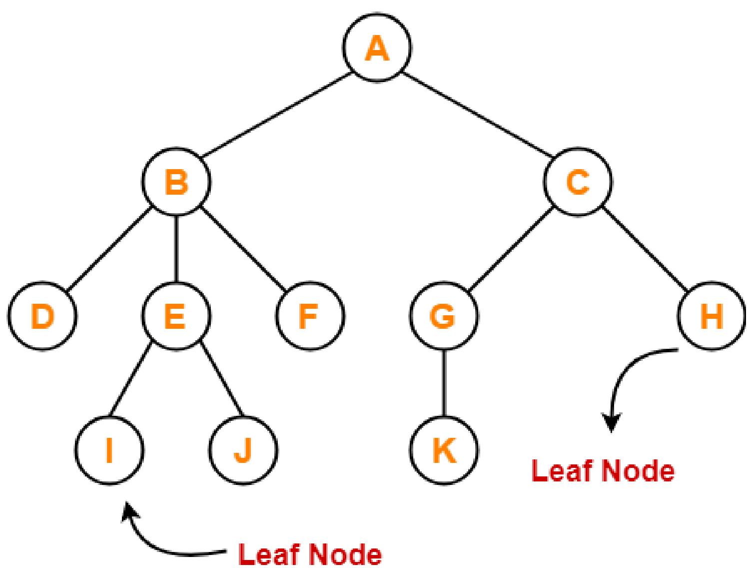
- The node which has at least one child is called as an **internal node**.
- Internal nodes are also called as **non-terminal nodes**.
- Every non-leaf node is an internal node.



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

Leaf node

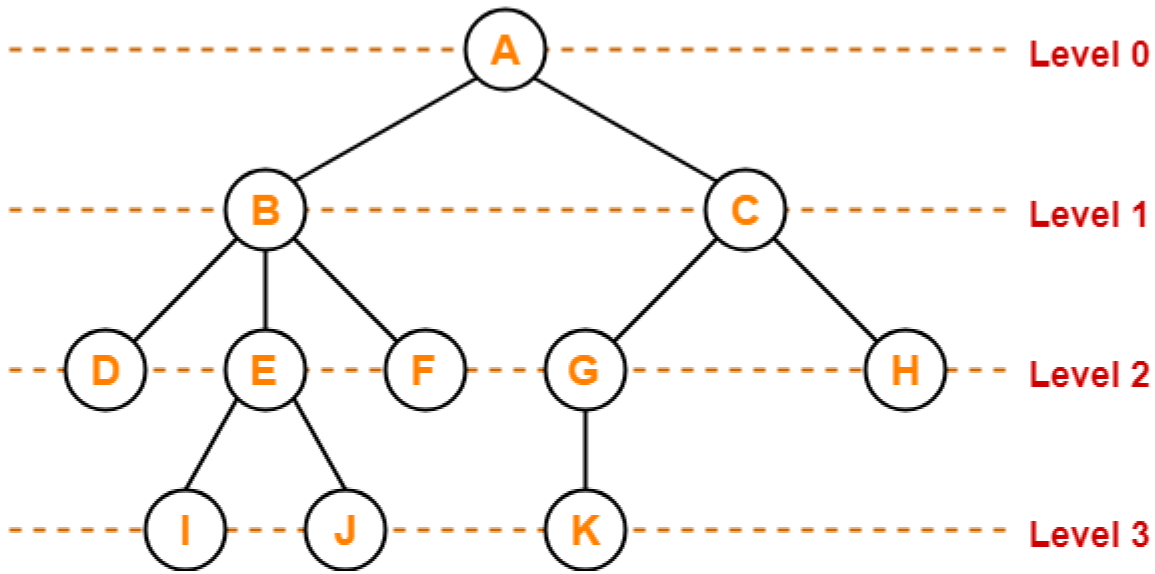
- The node which does not have any child is called as a **leaf node**.
- Leaf nodes are also called as **external nodes** or **terminal nodes**.



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

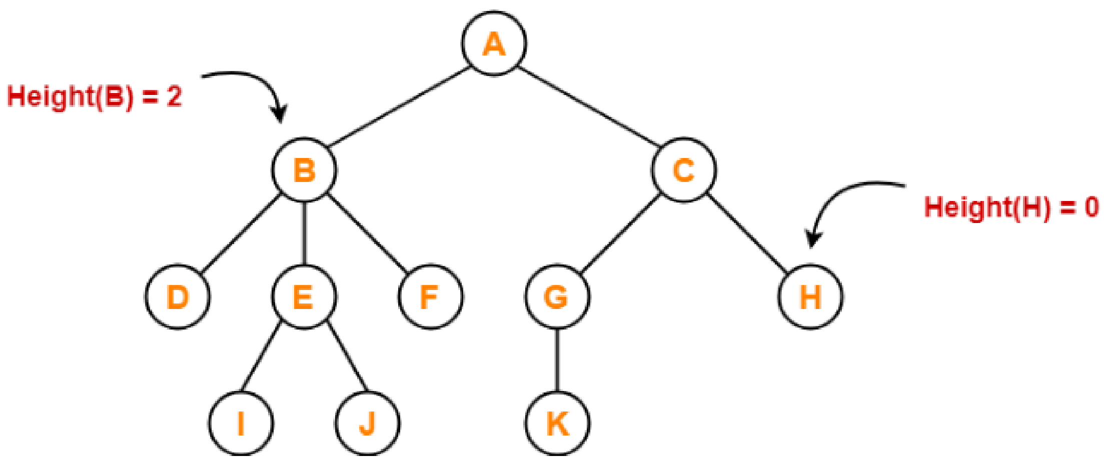
Level

- 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.



Height

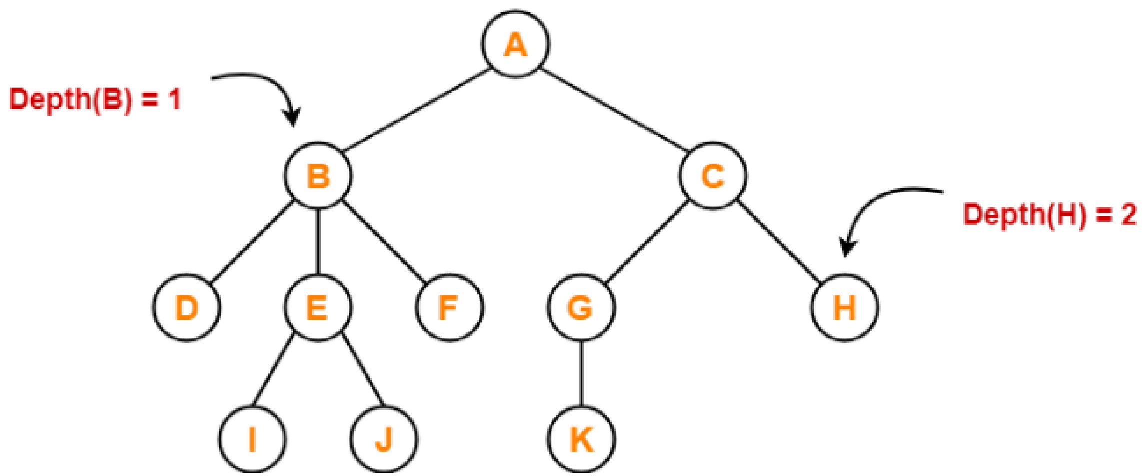
- Total number of edges that lies on the longest path from any leaf node to a particular node is called as **height of that node**.
- **Height of a tree** is the height of root node.
- Height of all leaf nodes = 0



- 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

Depth

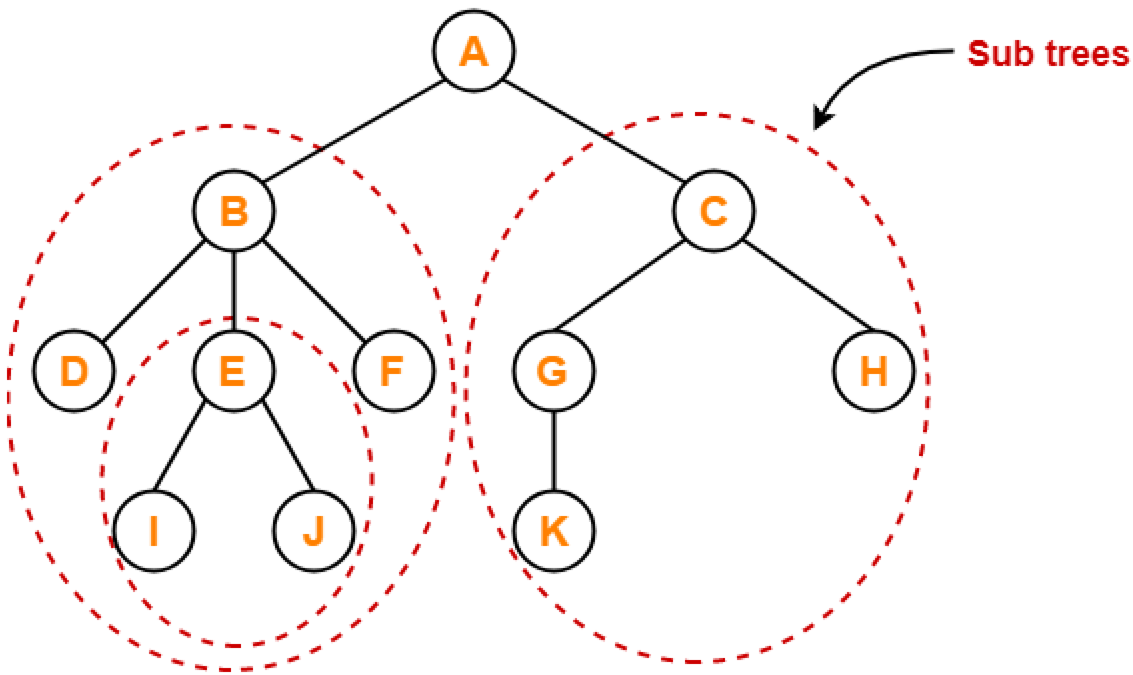
- Total number of edges from root node to a particular node is called as **depth of that node**.
- **Depth of a tree** is the total number of edges from root node to a leaf node in the longest path.
- Depth of the root node = 0
- The terms “level” and “depth” are used interchangeably.



- 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

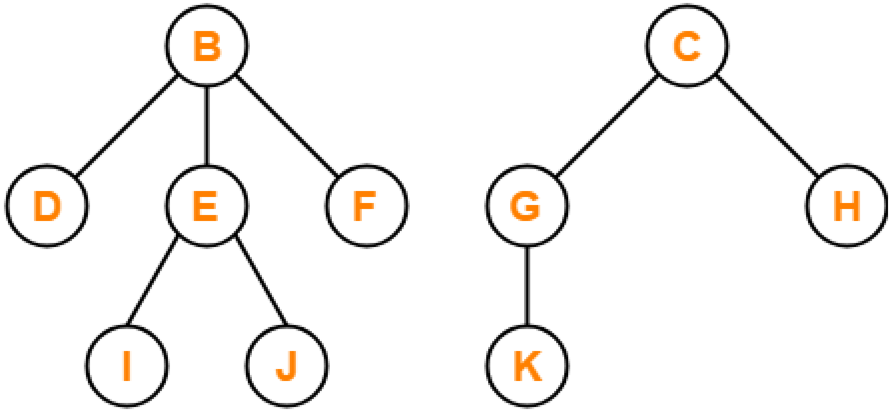
Subtree

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



Forest

- A forest is a set of disjoint trees.



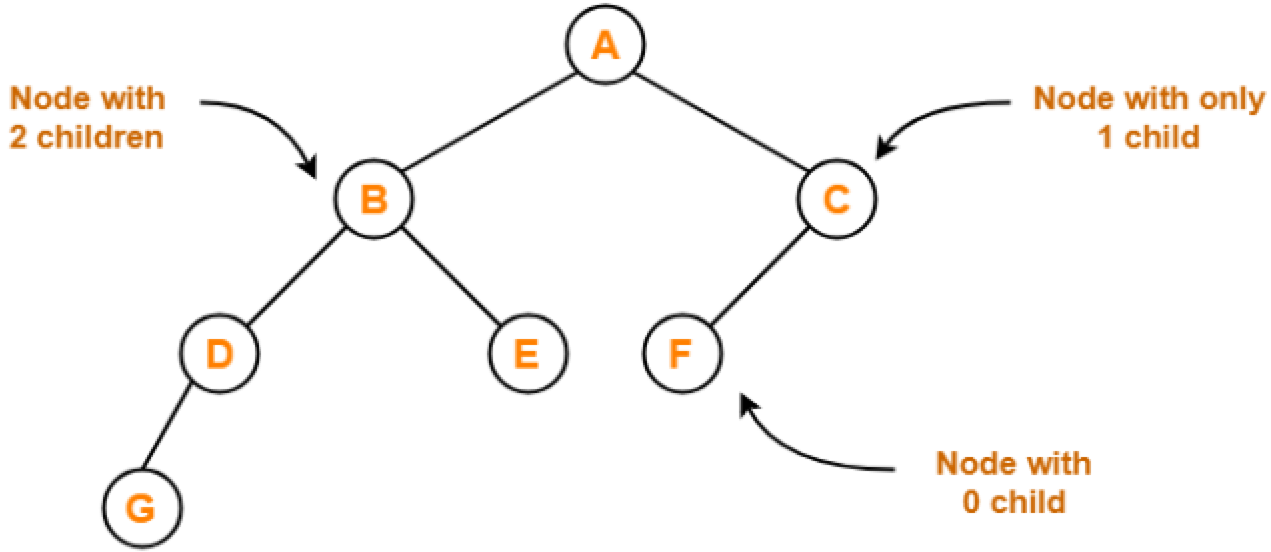
Forest

Types of Tree

- General Tree
- Binary Tree
- Binary Search Tree
- AVL Tree
- Red-Black Tree
- N-ary Tree

Binary Tree

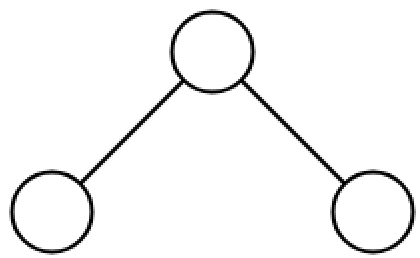
- Binary tree is a special tree data structure in which each node can have at most 2 children.
- Thus, in a binary tree, Each node has either 0 child or 1 child or 2 children.



Binary Tree Example

Unlabeled Binary Tree

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

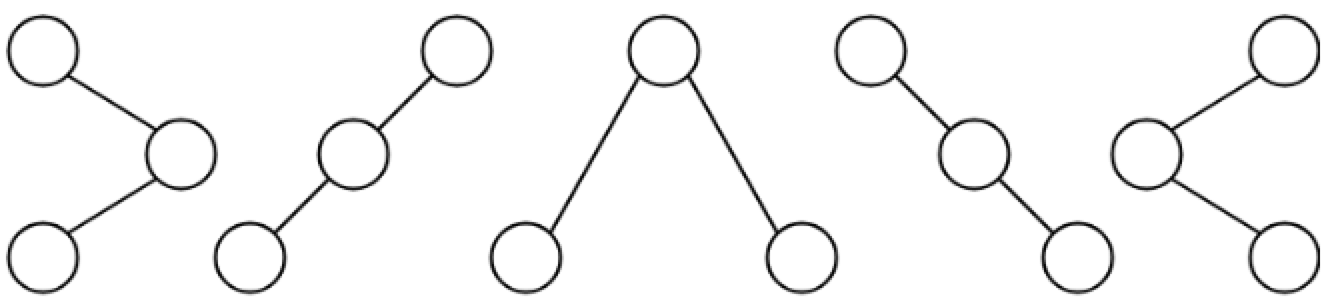


Unlabeled Binary Tree

Number of different Binary Trees possible with 'n' unlabeled nodes $= \dfrac{{}^{2n}C_{n}}{n+1}$

Example

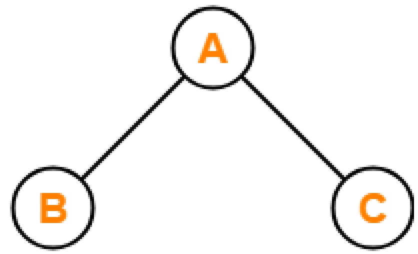
- Consider we want to draw all the binary trees possible
- Number of binary trees possible with 3 unlabeled nodes
- $= {}^{2 \times 3}C_3 / (3 + 1)$
- $= {}^{6}C_3 / 4$
- $= 5$



Binary Trees Possible With 3 Unlabeled Nodes

Labeled Binary Tree

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

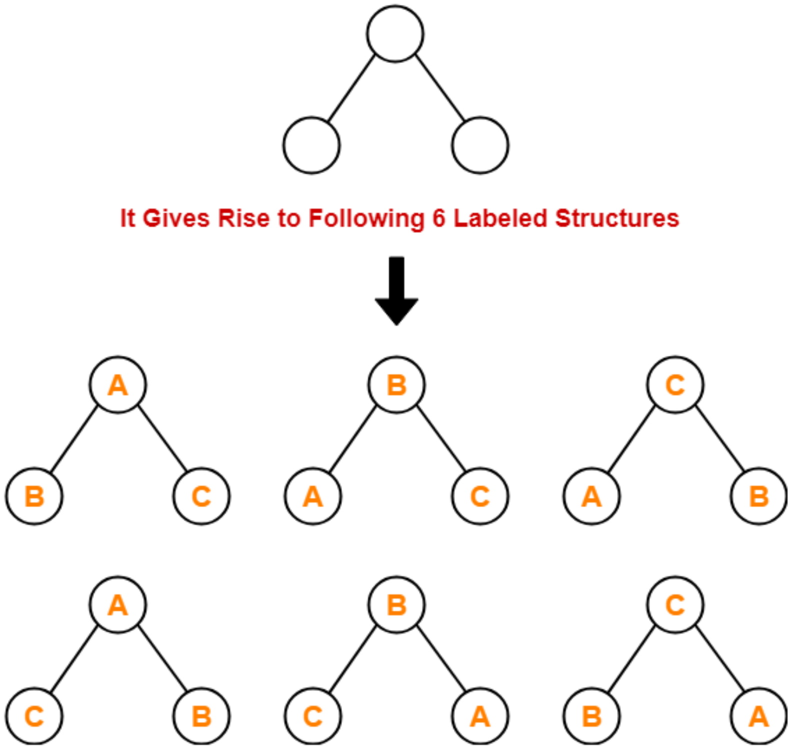


Labeled Binary Tree

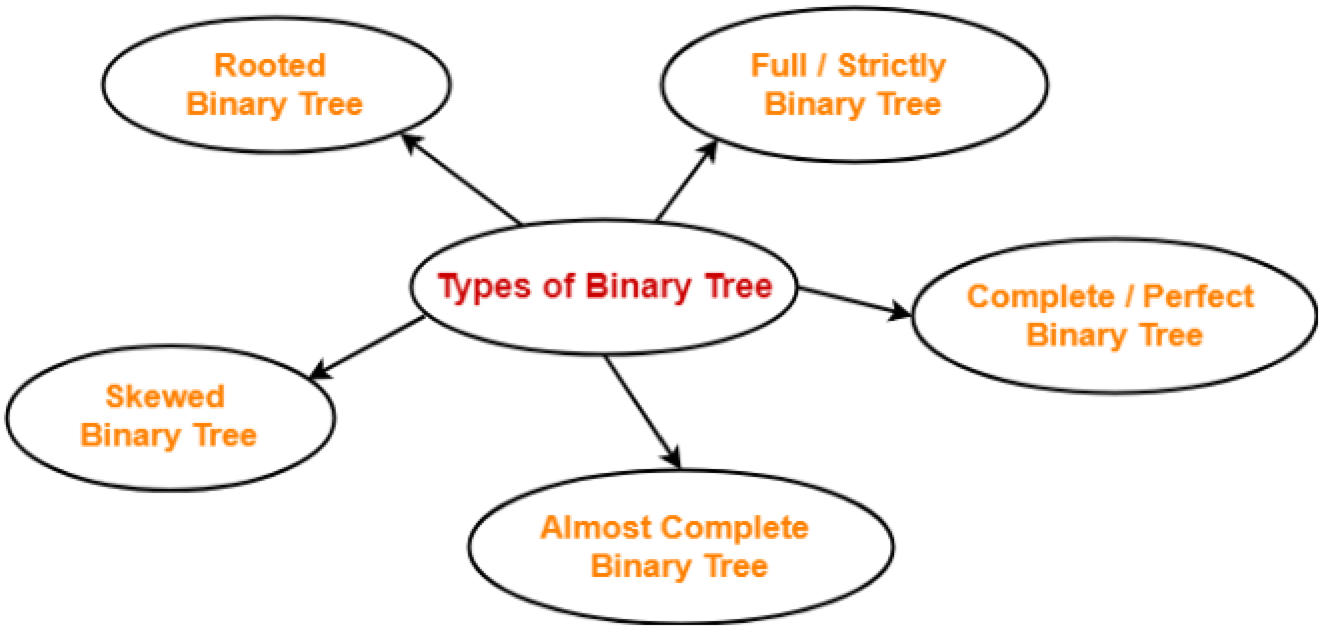
Number of different Binary Trees possible with 'n' labeled nodes $= \dfrac{^{2n}C_n}{n+1} \times n!$

Example

- Consider we want to draw all the binary trees possible with 3 labeled nodes.
- Number of binary trees possible with 3 labeled nodes
- $= \{ {}^{2 \times 3}C_3 / (3 + 1) \} \times 3!$
- $= \{ {}^{6}C_3 / 4 \} \times 6$
- $= 5 \times 6$
- $= 30$

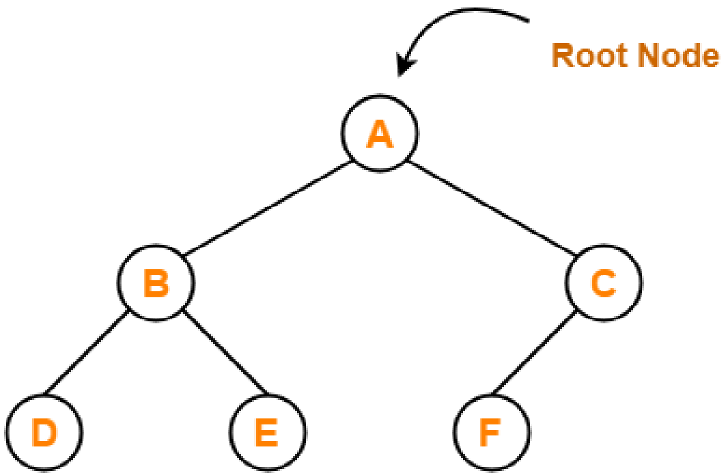


Types of Binary Trees



Rooted Binary Tree

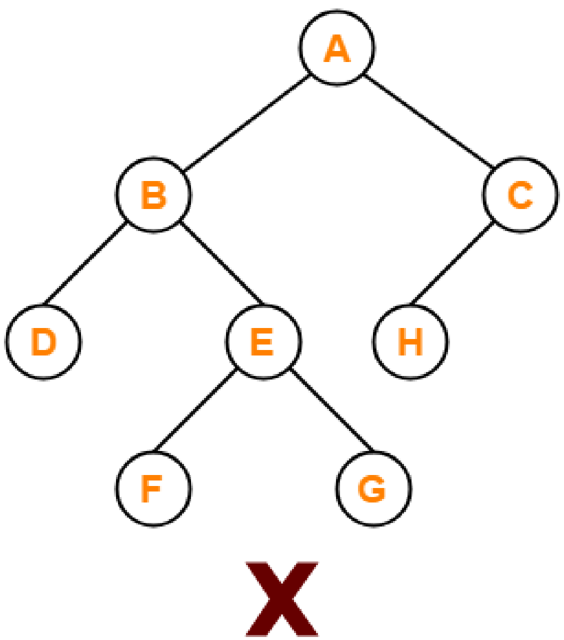
- A **rooted binary tree** is a binary tree that satisfies the following 2 properties:
  - It has a root node.
  - Each node has at most 2 children.

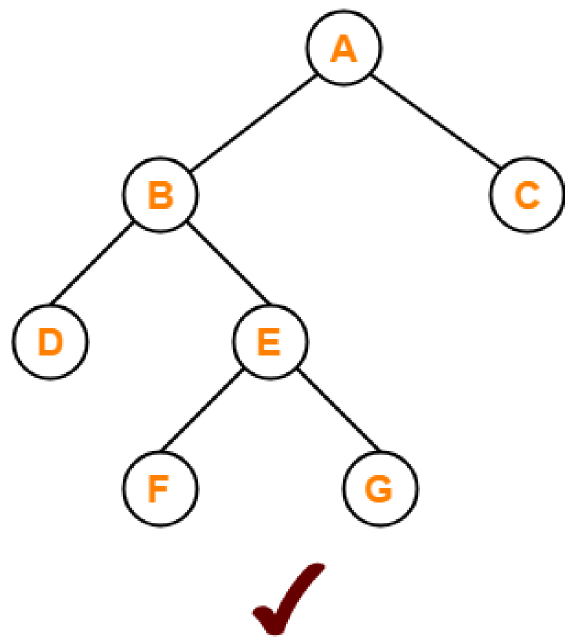


Rooted Binary Tree

Full/Strictly Binary Tree

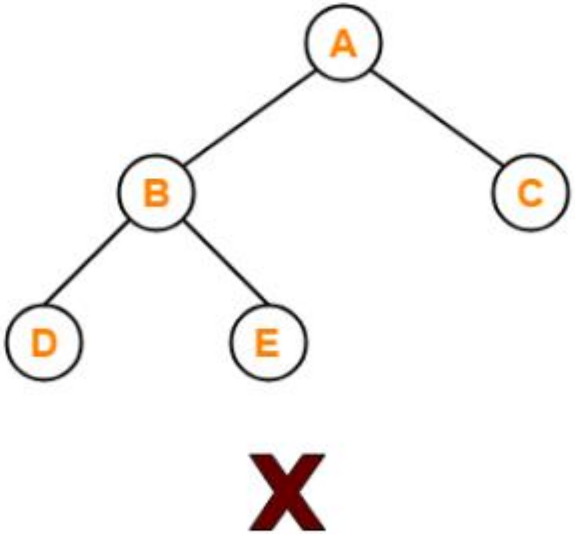
- A binary tree in which every node has either 0 or 2 children is called as a **Full binary tree**.
- Full binary tree is also called as **Strictly binary tree**.

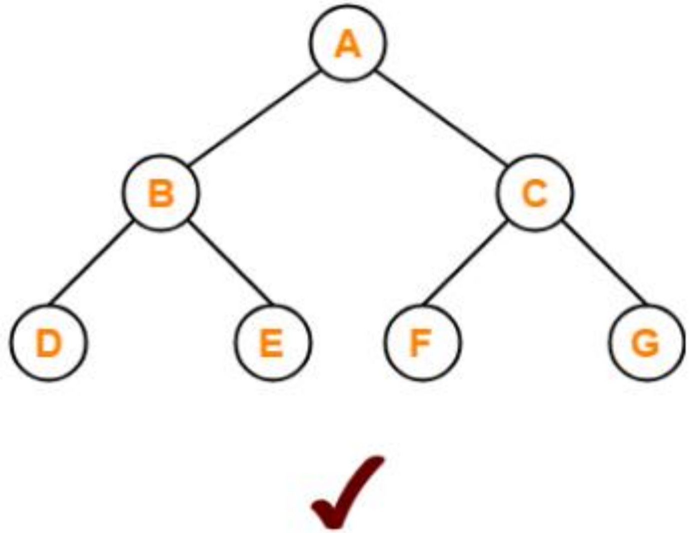




Complete /Perfect Binary Tree

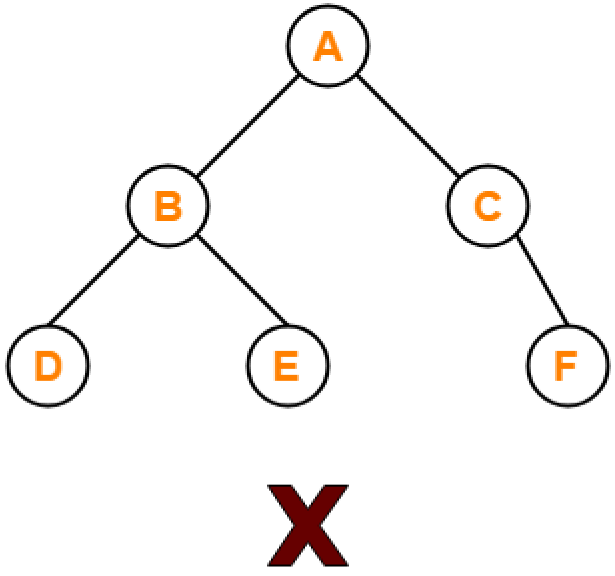
- A **complete binary tree** is a binary tree that satisfies the following 2 properties:
  - Every internal node has exactly 2 children.
  - All the leaf nodes are at the same level.

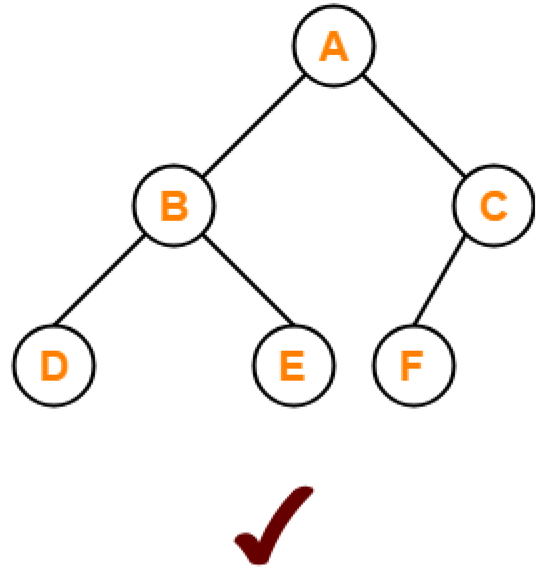




Almost Complete Binary Tree

- 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.



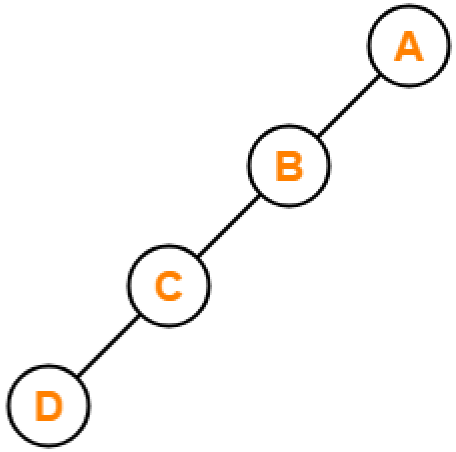


Skewed Binary Tree

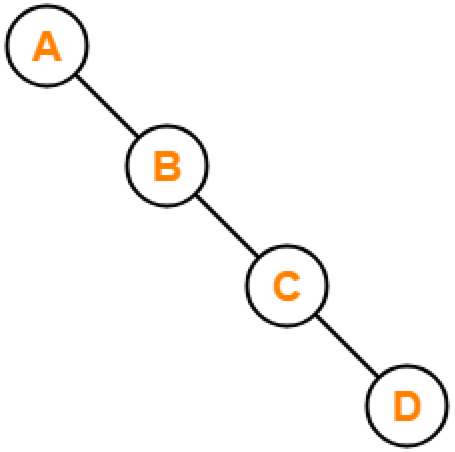
- A **skewed binary tree** is a binary tree that satisfies the following 2 properties-
- All the nodes except one node has one and only one child.
- The remaining node has no child.

OR

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



Left Skewed Binary Tree

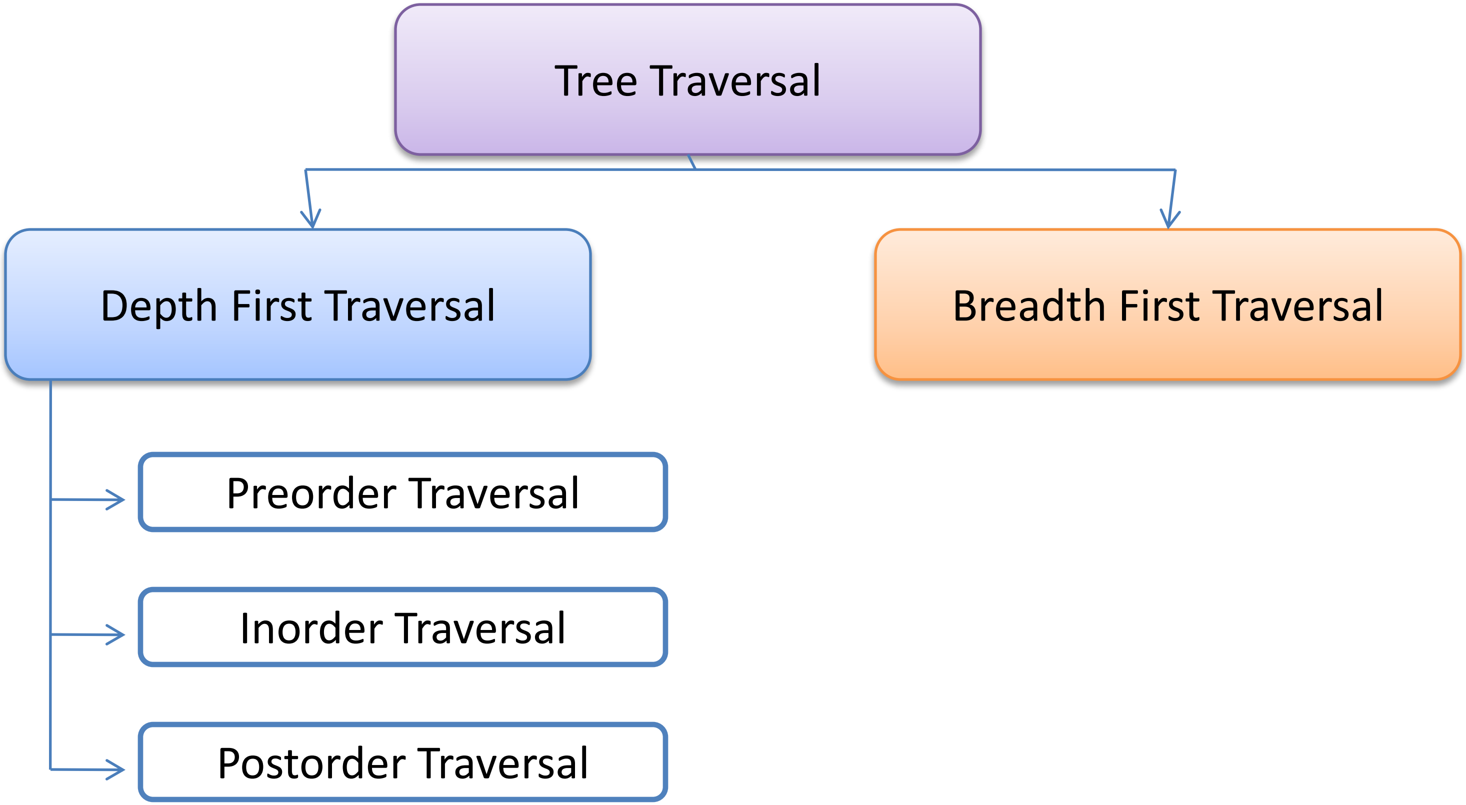


Right Skewed Binary Tree

Tree Traversal

- In order to perform any operation on a tree, you need to reach to the specific node. The tree traversal algorithm helps in visiting a required node in the tree.
- Tree Traversal refers to the process of visiting each node in a tree data structure exactly once.

Tree traversal techniques

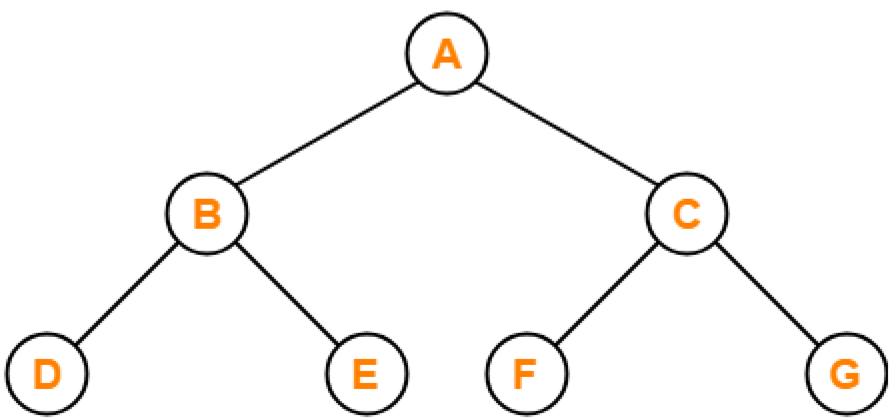


Depth First Traversal

- Following three traversal techniques fall under Depth First Traversal-
  1. Preorder Traversal
  2. Inorder Traversal
  3. Postorder Traversal

Preorder Traversal

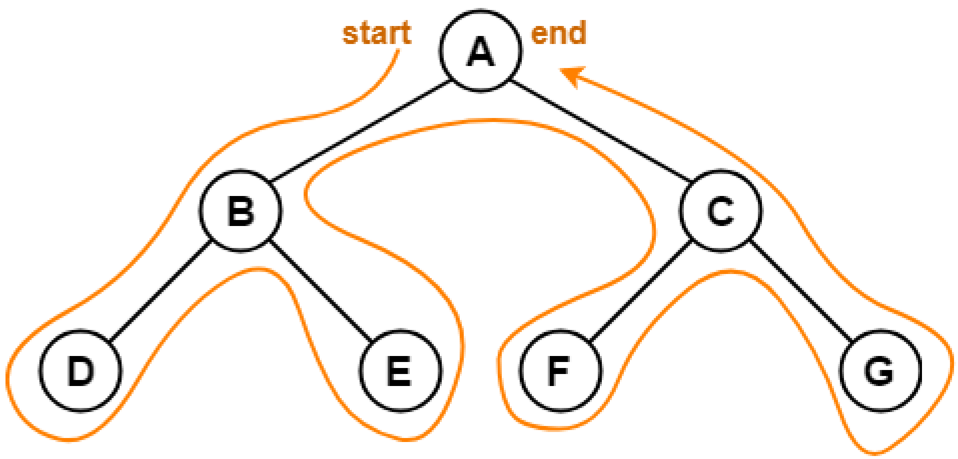
- **<u>Algorithm-</u>**
  - Visit the root
  - Traverse the left sub tree i.e. call Preorder (left sub tree)
  - Traverse the right sub tree i.e. call Preorder (right sub tree)



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

Preorder Traversal Shortcut

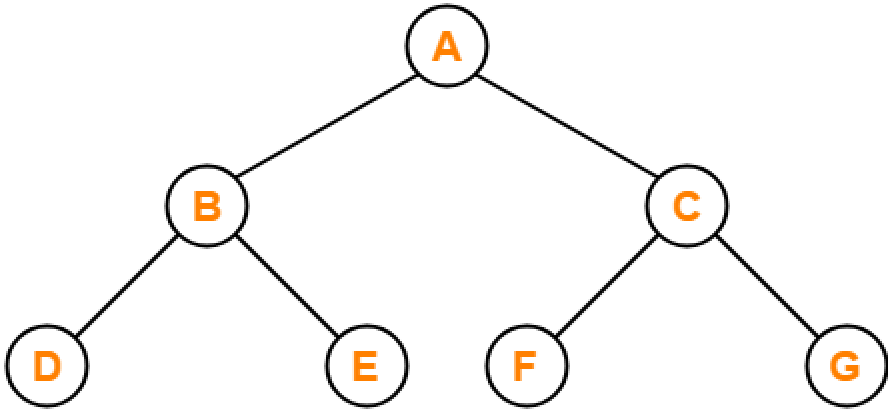
Traverse the entire tree starting from the root node keeping yourself to the left.



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

Inorder Traversal

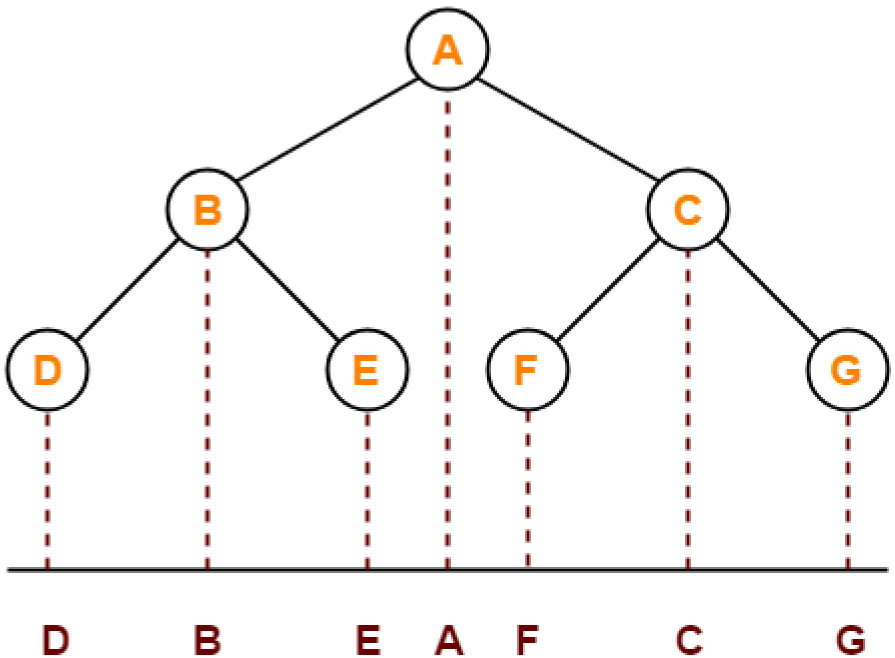
- **Algorithm-**
  - Traverse the left sub tree i.e. call Inorder (left sub tree)
  - Visit the root
  - Traverse the right sub tree i.e. call Inorder (right sub tree)



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

Inorder Traversal Shortcut

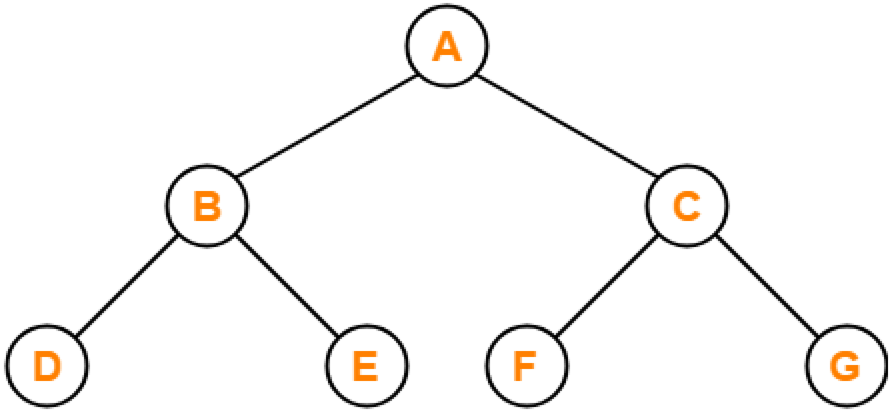
Keep a plane mirror horizontally at the bottom of the tree and take the projection of all the nodes.



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

Postorder Traversal

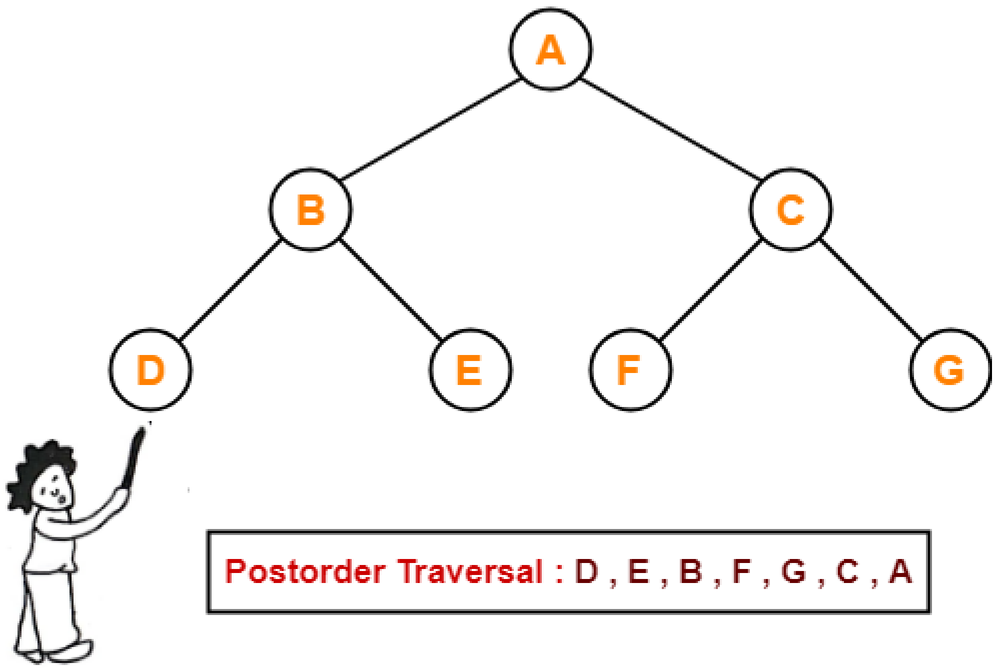
- **Algorithm-**
  - Traverse the left sub tree i.e. call Postorder (left sub tree)
  - Traverse the right sub tree i.e. call Postorder (right sub tree)
  - Visit the root



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

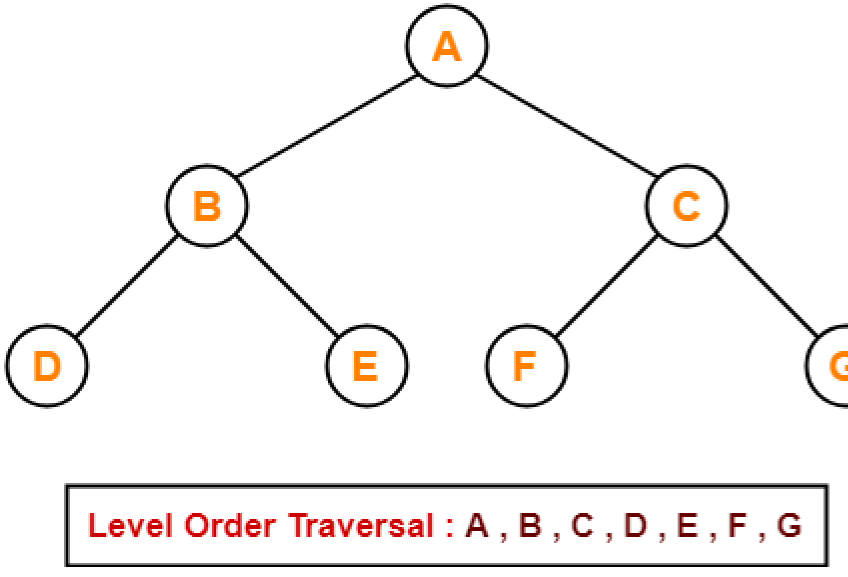
Postorder Traversal Shortcut

Pluck all the leftmost leaf nodes one by one.



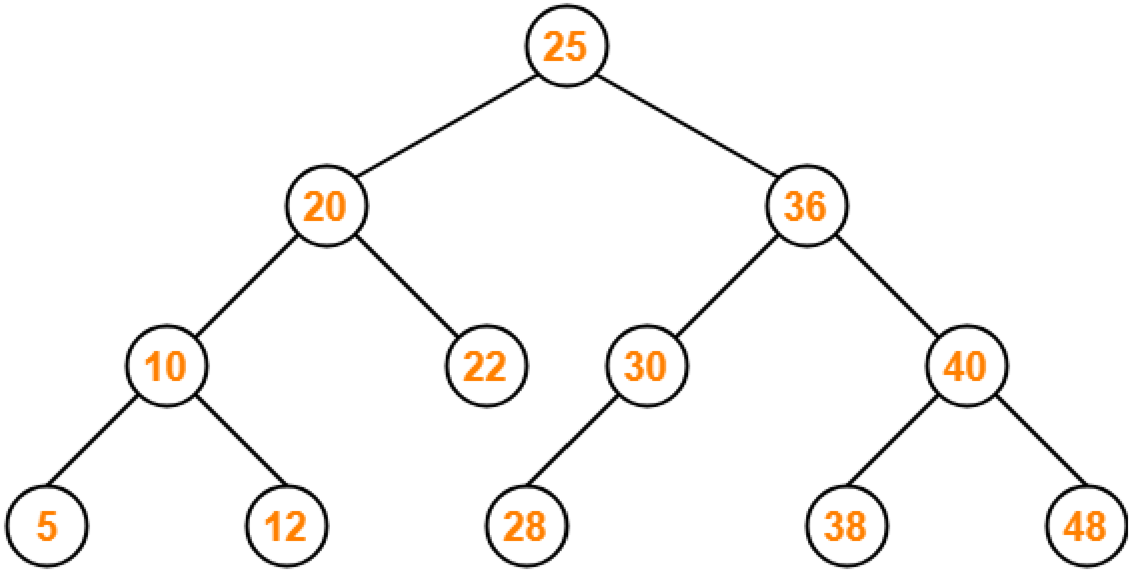
Breadth First Search

- 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**.



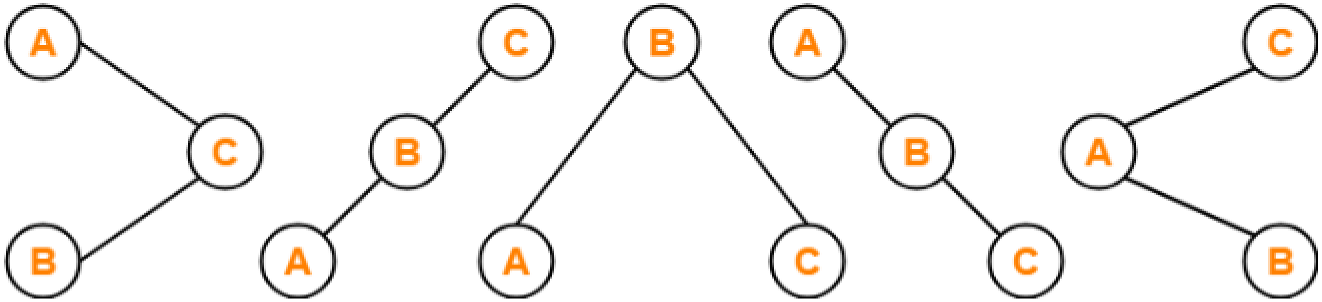
Binary Search Tree construction

- 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



BST construction

- Number of distinct binary search trees possible with 3 distinct Nodes
  $= {}^{2\times3}C_3 / 3+1$
  $= {}^{6}C_3 / 4$
  $= 5$
- If three distinct Nodes are A, B and C, then 5 distinct binary search trees are:



Example

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

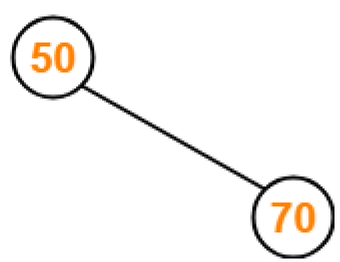
50, 70, 60, 20, 90, 10, 40, 100

- 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.

Insert 50-

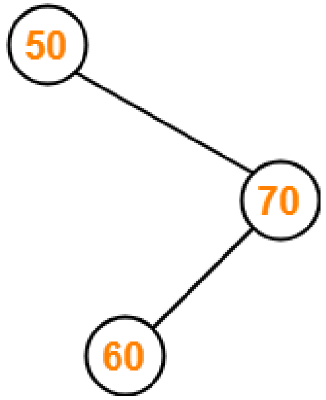
Insert 70-

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



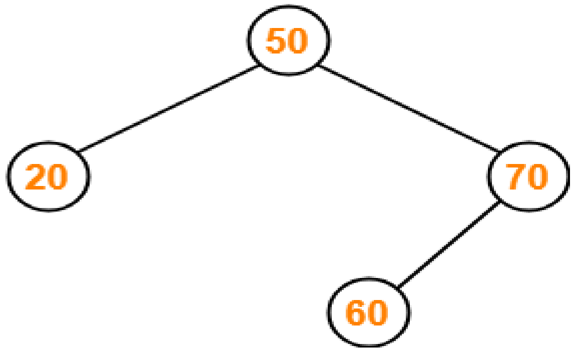
Insert 60-

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



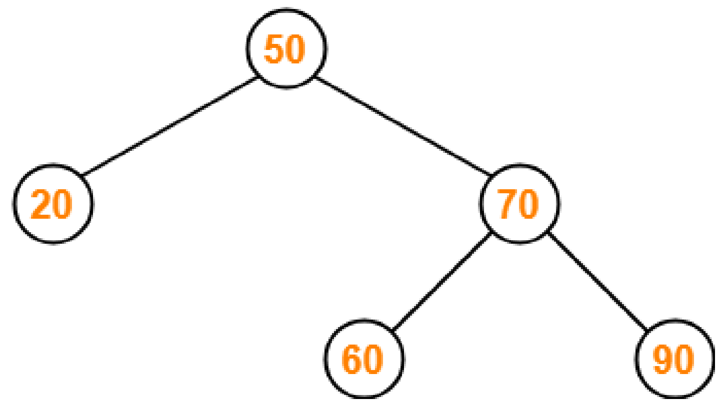
Insert 20-

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



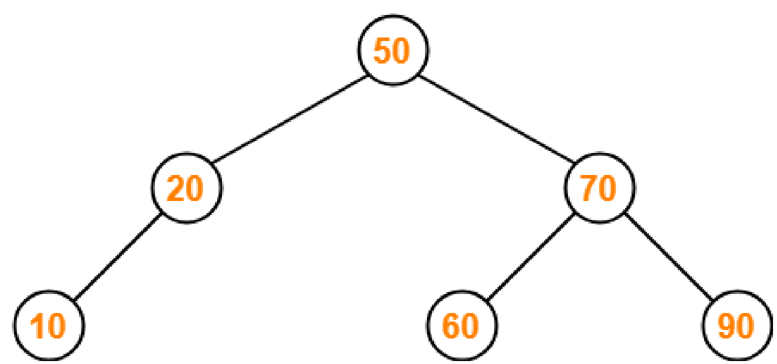
Insert 90-

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



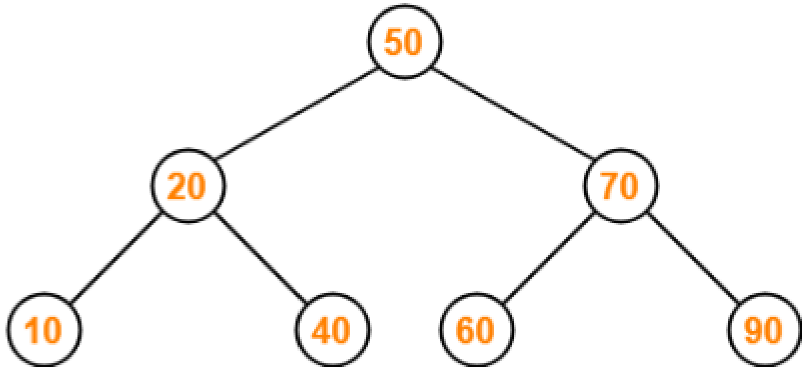
Insert 10-

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



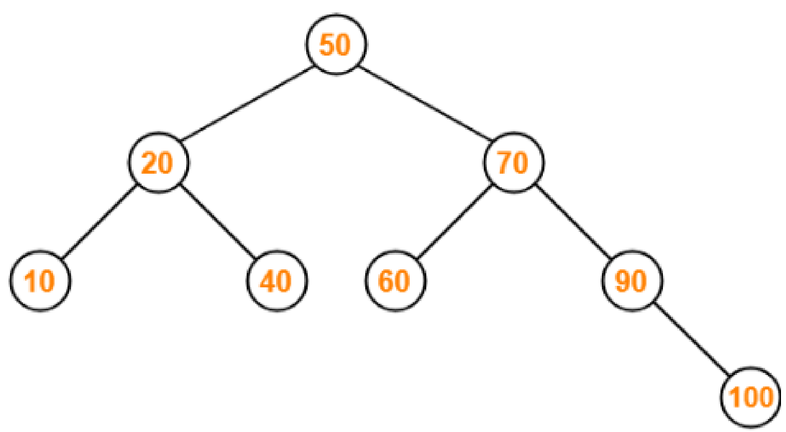
Insert 40-

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

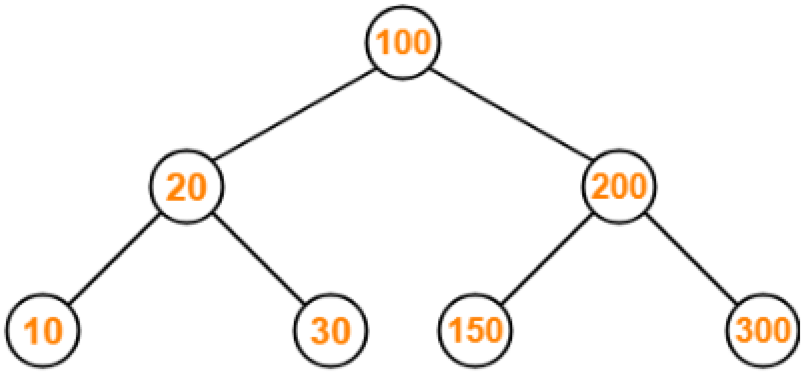


Insert 100-

- 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.



BST Traversal



Preorder Traversal-

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

Inorder Traversal-

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

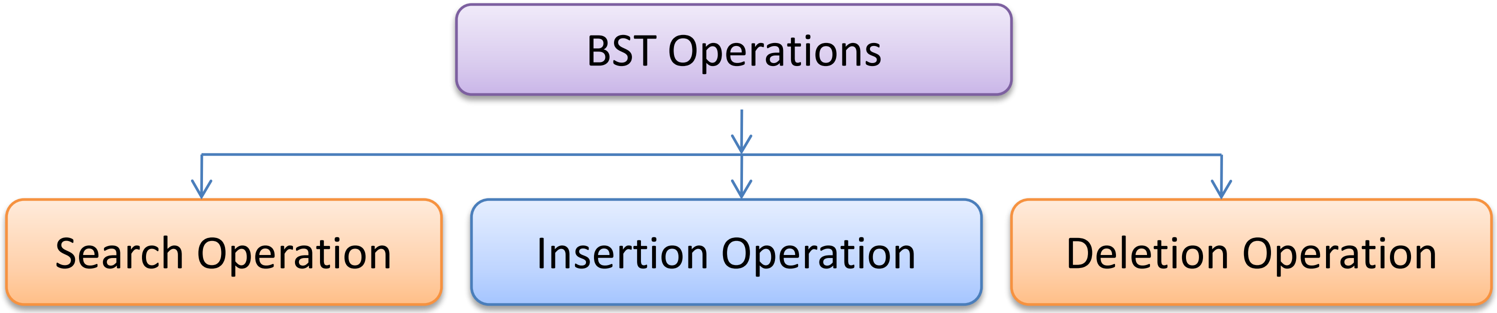
Postorder Traversal-

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

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

BST Operations

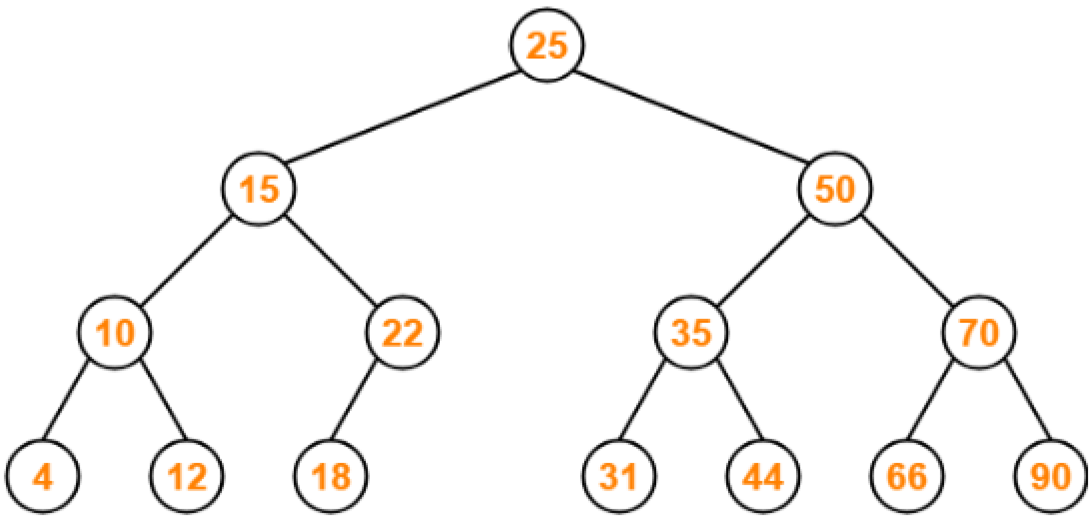
- Commonly performed binary search tree operations are:



Search Operation

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

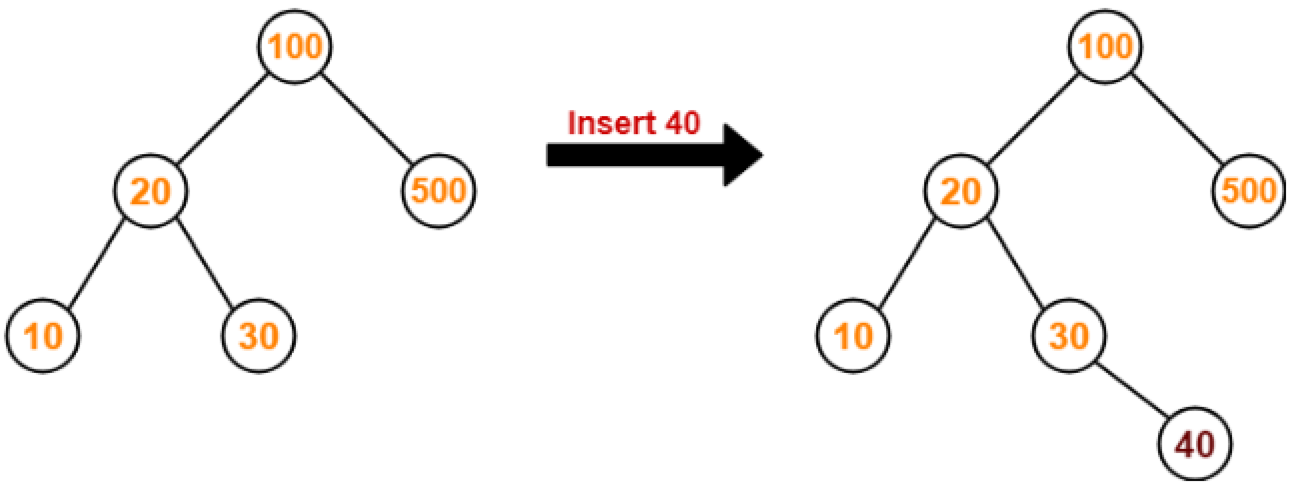
Consider key = 45 has to be searched in the given BST-



- 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.

Insertion operation

- 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.



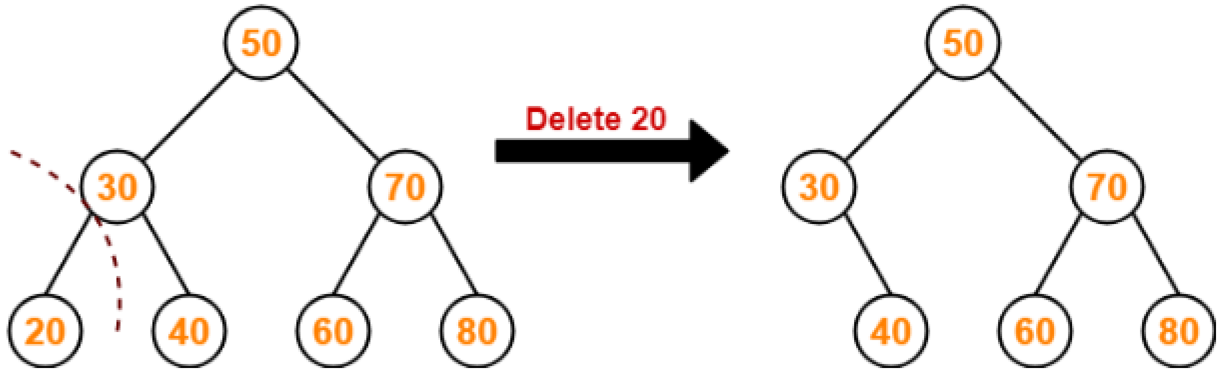
- We start searching for value 40 from the root node 100.
- As 40 < 100, so we search in 100's left subtree.
- As 40 > 20, so we search in 20's right subtree.
- As 40 > 30, so we add 40 to 30's right subtree.

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, three cases are possible.

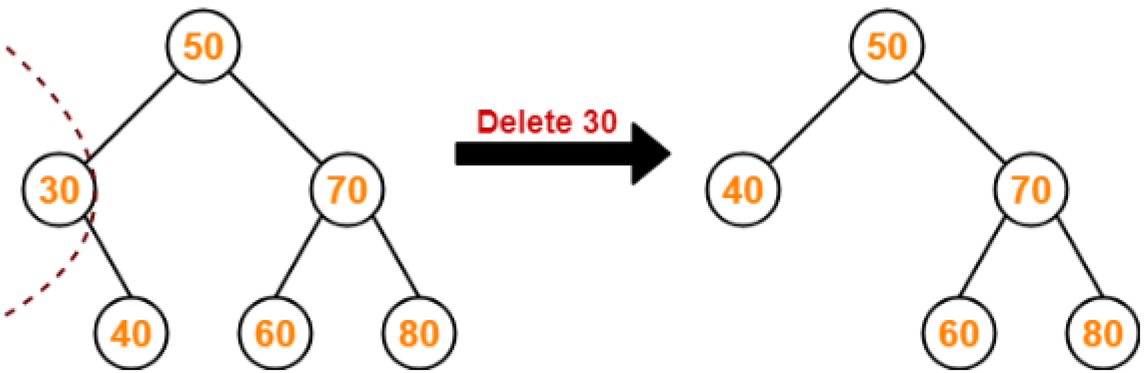
Case-01: Deletion Of A Node Having No Child (Leaf Node)

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



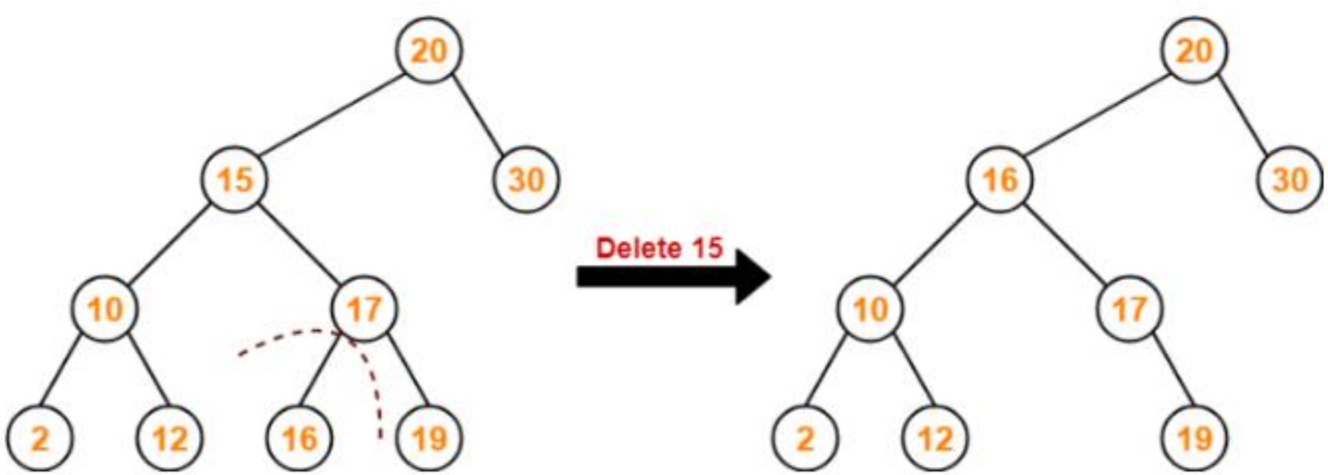
Case-02: Deletion Of A Node Having Only One Child

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



Case-03: Deletion Of A Node Having Two Children

- Consider the following example where node with value = 15 is deleted from the BST
- Method-1:
  - 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.



- Method-2:
  - Visit to the left subtree of the deleting node.
  - Pluck the greatest value element called as inorder successor.
  - Replace the deleting element with its inorder successor.