Tree Terminology

Tree 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.

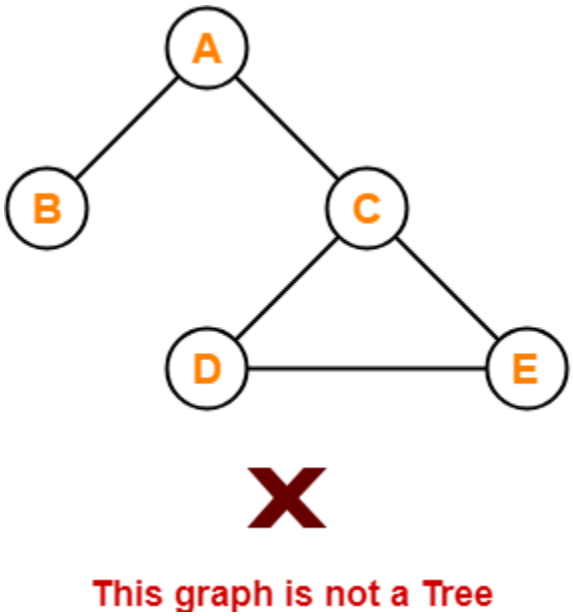
OR

A tree is a connected graph without any circuits.

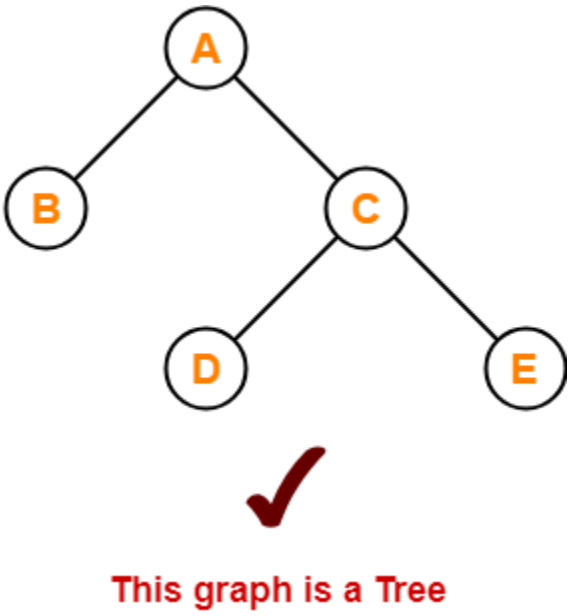
OR

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

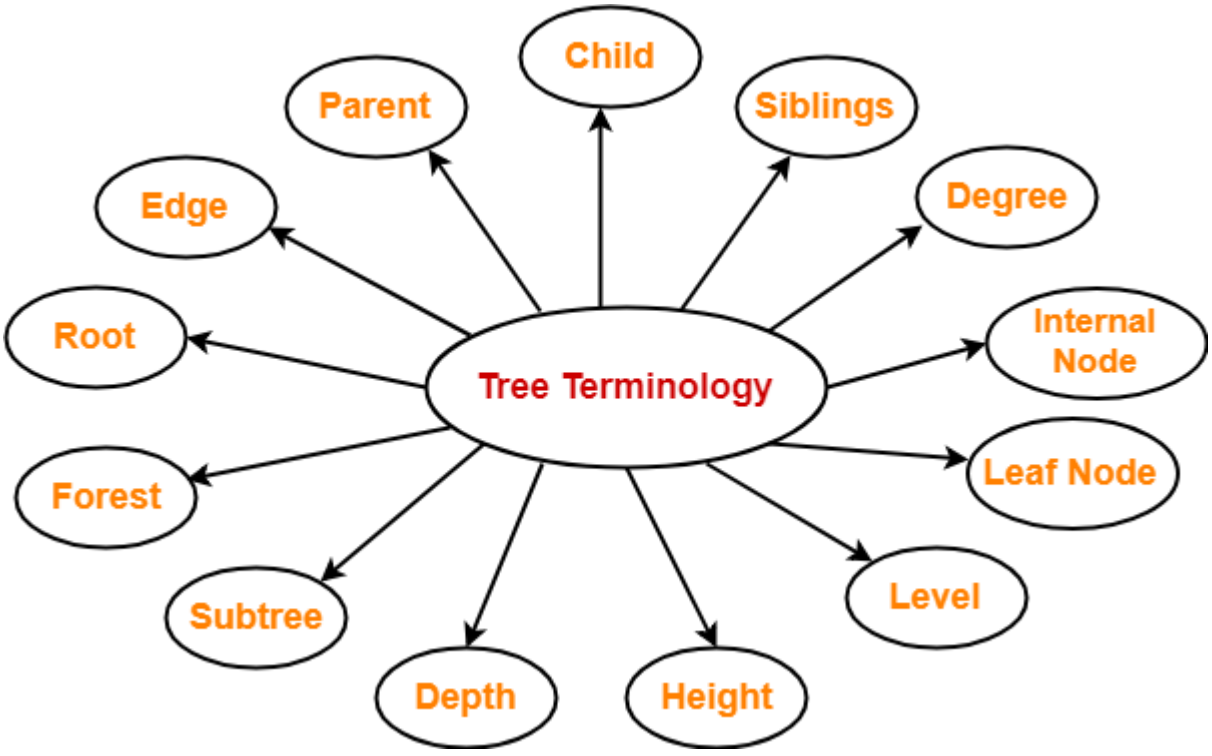
Properties-

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-

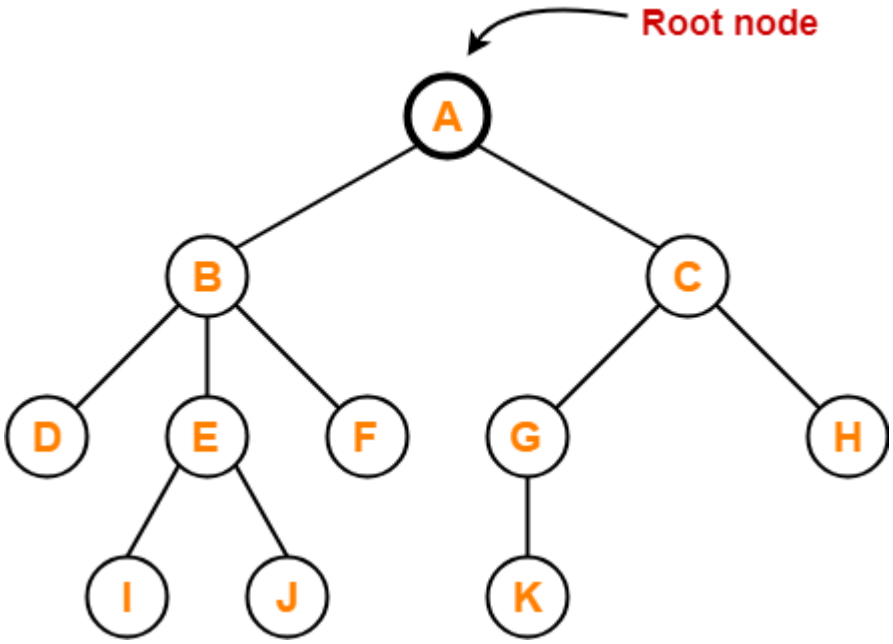
The important terms related to tree data structure are-



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-

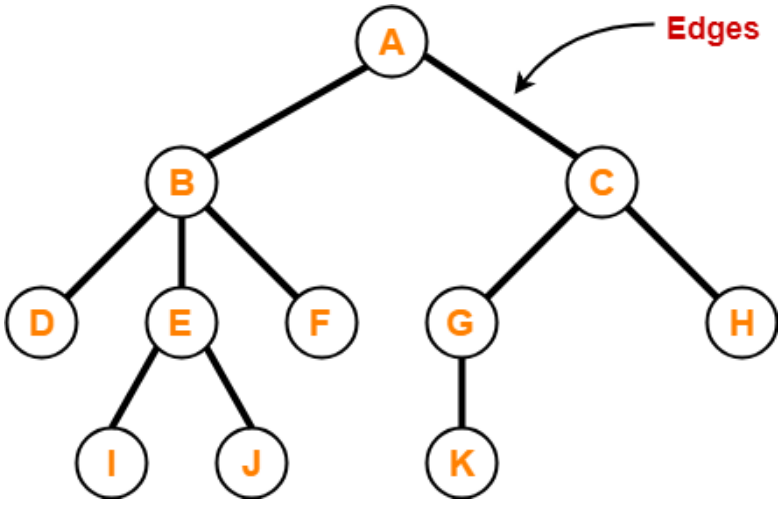


Here, node A is the only root node.

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.

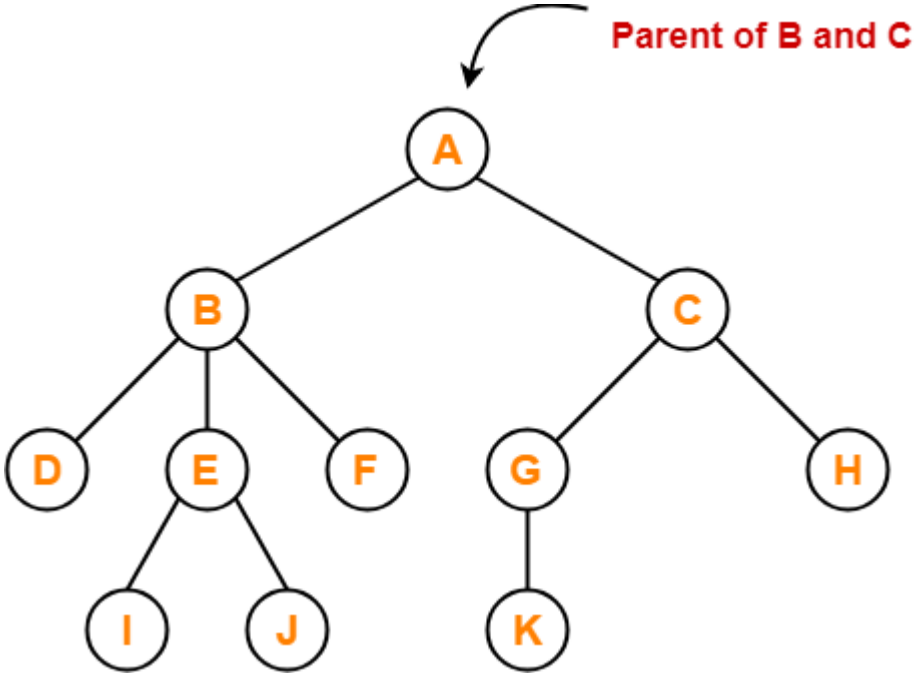
Example-



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-



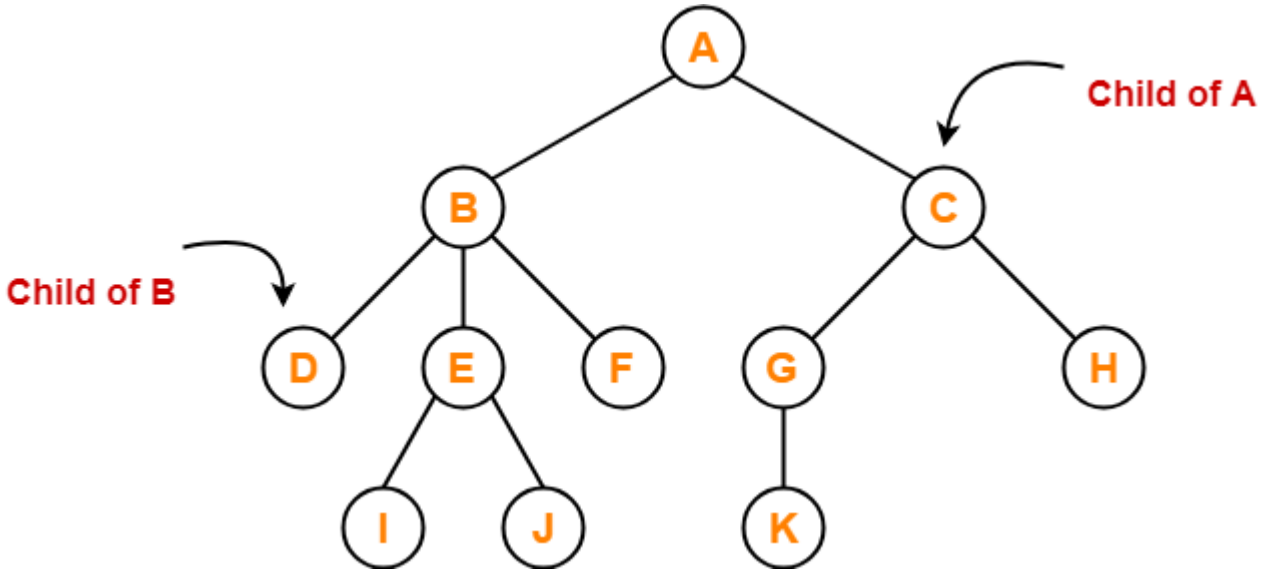
Here,

- 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-



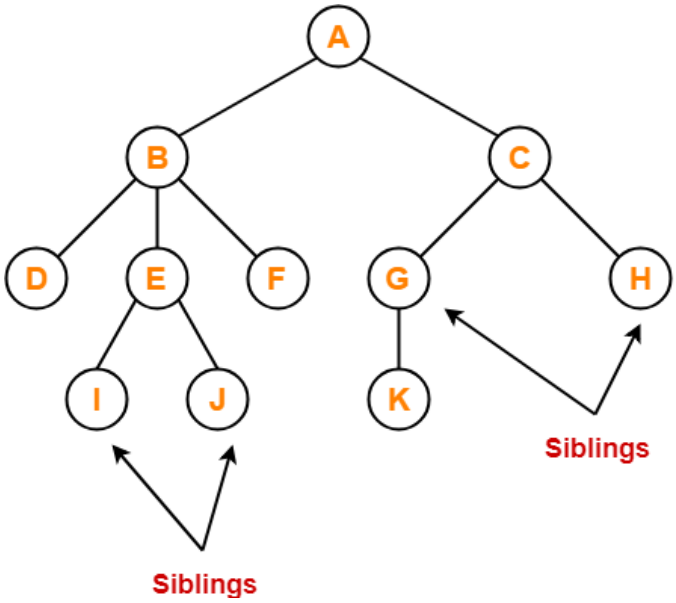
Here,

- 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-



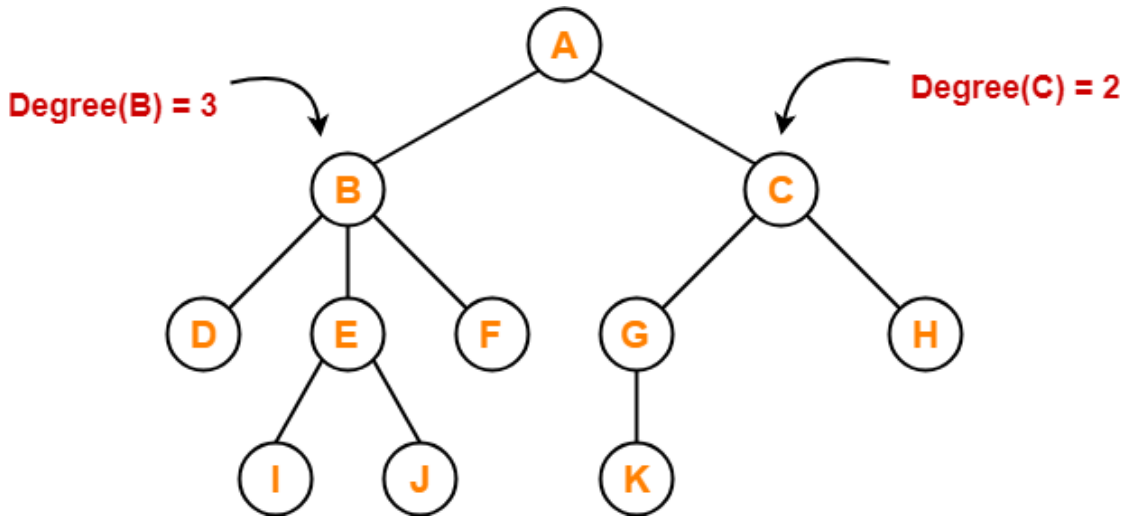
Here,

- 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-



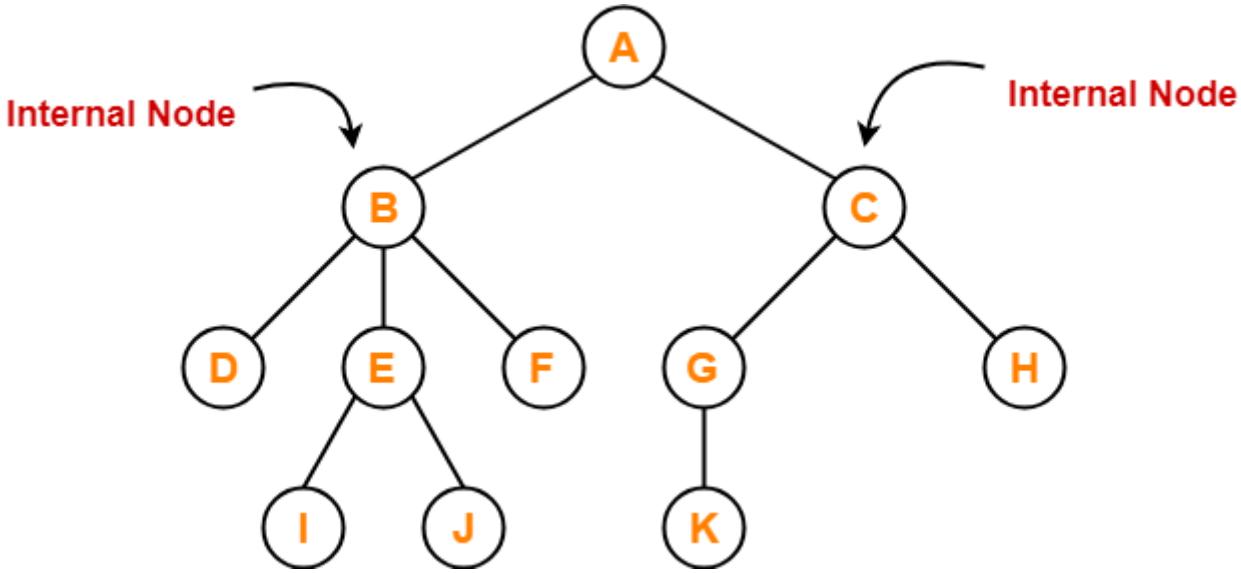
Here,

- 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-

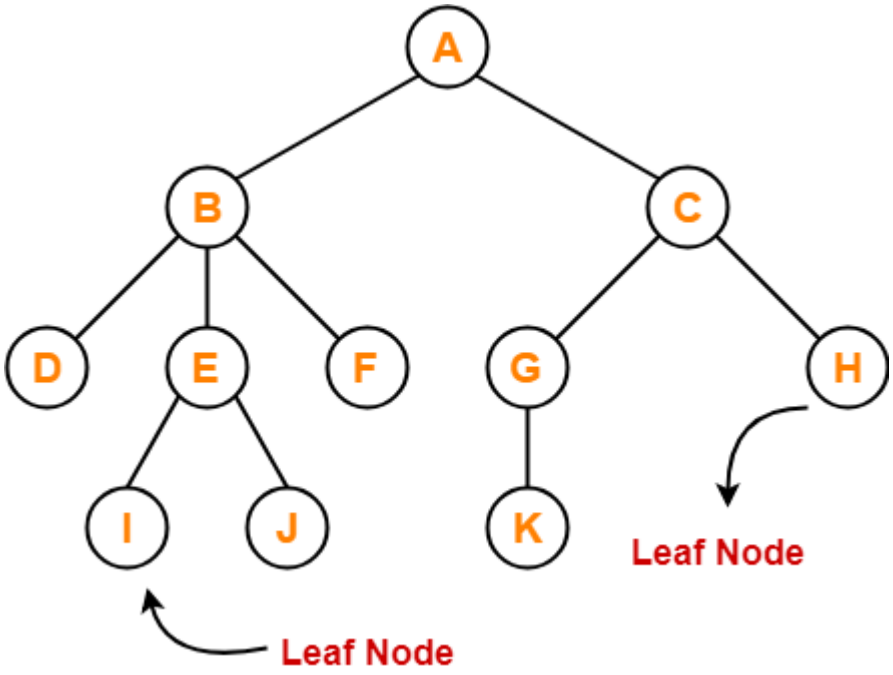


Here, nodes A, B, C, E and G are internal nodes.

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-

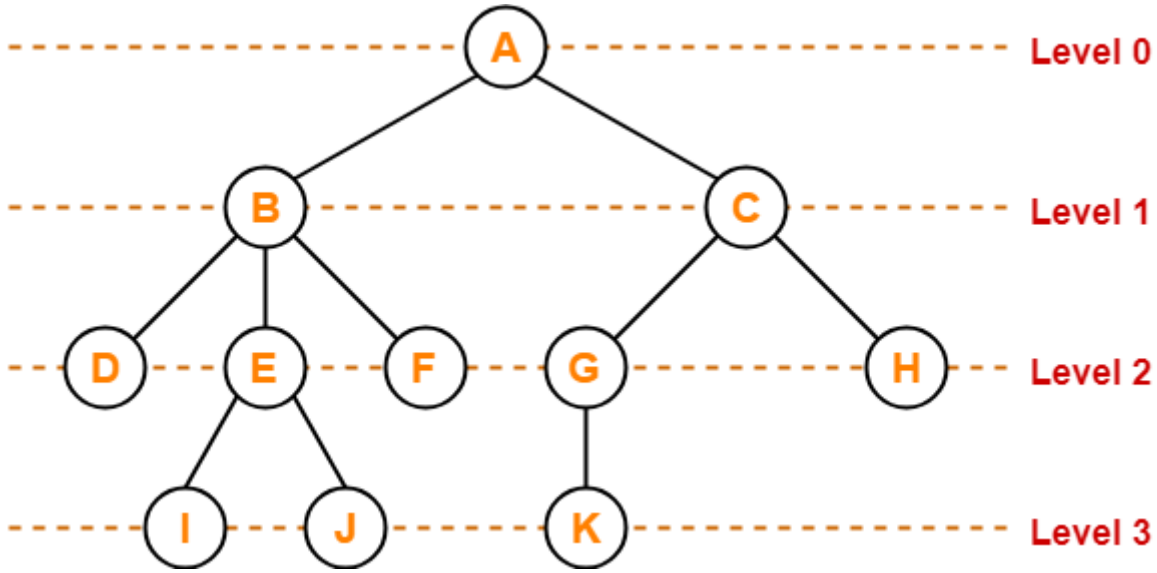


Here, nodes D, I, J, F, K and H are leaf nodes.

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.

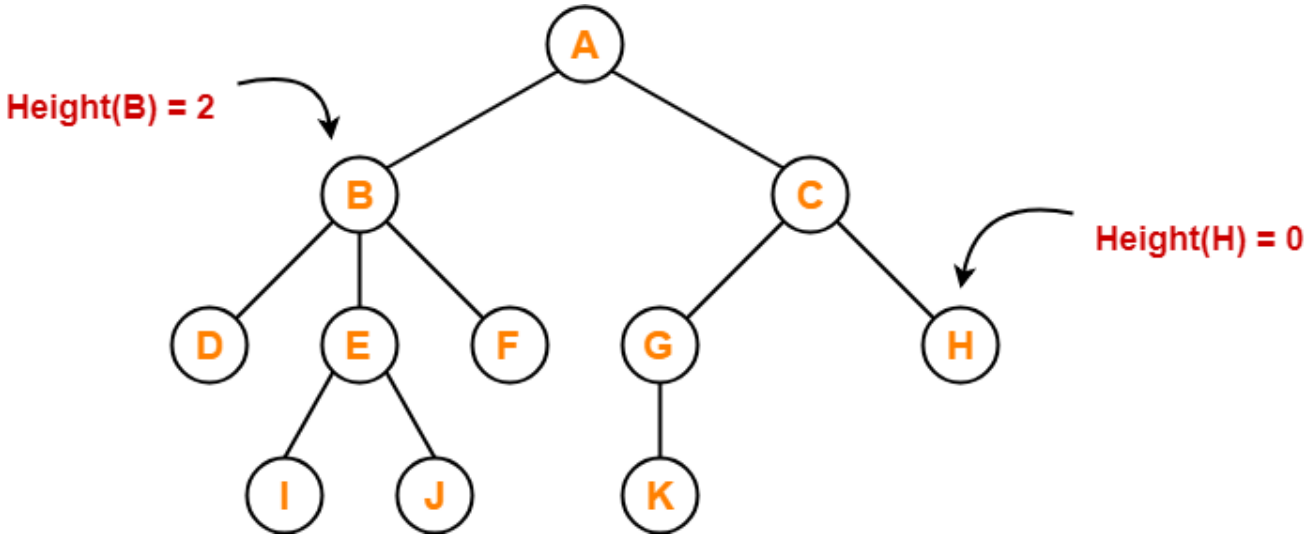
Example-



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,

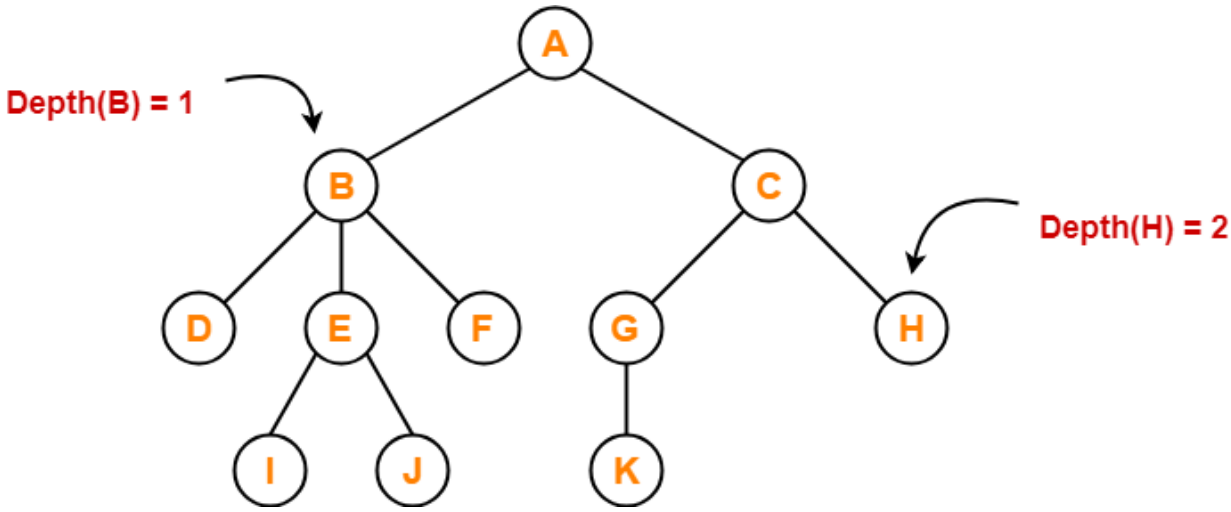
- Height of node A = 3
- Height of node B = 2

- 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,

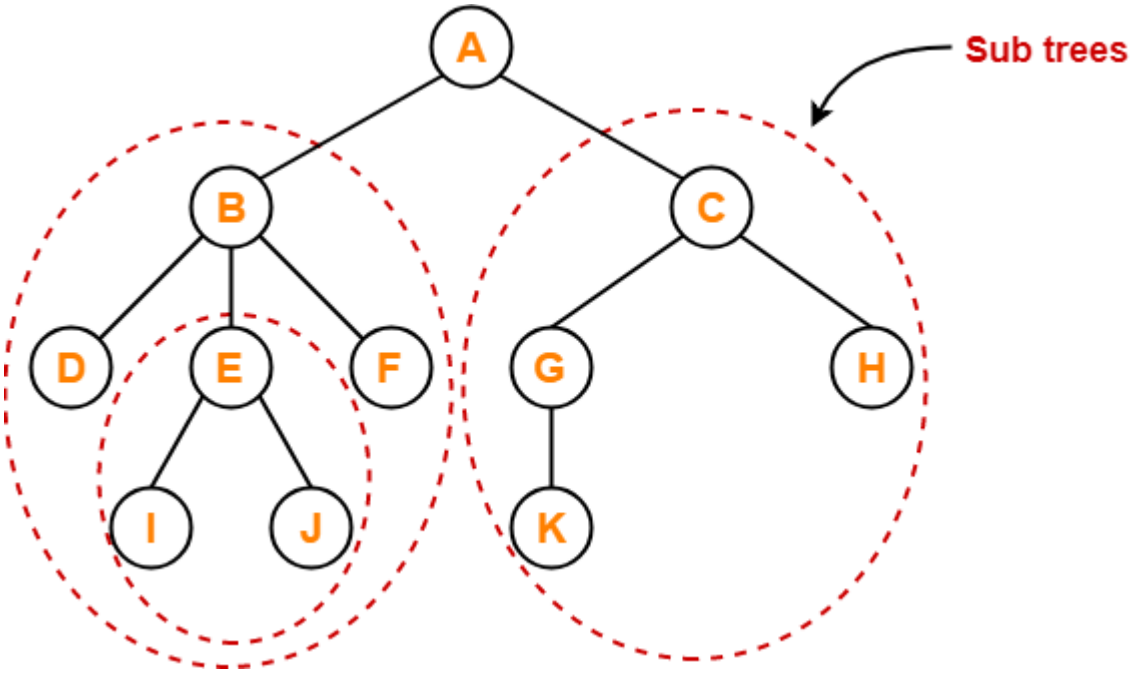
- Depth of node A = 0
- Depth of node B = 1
- Depth of node C = 1
- Depth of node D = 2
- Depth of node E = 2

- 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.

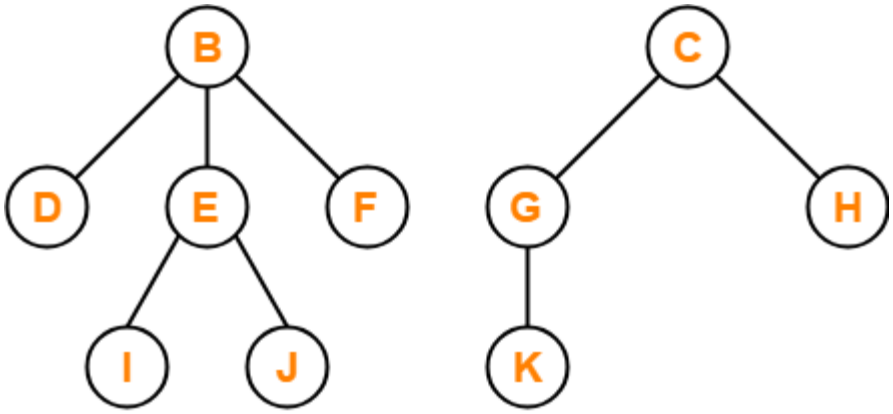
Example-



13. Forest-

A forest is a set of disjoint trees.

Example-



Forest

Binary Tree | Types of Binary Trees

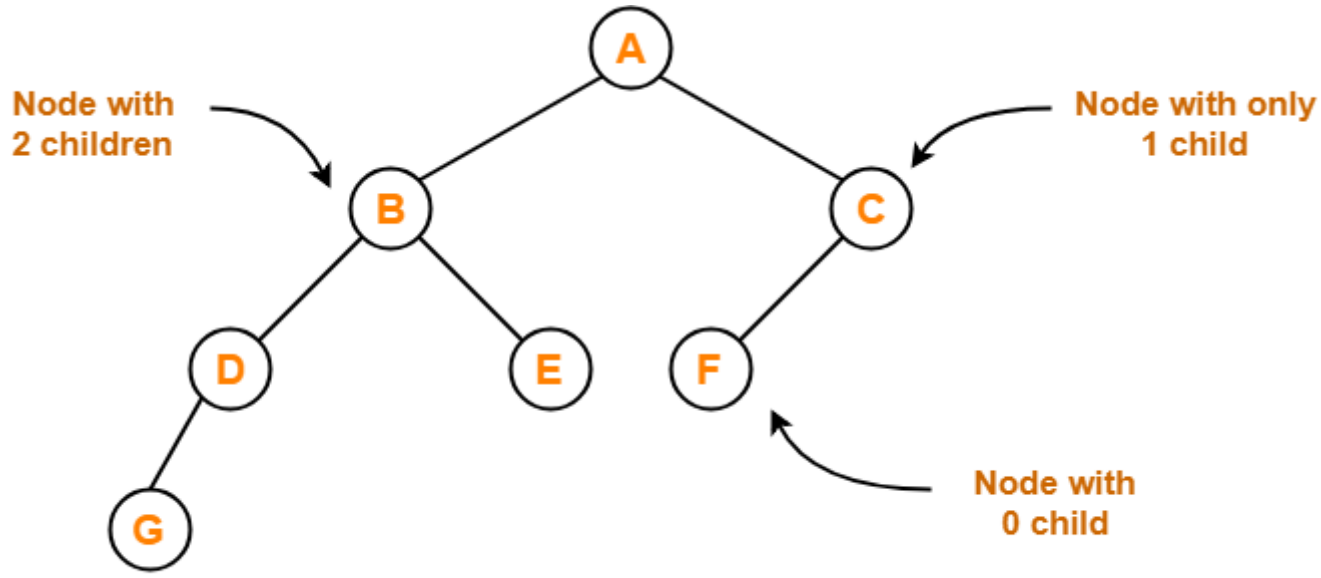
Binary Tree-

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.

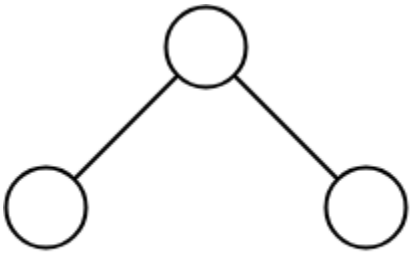
Example-



Binary Tree Example

Unlabeled Binary Tree-

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



Unlabeled Binary Tree

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

Example-

Consider we want to draw all the binary trees possible with 3 unlabeled nodes.

Using the above formula, we have-

Number of binary trees possible with 3 unlabeled nodes

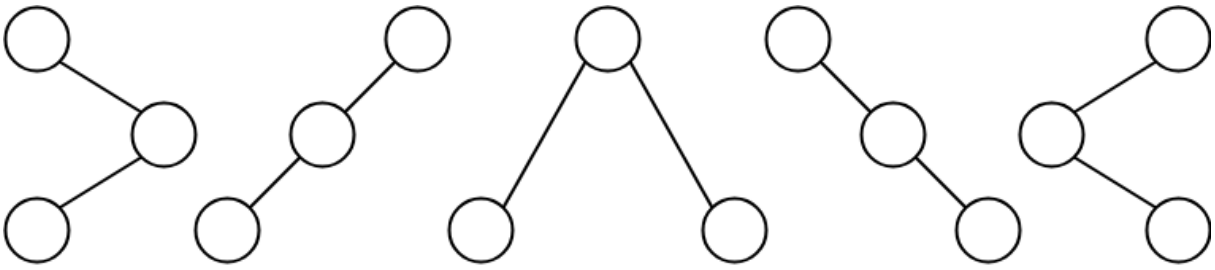
$= {}^{2 \times 3}C_3 / (3 + 1)$

$= {}^{6}C_3 / 4$

$= 5$

Thus,

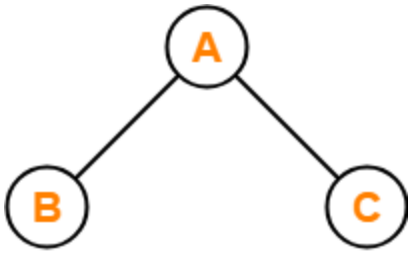
- With 3 unlabeled nodes, 5 unlabeled binary trees are possible.
- These unlabeled binary trees are as follows-



Binary Trees Possible With 3 Unlabeled Nodes

Labeled Binary Tree-

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



Labeled Binary Tree

Number of different Binary Trees possible with 'n' labeled nodes

$$= \frac{{}^{2n}C_n}{n+1} \times n!$$

Example-

Consider we want to draw all the binary trees possible with 3 labeled nodes.

Using the above formula, we have-

Number of binary trees possible with 3 labeled nodes

$= \{ {}^{2 \times 3}C_3 / (3 + 1) \} \times 3!$

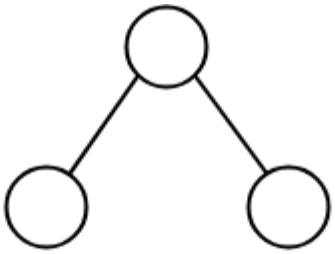
$= \{ {}^{6}C_3 / 4 \} \times 6$

$= 5 \times 6$

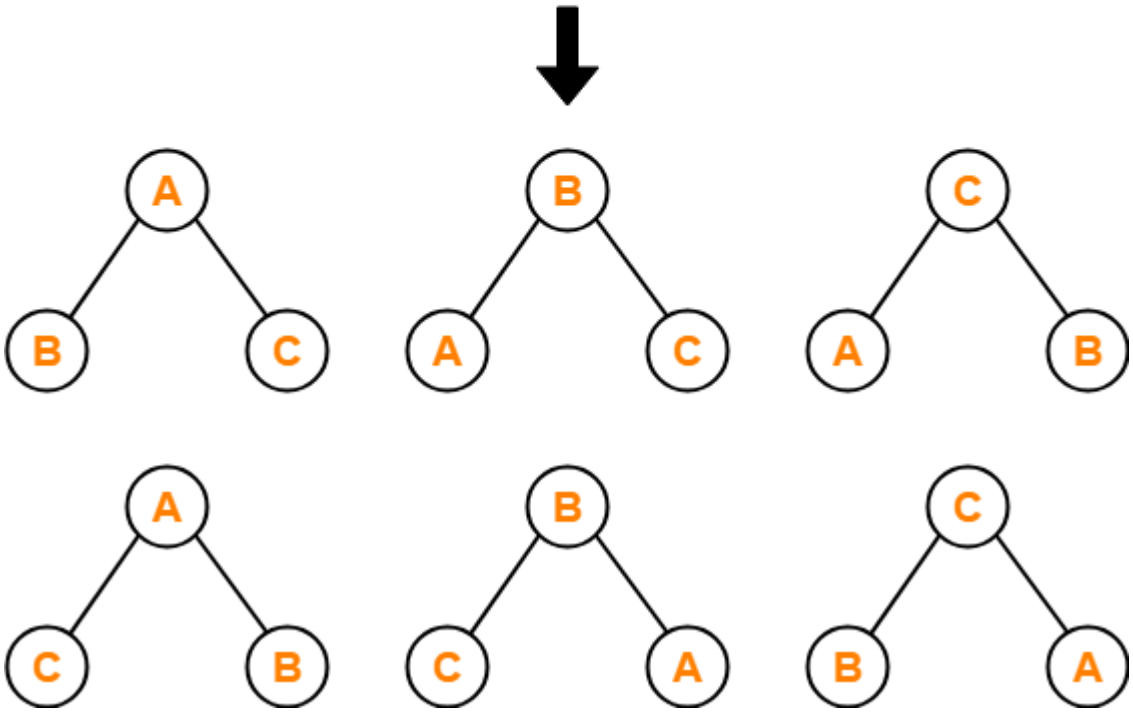
$= 30$

Thus,

- With 3 labeled nodes, 30 labeled binary trees are possible.
- Each unlabeled structure gives rise to 3! = 6 different labeled structures.



It Gives Rise to Following 6 Labeled Structures

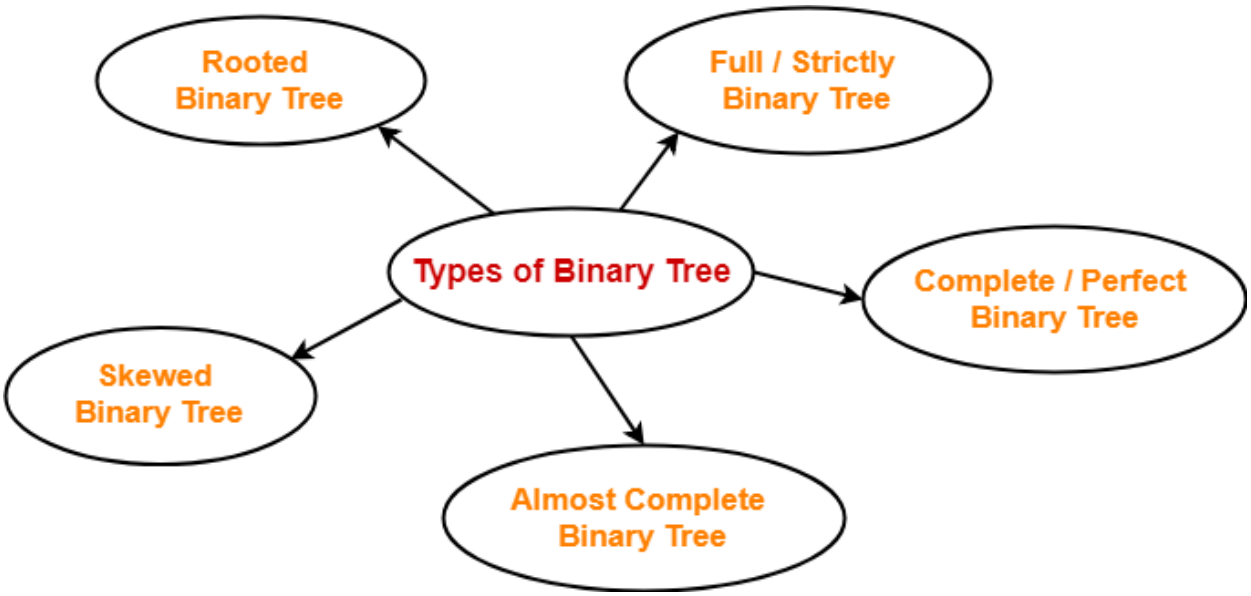


Similarly,

- Every other unlabeled structure gives rise to 6 different labeled structures.
- Thus, in total 30 different labeled binary trees are possible.

Types of Binary Trees-

Binary trees can be of the following types-



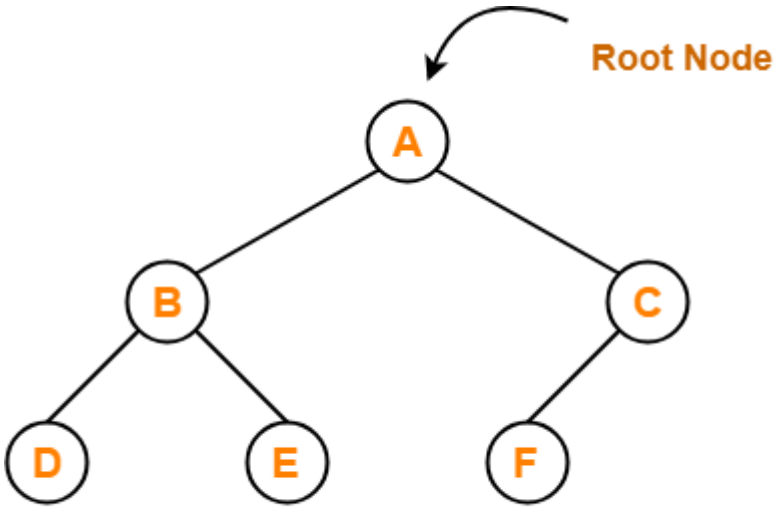
1. Rooted Binary Tree
2. Full / Strictly Binary Tree
3. Complete / Perfect Binary Tree
4. Almost Complete Binary Tree
5. Skewed Binary Tree

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-

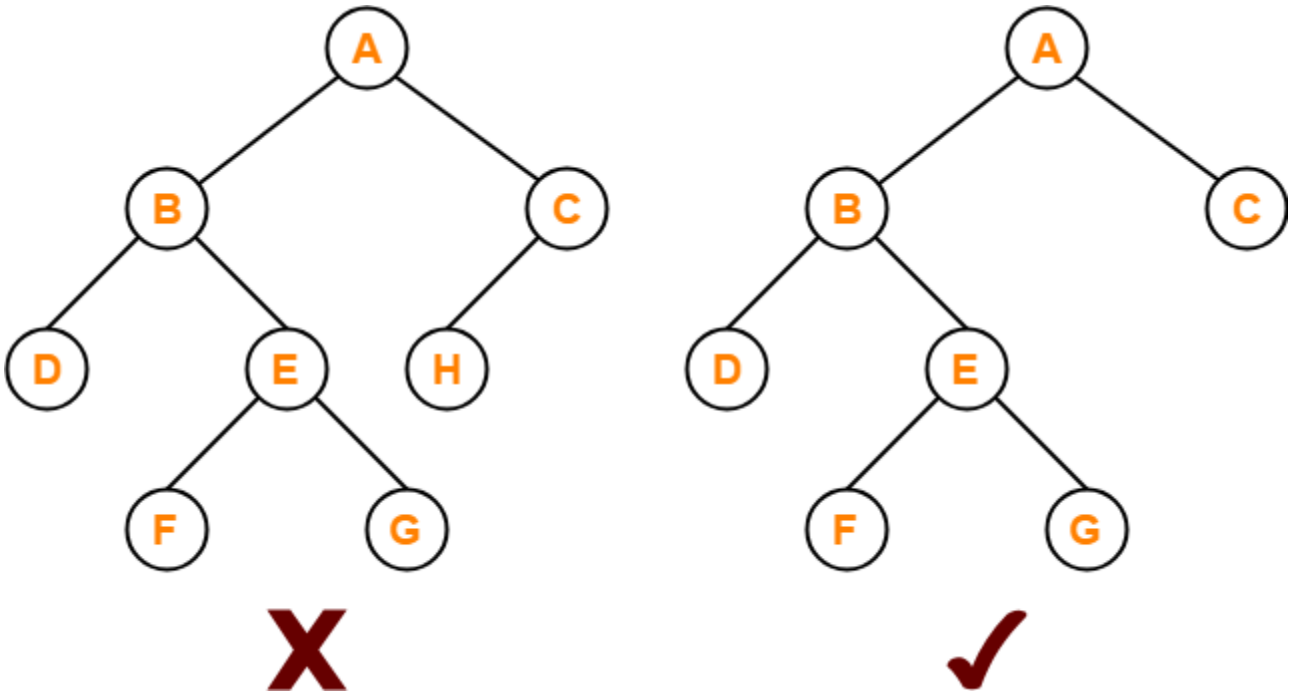


Rooted Binary Tree

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.

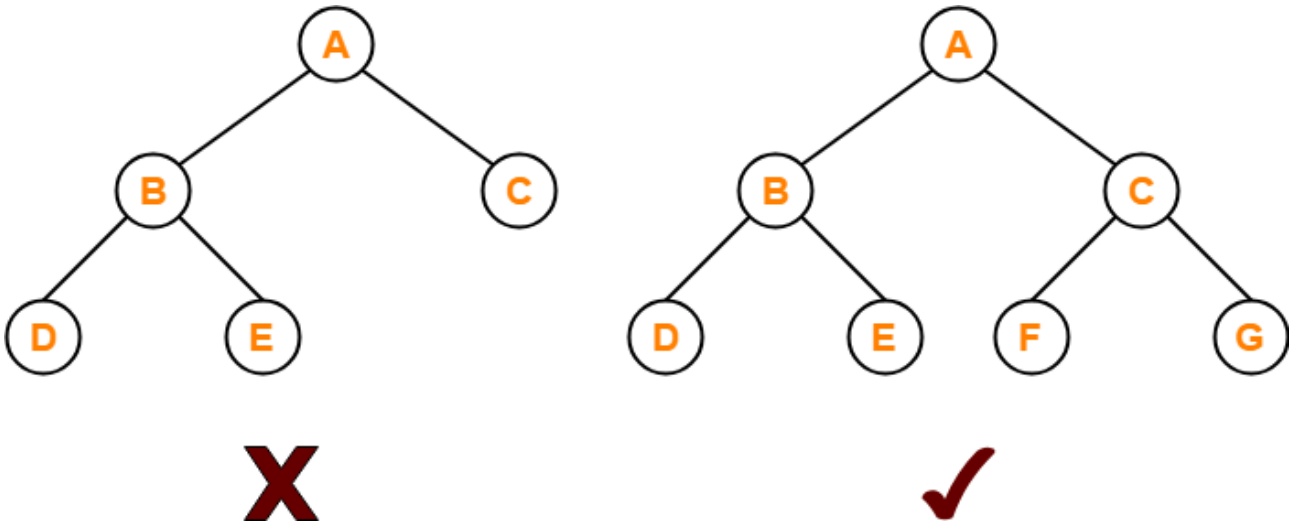
3. Complete / Perfect Binary Tree-

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,

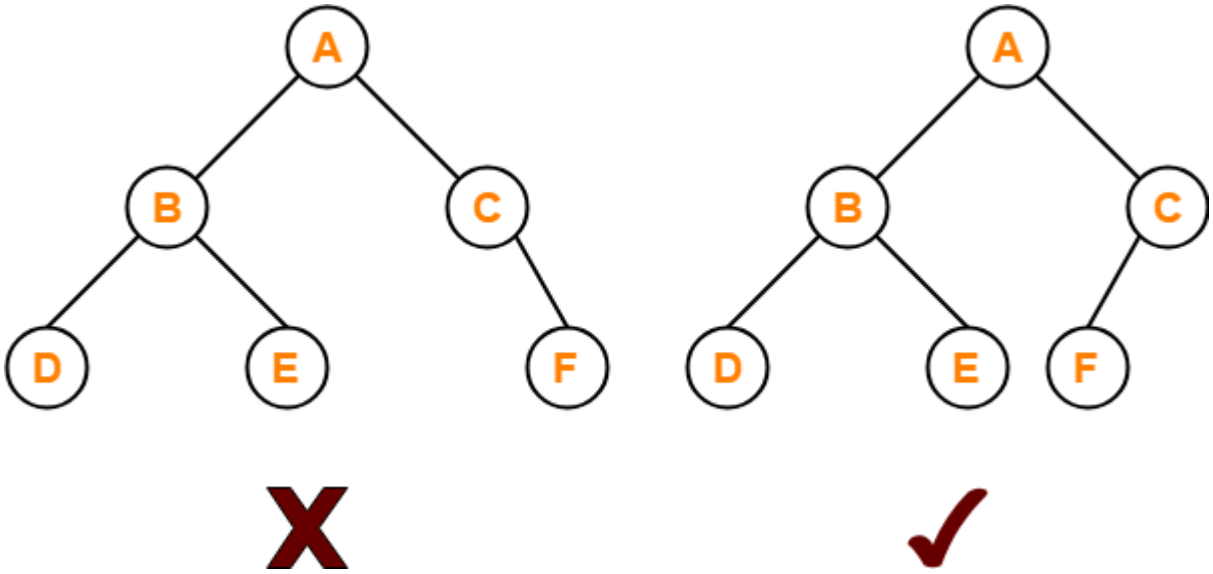
- First binary tree is not a complete binary tree.
- This is because all the leaf nodes are not at the same level.

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-

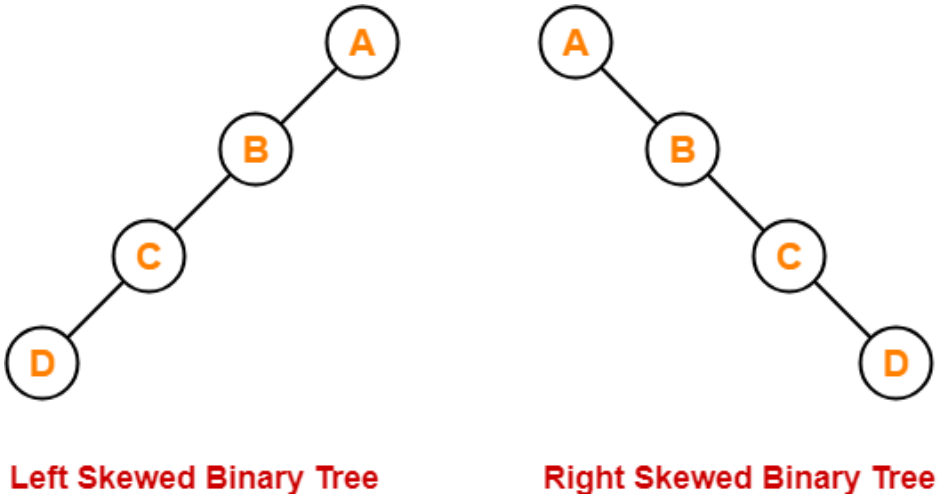
A **skewed binary tree** is a binary tree that satisfies the following 2 properties-

- 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:

Important properties of binary trees are-

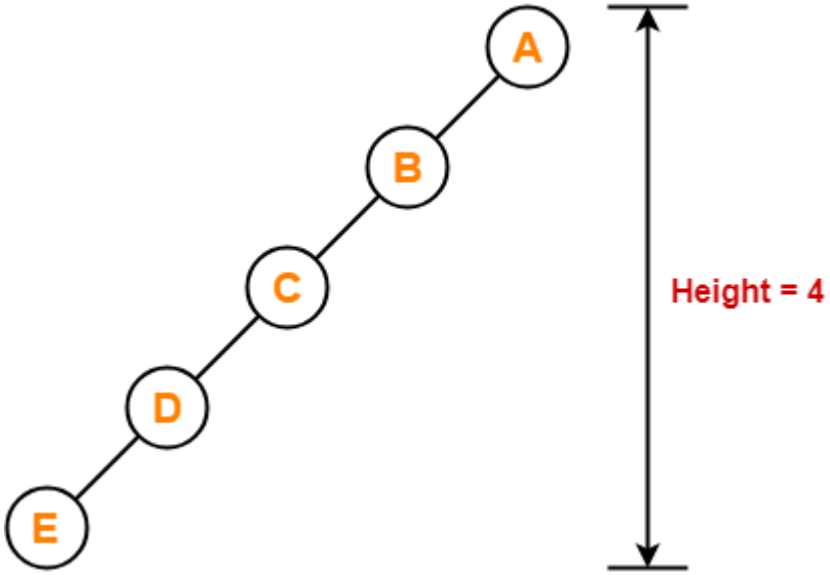
Property-01:

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-

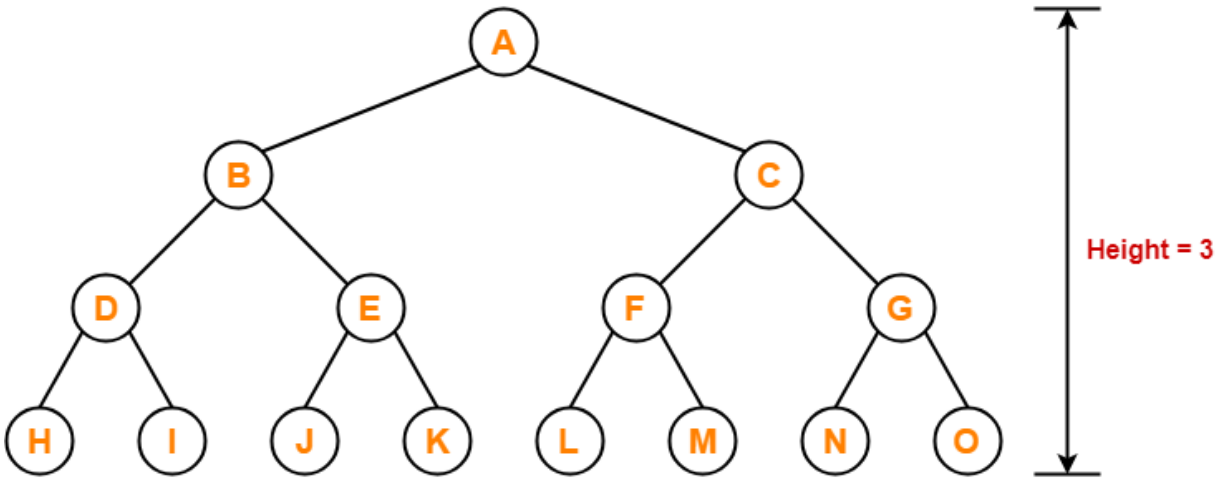
Maximum number of nodes in a binary tree of height 3

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

$= 16 - 1$

$= 15$ nodes

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



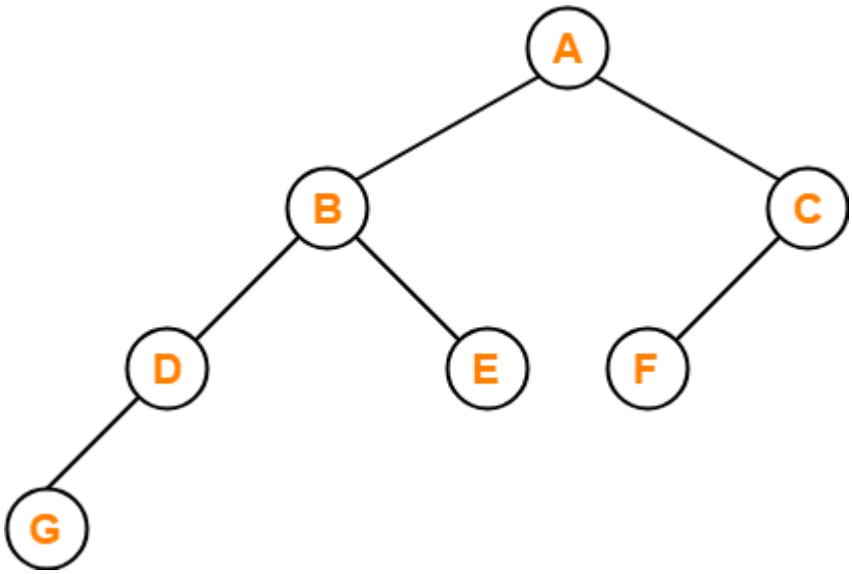
We can not insert more number of nodes in this binary tree.

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

$= 2^L$

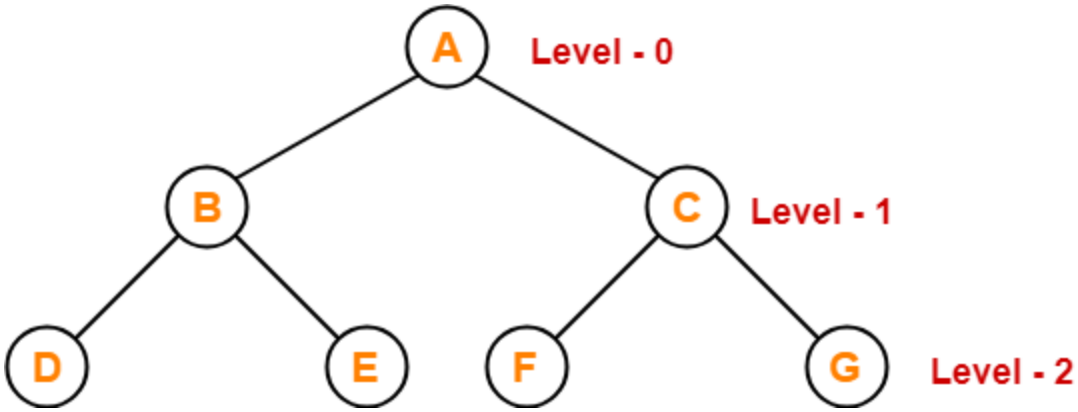
Example-

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.



PRACTICE PROBLEMS BASED ON BINARY TREE PROPERTIES-

Problem-01:

A binary tree T has n leaf nodes. The number of nodes of degree-2 in T is _____?

1. $\log_2 n$
2. n-1
3. n
4. 2^n

Solution-

Using property-3, we have-

Number of degree-2 nodes

= Number of leaf nodes – 1

= n – 1

Thus, Option (B) is correct.

Problem-02:

In a binary tree, for every node the difference between the number of nodes in the left and right subtrees is at most 2. If the height of the tree is h > 0, then the minimum number of nodes in the tree is _____?

1. 2^{h-1}
2. $2^{h-1} + 1$
3. $2^{h} – 1$
4. 2^{h}

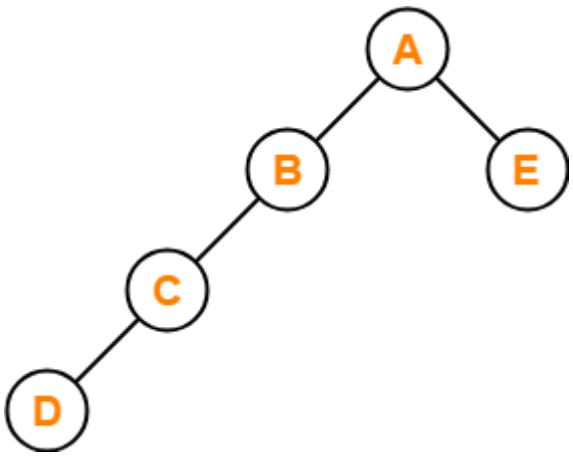
Solution-

Let us assume any random value of h. Let h = 3.

Then the given options reduce to-

1. 4
2. 5
3. 7
4. 8

Now, consider the following binary tree with height h = 3-



- This binary tree satisfies the question constraints.
- It is constructed using minimum number of nodes.

Thus, Option (B) is correct.

Problem-03:

In a binary tree, the number of internal nodes of degree-1 is 5 and the number of internal nodes of degree-2 is 10. The number of leaf nodes in the binary tree is _____?

1. 10
2. 11
3. 12
4. 15

Solution-

Using property-3, we have-

Number of leaf nodes in a binary tree

= Number of degree-2 nodes + 1

= 10 + 1

= 11

Thus, Option (B) is correct.

Problem-04:

The height of a binary tree is the maximum number of edges in any root to leaf path. The maximum number of nodes in a binary tree of height h is _____?

1. 2^h
2. $2^{h-1} – 1$
3. $2^{h+1} – 1$
4. 2^{h+1}

Solution-

Using property-2, Option (C) is correct.

Problem-05:

A binary tree T has 20 leaves. The number of nodes in T having 2 children is _____?

Solution-

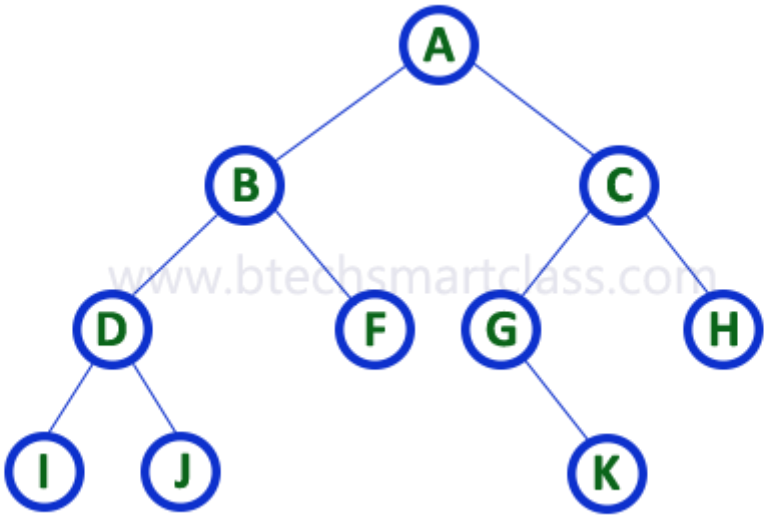
Using property-3, correct answer is 19.

Binary Tree Representations

A binary tree data structure is represented using two methods. Those methods are as follows...

1. **Array Representation**
2. **Linked List Representation**

Consider the following binary tree...



1. Array Representation of Binary Tree

In array representation of a binary tree, we use one-dimensional array (1-D Array) to represent a binary tree.

Consider the above example of a binary tree and it is represented as follows...

A	B	C	D	F	G	H	I	J	-	-	-	K	-	-	-	-	-	-	-	-

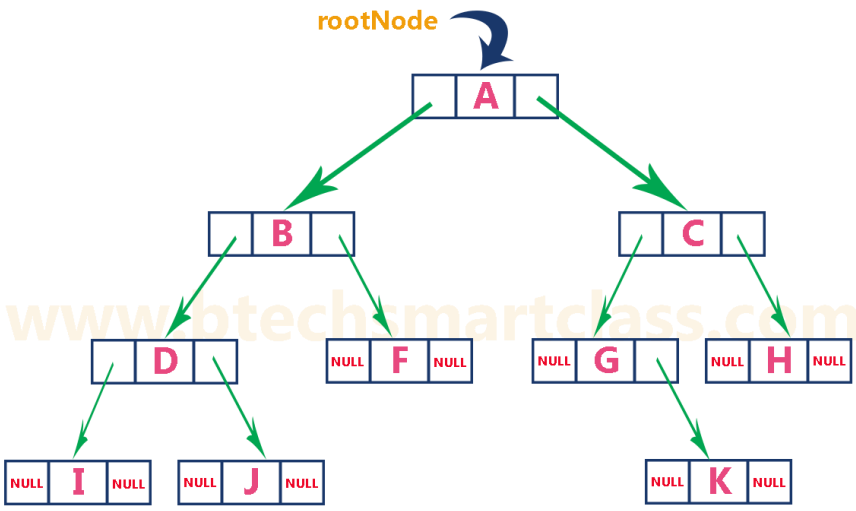
To represent a binary tree of depth **'n'** using array representation, we need one dimensional array with a maximum size of **2n + 1**.

2. Linked List Representation of Binary Tree

We use a double linked list to represent a binary tree. In a double linked list, every node consists of three fields. First field for storing left child address, second for storing actual data and third for storing right child address.

In this linked list representation, a node has the following structure...

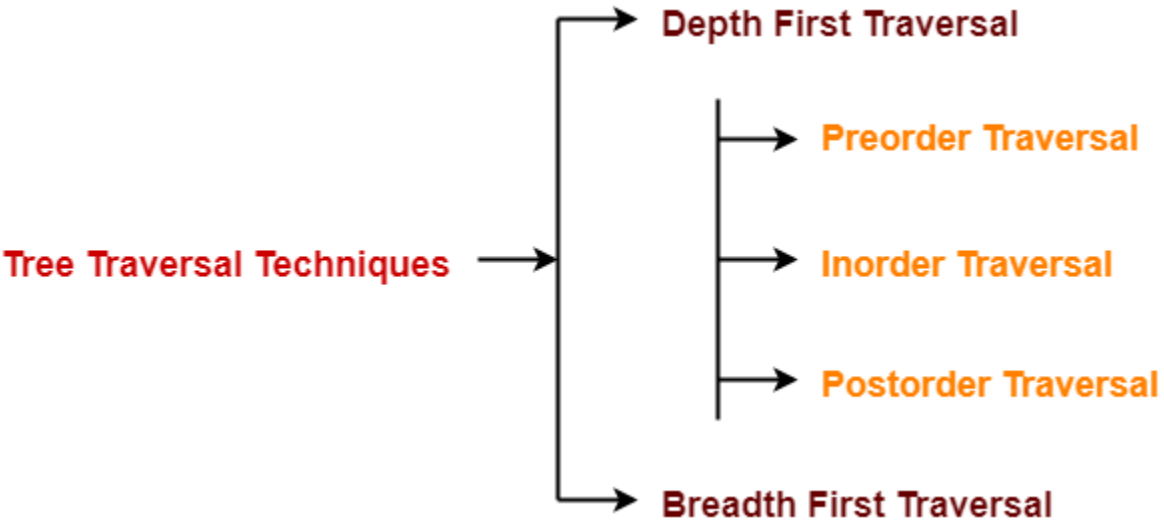
The above example of the binary tree represented using Linked list representation is shown as follows...



Tree Traversal-

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

Various tree traversal techniques are-



Depth First Traversal-

Following three traversal techniques fall under Depth First Traversal-

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

1. Preorder Traversal-

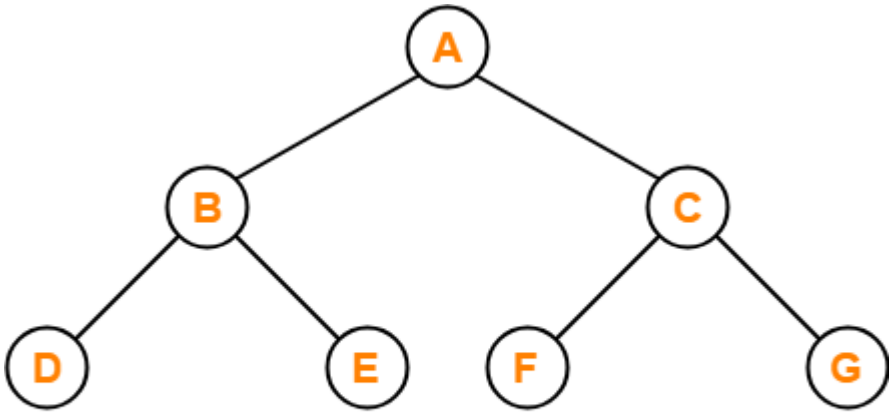
Algorithm-

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-

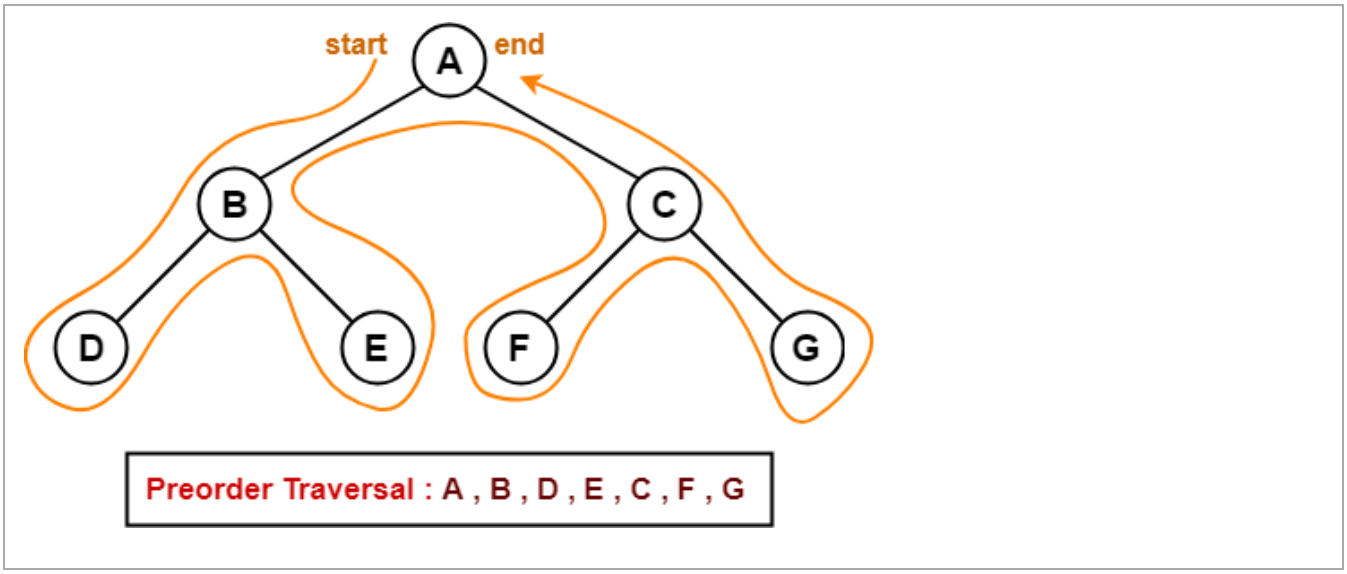
Consider the following example-



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.



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-

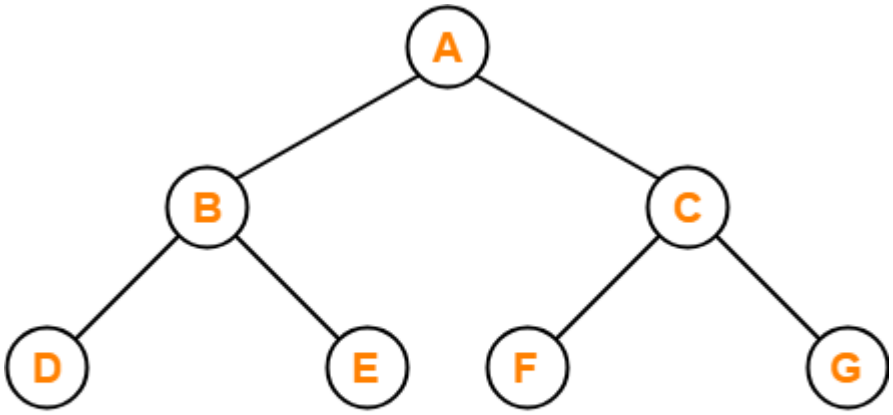
Algorithm-

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-

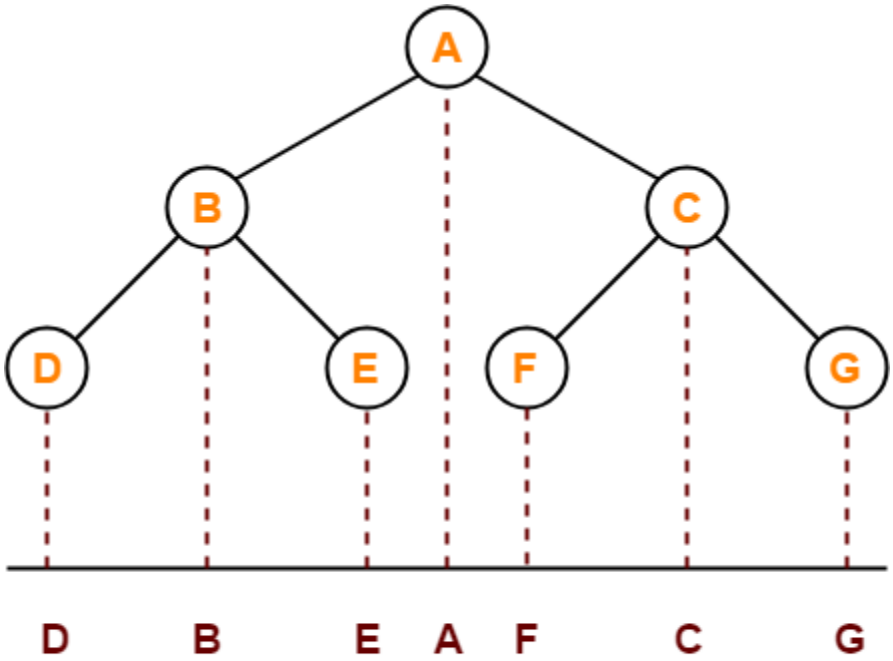
Consider the following example-



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-

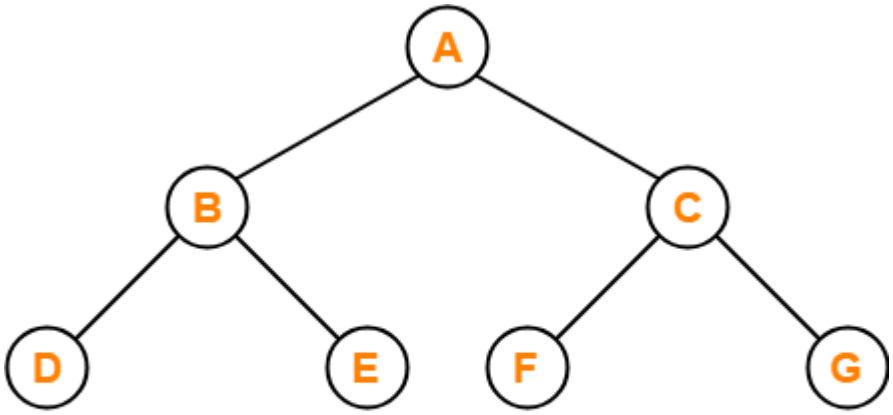
Algorithm-

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-

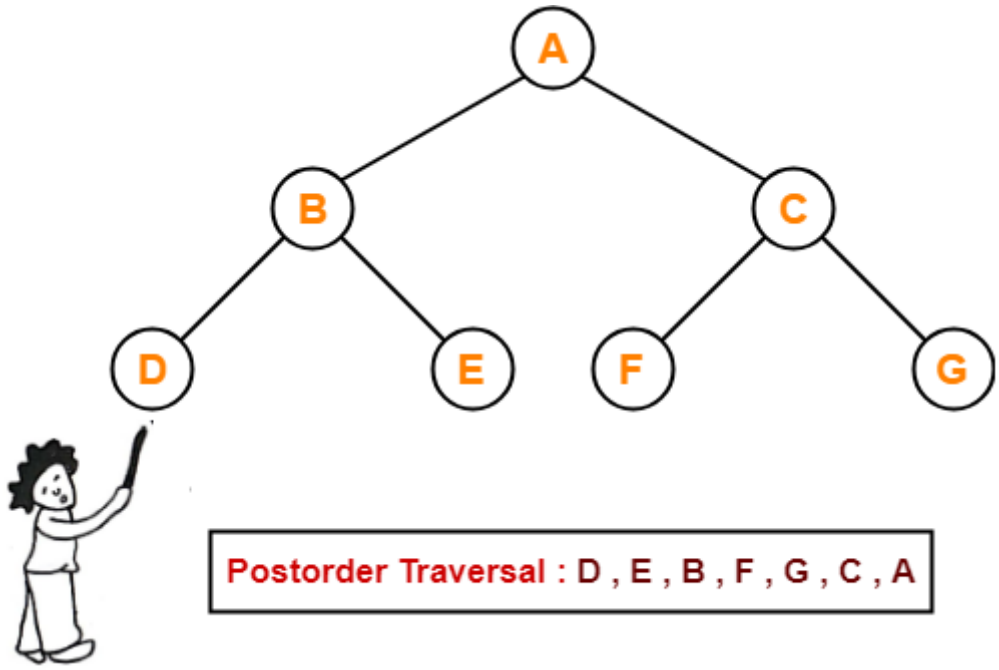
Consider the following example-



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

Postorder Traversal Shortcut

Pluck all the leftmost leaf nodes one by one.



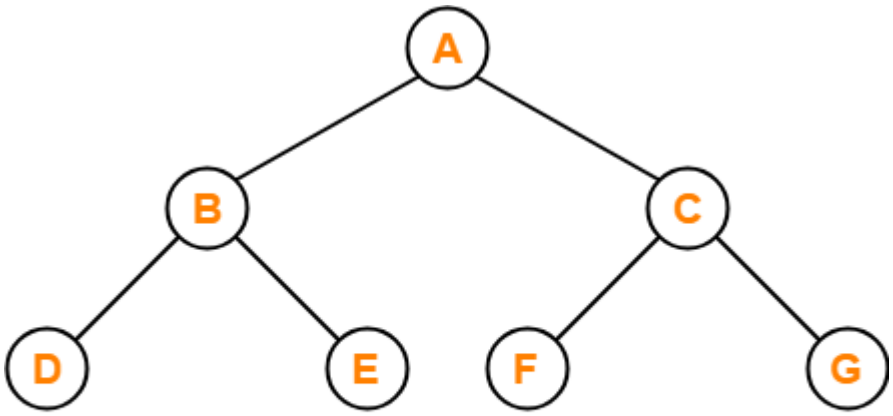
Applications-

- 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-



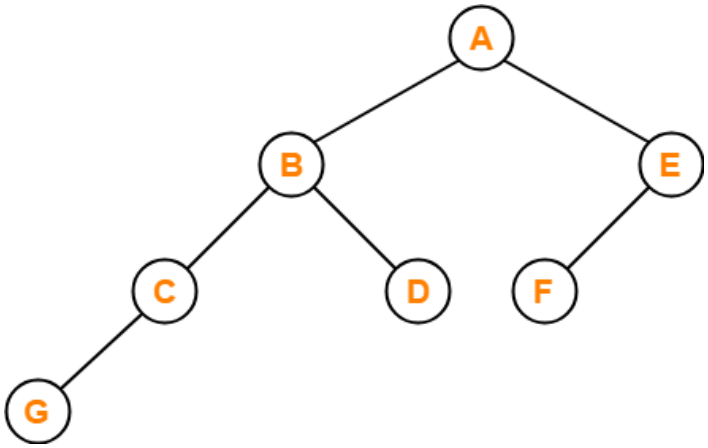
Level Order Traversal : A , B , C , D , E , F , G

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.

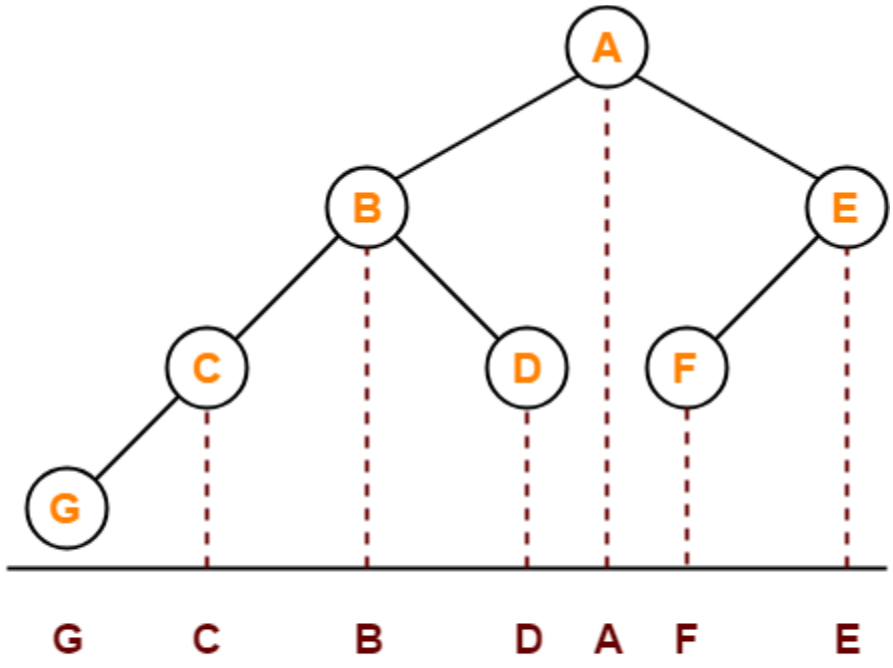
Problem-01:

If the binary tree in figure is traversed in inorder, then the order in which the nodes will be visited is ____?



Solution-

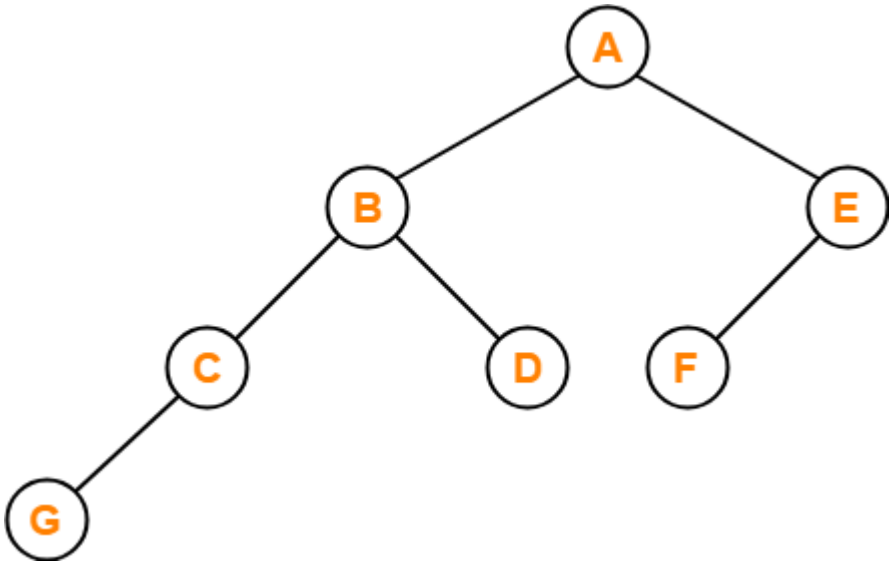
The inorder traversal will be performed as-



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

Problem-02:

Which of the following sequences denotes the postorder traversal sequence of the tree shown in figure?



1. FEGCBDBA
2. GCBDAFE
3. GCDBFEA
4. FDEGCBA

Solution-

Perform the postorder traversal by plucking all the leftmost leaf nodes one by one.

Then,

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

Thus, Option (C) is correct.

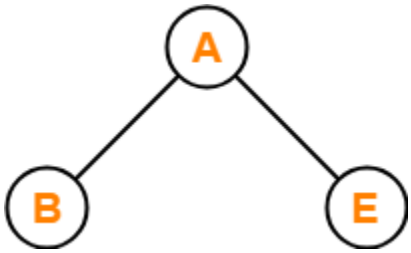
Problem-03:

Let LASTPOST, LASTIN, LASTPRE denote the last vertex visited in a postorder, inorder and preorder traversal respectively of a complete binary tree. Which of the following is always true?

1. LASTIN = LASTPOST
2. LASTIN = LASTPRE
3. LASTPRE = LASTPOST
4. None of these

Solution-

Consider the following complete binary tree-



Preorder Traversal : B , A , E

Inorder Traversal : B , A , E

Postorder Traversal : B , E , A

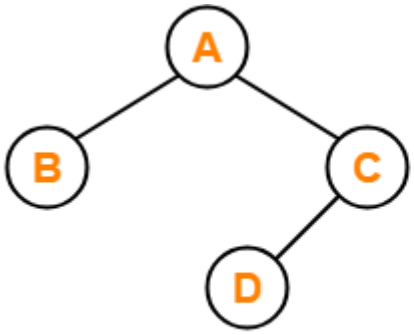
Clearly, LASTIN = LASTPRE.

Thus, Option (B) is correct.

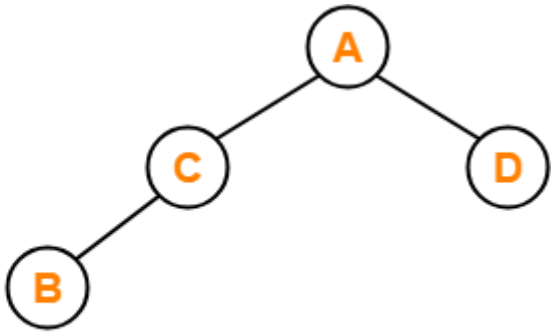
Problem-04:

Which of the following binary trees has its inorder and preorder traversals as BCAD and ABCD respectively-

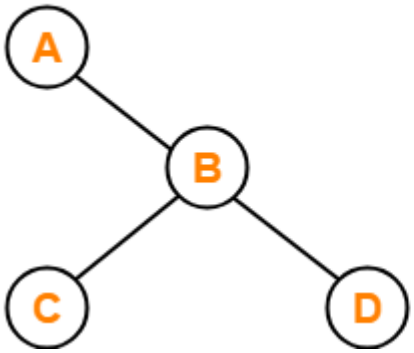
A)



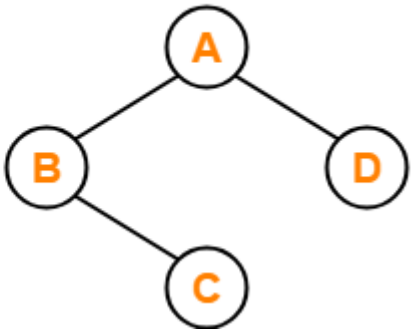
B)



C)



D)



Solution-

Option (D) is correct.