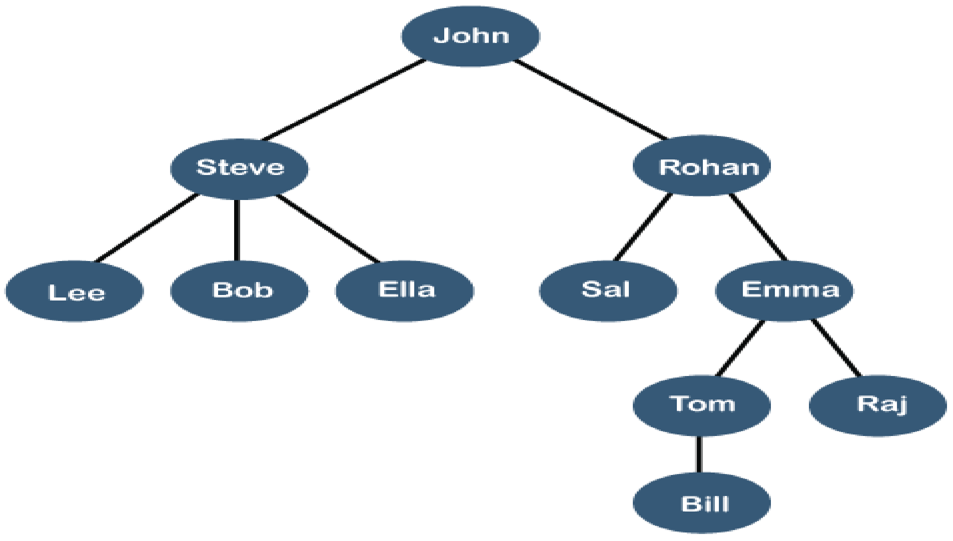
Trees

We read the linear data structures like an array, linked list, stack and queue in which all the elements are arranged in a sequential manner. The different data structures are used for different kinds of data.

Some factors are considered for choosing the data structure:

- **What type of data needs to be stored**?: It might be a possibility that a certain data structure can be the best fit for some kind of data.
- **Cost of operations:** If we want to minimize the cost for the operations for the most frequently performed operations. For example, we have a simple list on which we have to perform the search operation; then, we can create an array in which elements are stored in sorted order to perform the ***binary search***. The binary search works very fast for the simple list as it divides the search space into half.
- **Memory usage:** Sometimes, we want a data structure that utilizes less memory.

A tree is also one of the data structures that represent hierarchical data. Suppose we want to show the employees and their positions in the hierarchical form then it can be represented as shown below:

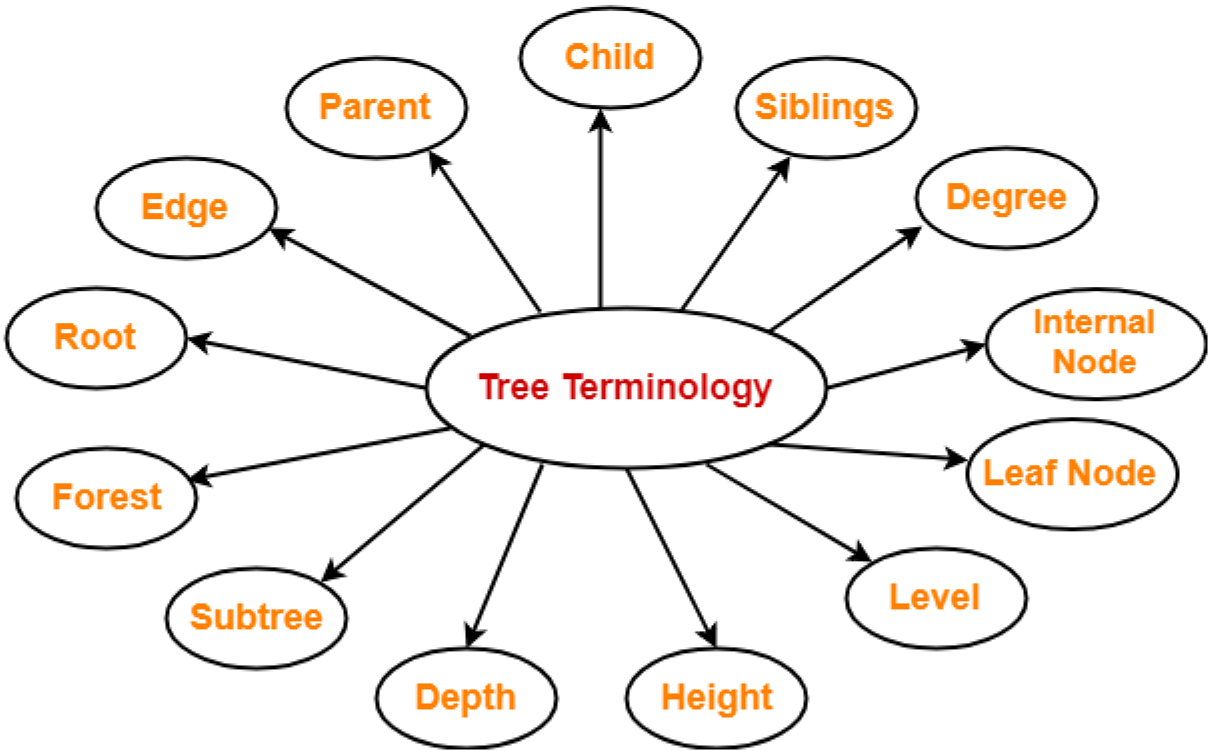


The above tree shows the **organization hierarchy** of some company.

- In the above structure, ***john*** is the CEO of the company, and John has two direct reports named as ***Steve*** and ***Rohan***.
- Steve has three direct reports named ***Lee, Bob, Ella*** where ***Steve*** is a manager.
- Bob has two direct reports named ***Sal*** and ***Emma***. Emma has two direct reports named ***Tom*** and ***Raj***. Tom has one direct report named ***Bill***.
- This particular logical structure is known as a ***Tree***.
- Its structure is similar to the real tree, so it is named a ***Tree***.
- In this structure, the ***root*** is at the top, and its branches are moving in a downward direction.
- Therefore, we can say that the Tree data structure is an efficient way of storing the data in a hierarchical way.

Tree Terminology-

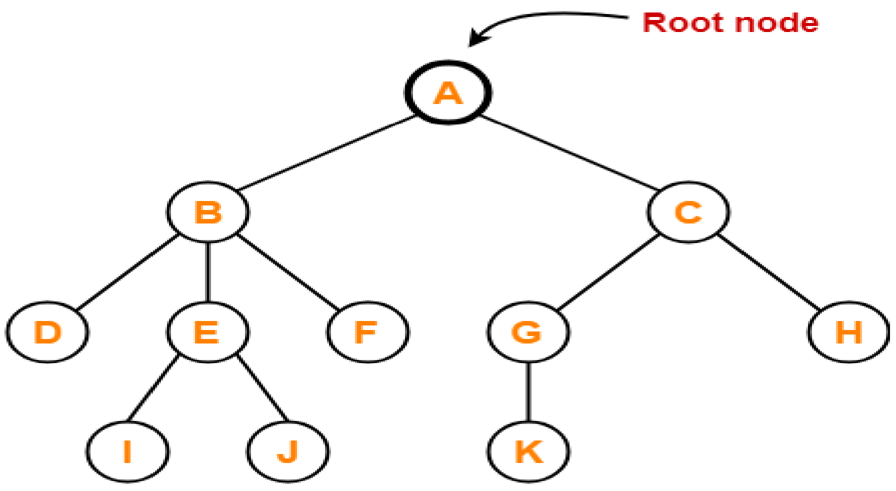
The important terms related to tree data structure are-



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-

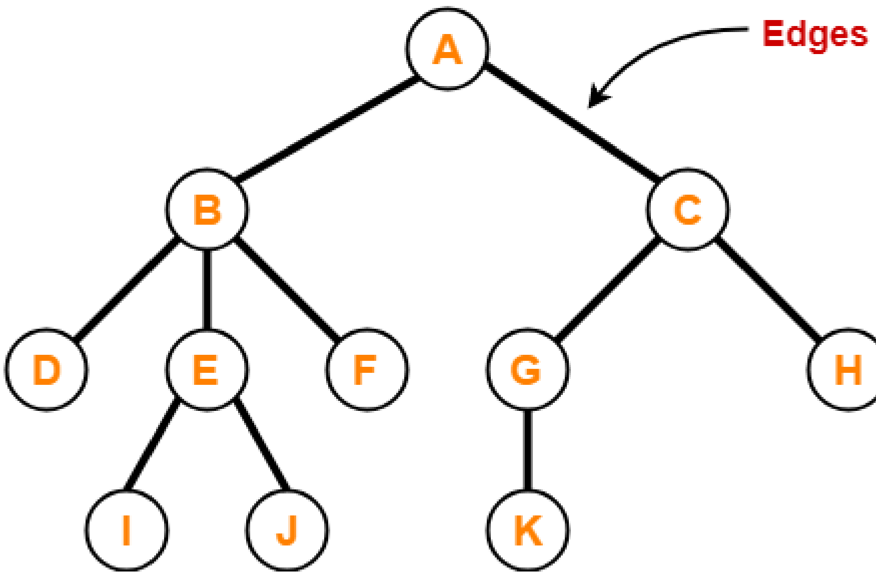


Here, node A is the only root node.

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.

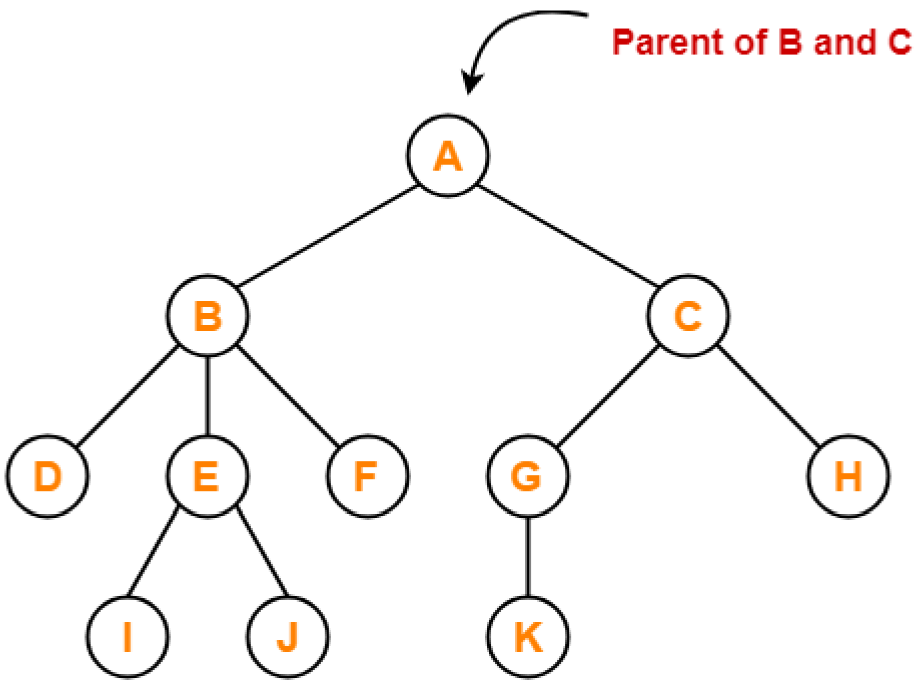
Example-



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-



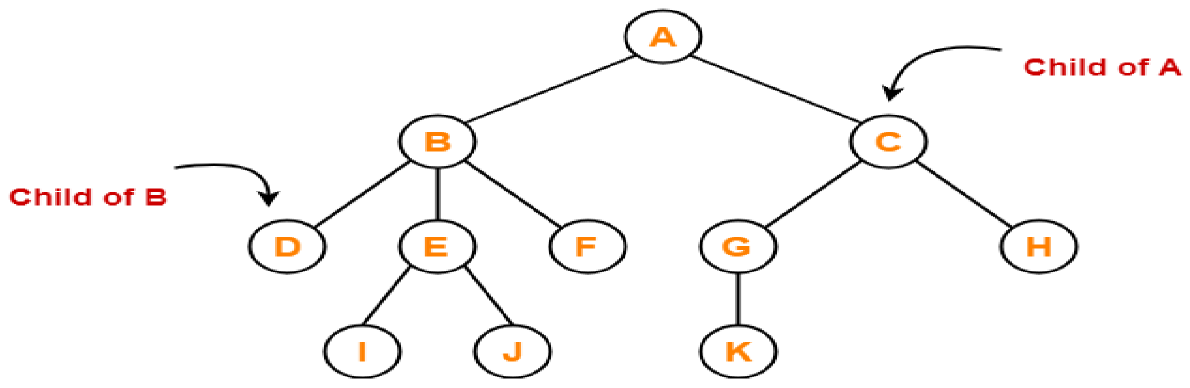
Here,

- 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-



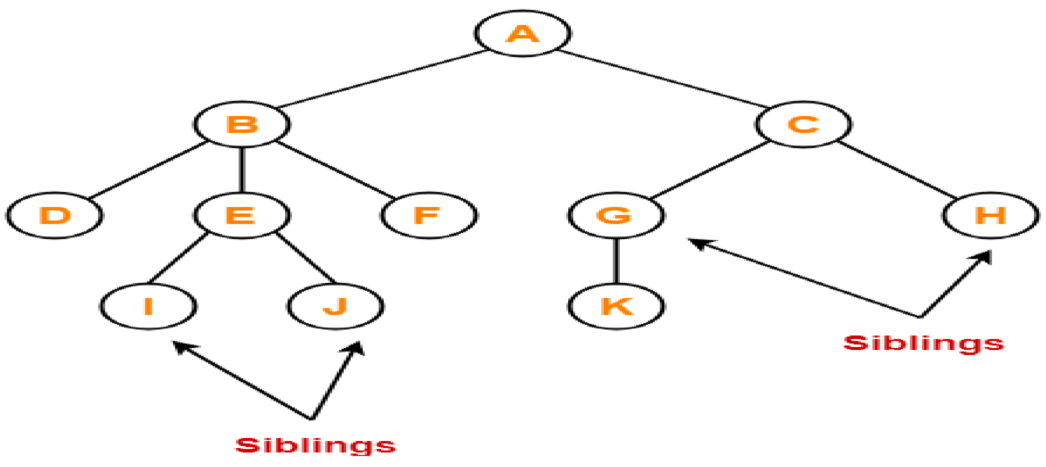
Here,

- 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-



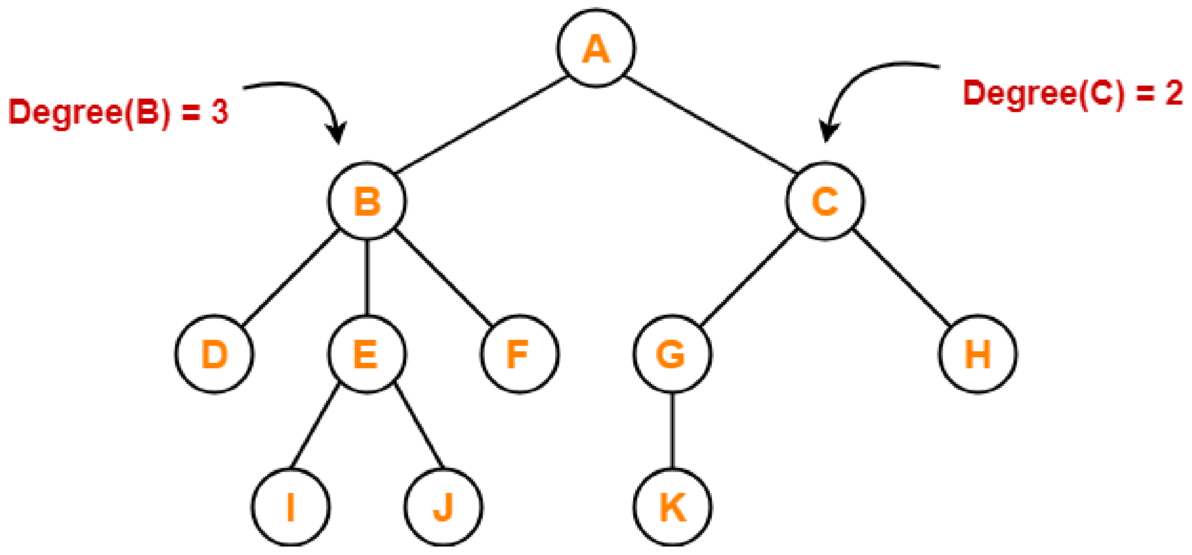
Here,

- 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,

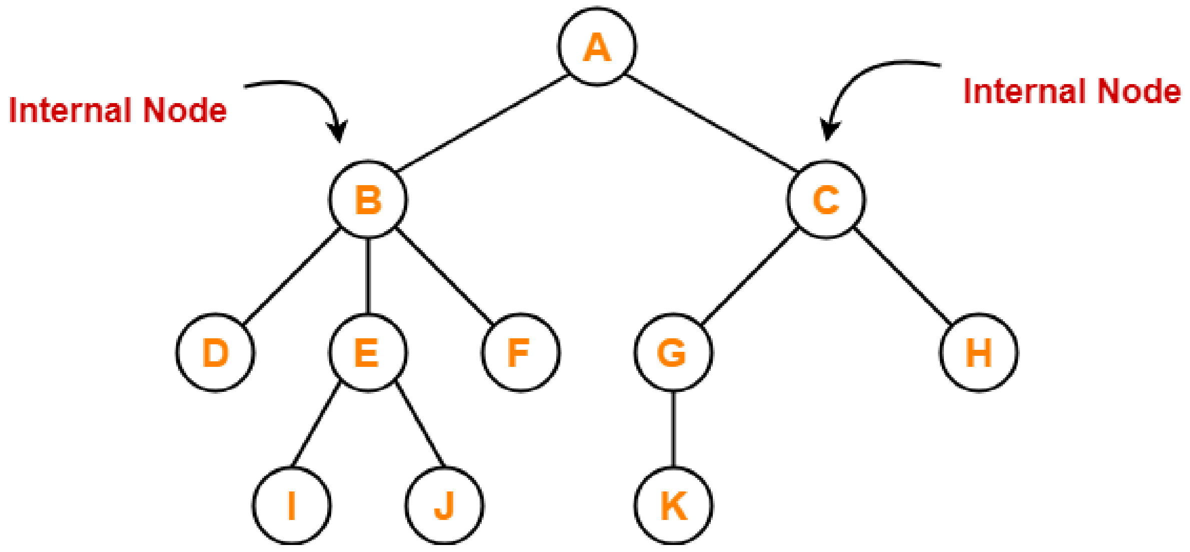
- Degree of node A = 2
- Degree of node B = 3
- Degree of node C = 2
- Degree of node D = 0

- 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-

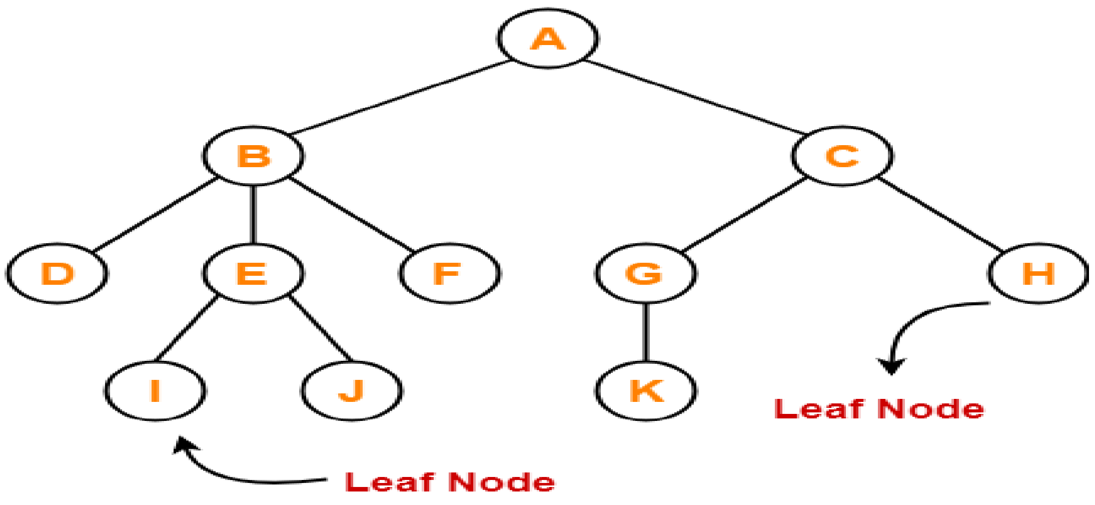


Here, nodes A, B, C, E and G are internal nodes.

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-

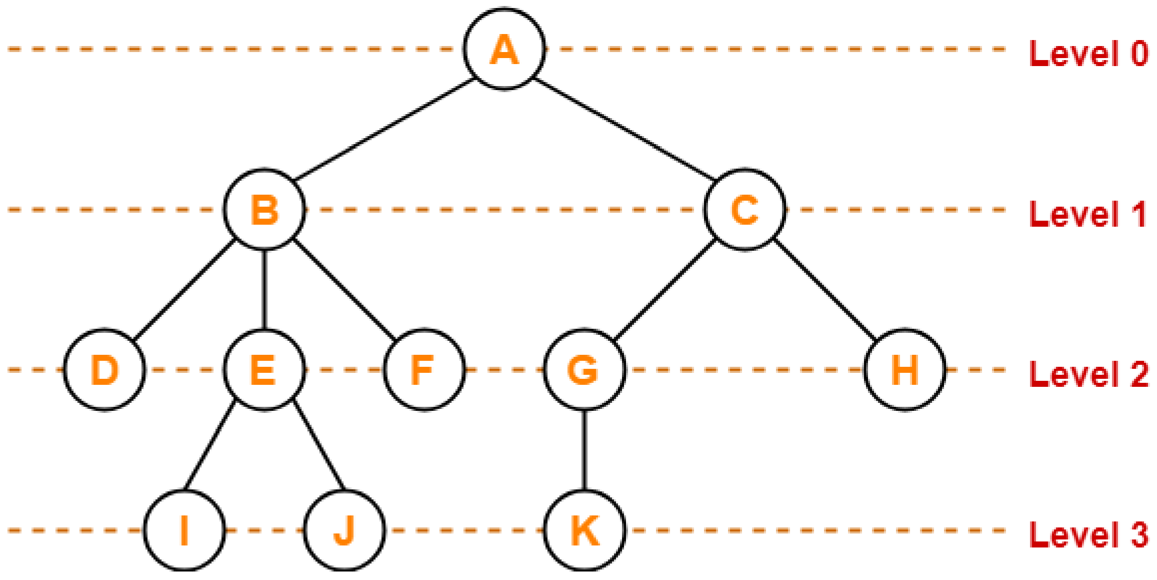


Here, nodes D, I, J, F, K and H are leaf nodes.

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.

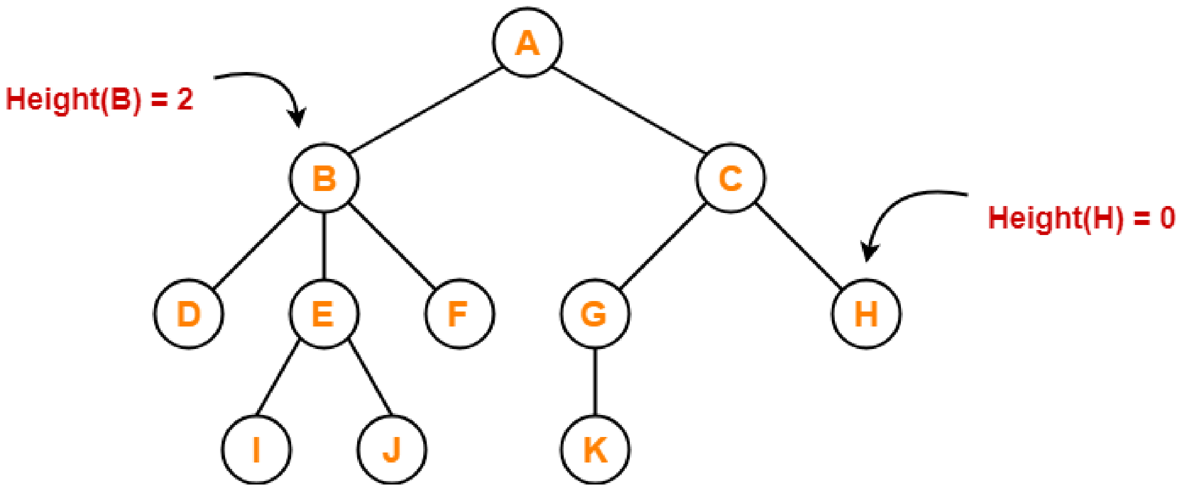
Example-



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-



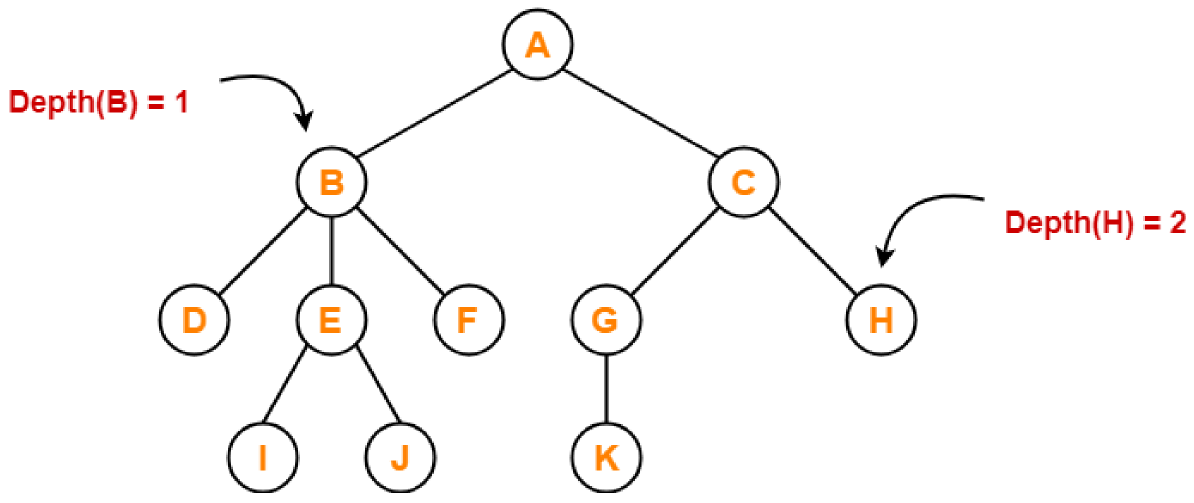
Here,

- 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-



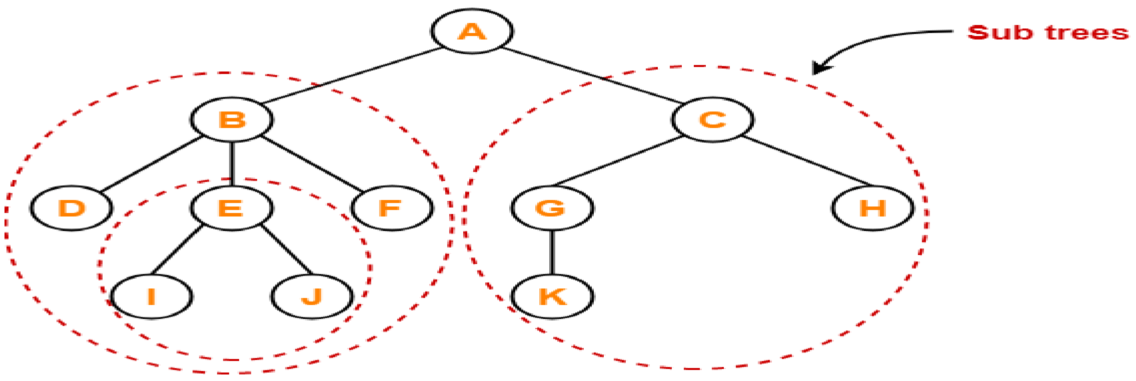
Here,

- 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.

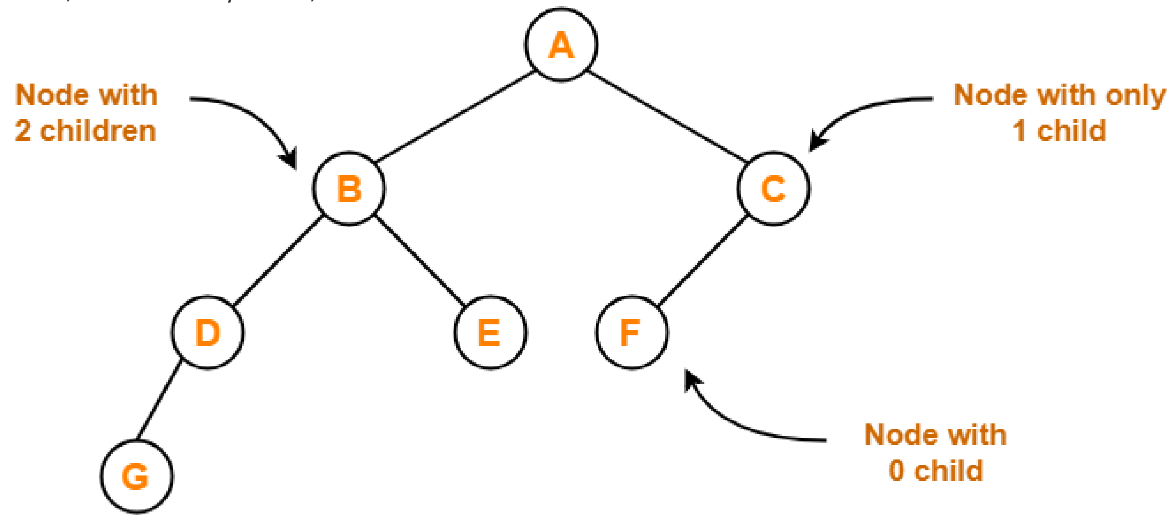
Example-



Binary tree

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

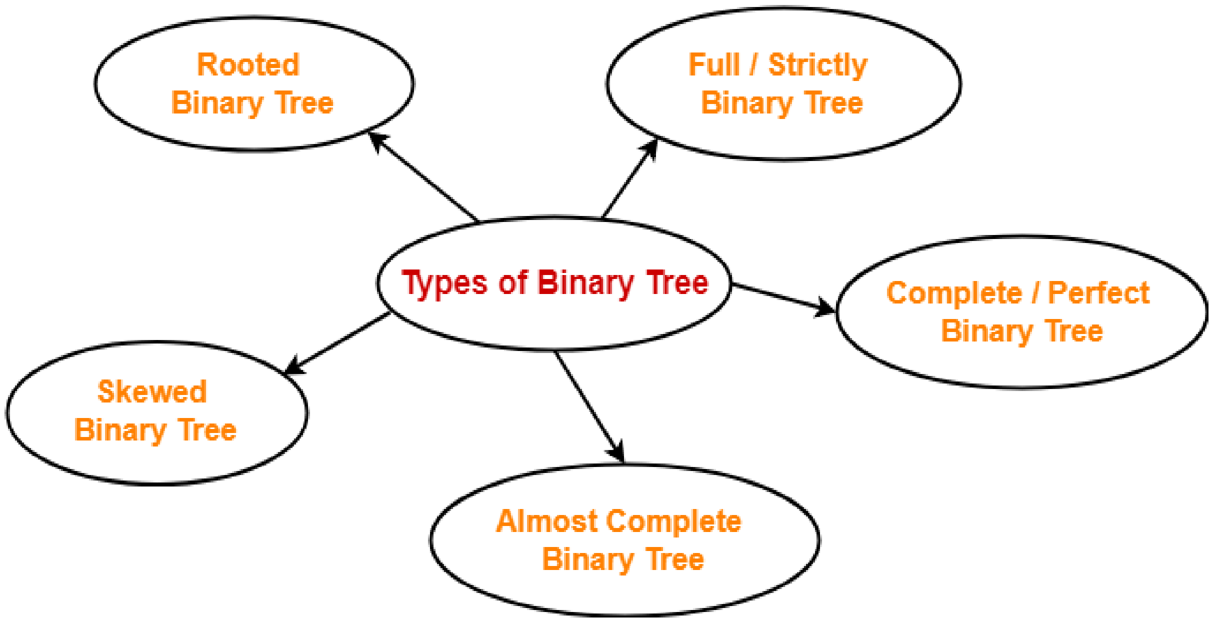
Thus, in a binary tree,Each node has either 0 child or 1 child or 2 children.



Binary Tree Example

Types of Binary Trees-

Binary trees can be of the following types-


Rooted Binary Tree
Full / Strictly Binary Tree
Types of Binary Tree
Complete / Perfect Binary Tree
Skewed Binary Tree
Almost Complete Binary Tree

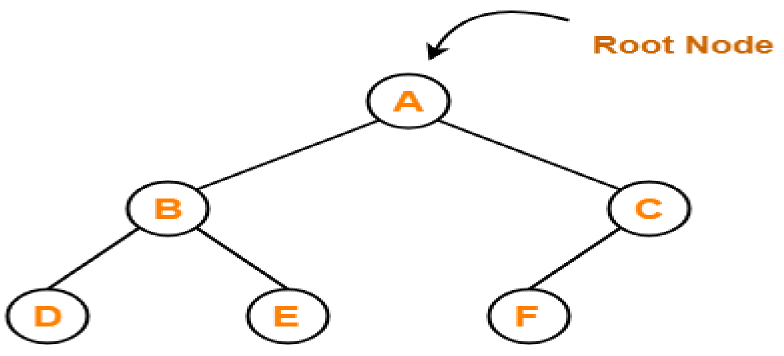
1. Rooted Binary Tree
2. Full / Strictly Binary Tree
3. Complete / Perfect Binary Tree
4. Almost Complete Binary Tree
5. Skewed Binary Tree

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-

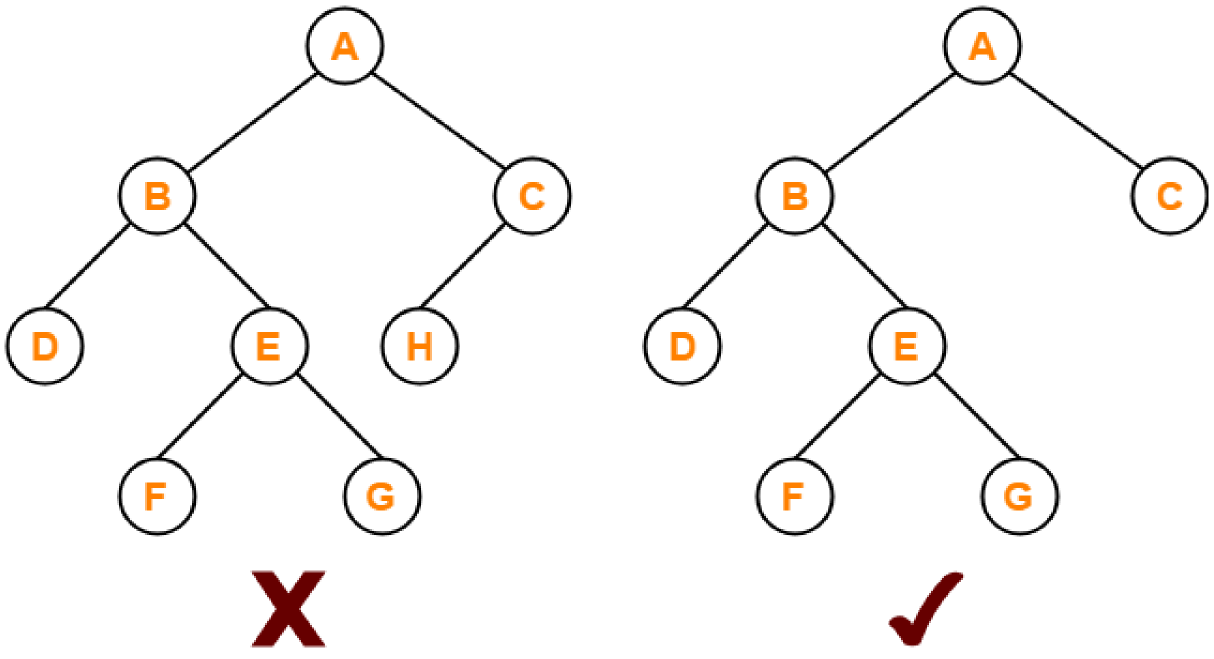


Rooted Binary Tree

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.

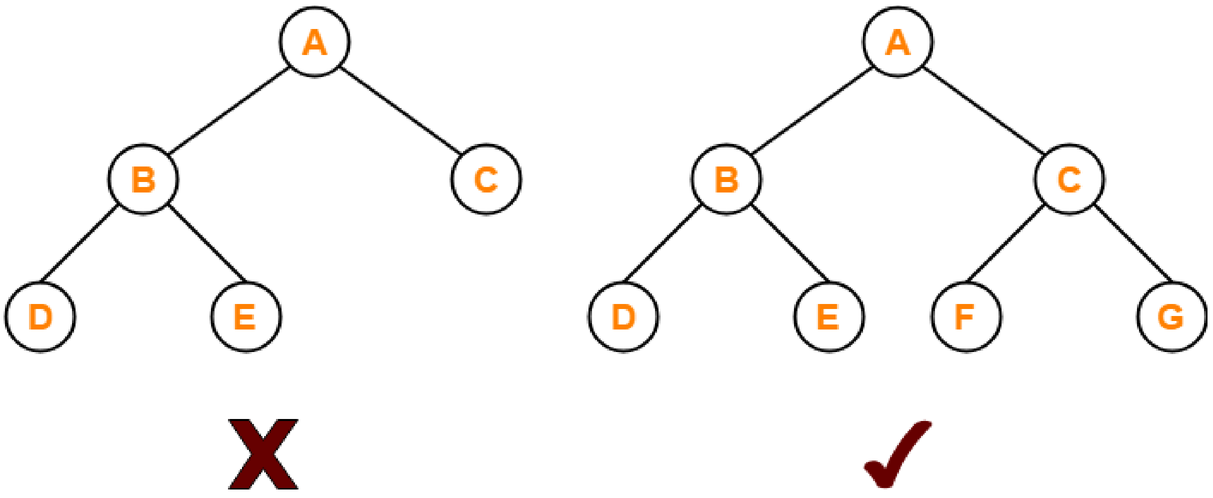
3. Complete / Perfect Binary Tree-

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,

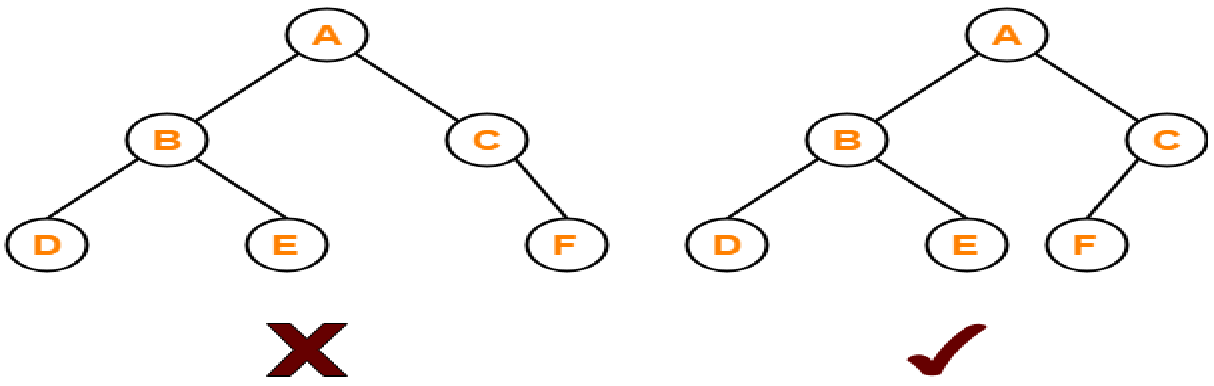
- First binary tree is not a complete binary tree.
- This is because all the leaf nodes are not at the same level.

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-

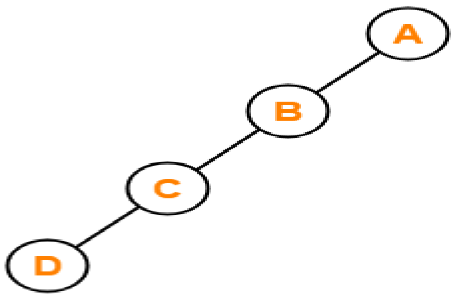
A **skewed binary tree** is a binary tree that satisfies the following 2 properties-

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

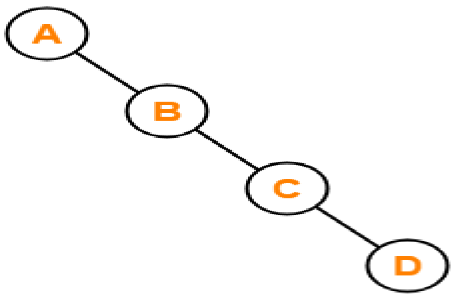
OR

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

Example-



Left Skewed Binary Tree



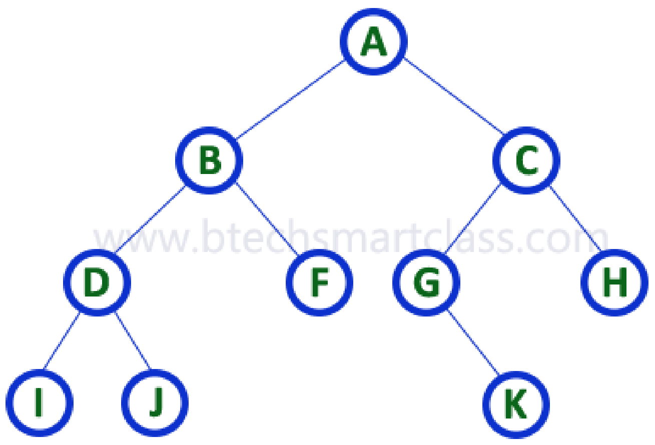
Right Skewed Binary Tree

Binary Tree Representations

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

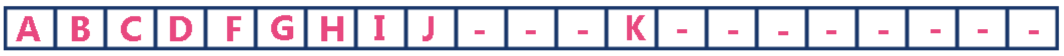
1. Array Representation
2. Linked List Representation

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

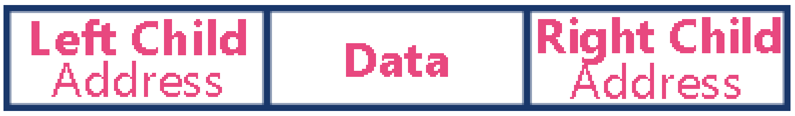


A	B	C	D	F	G	H	I	J	-	-	-	K	-	-	-	-	-	-	-	-

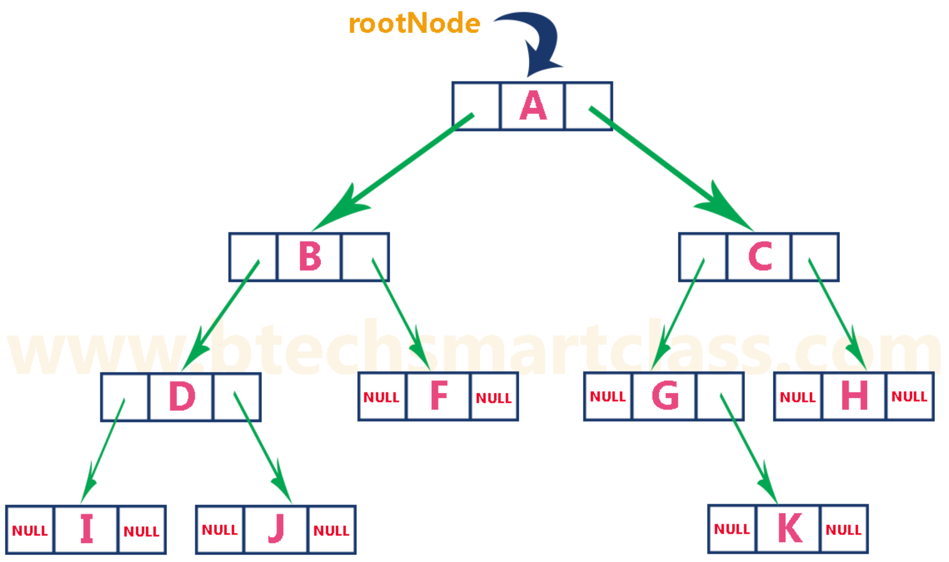
To represent a binary tree of depth **'n'** using array representation, we need one dimensional array with a maximum size of 2n + 1.

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



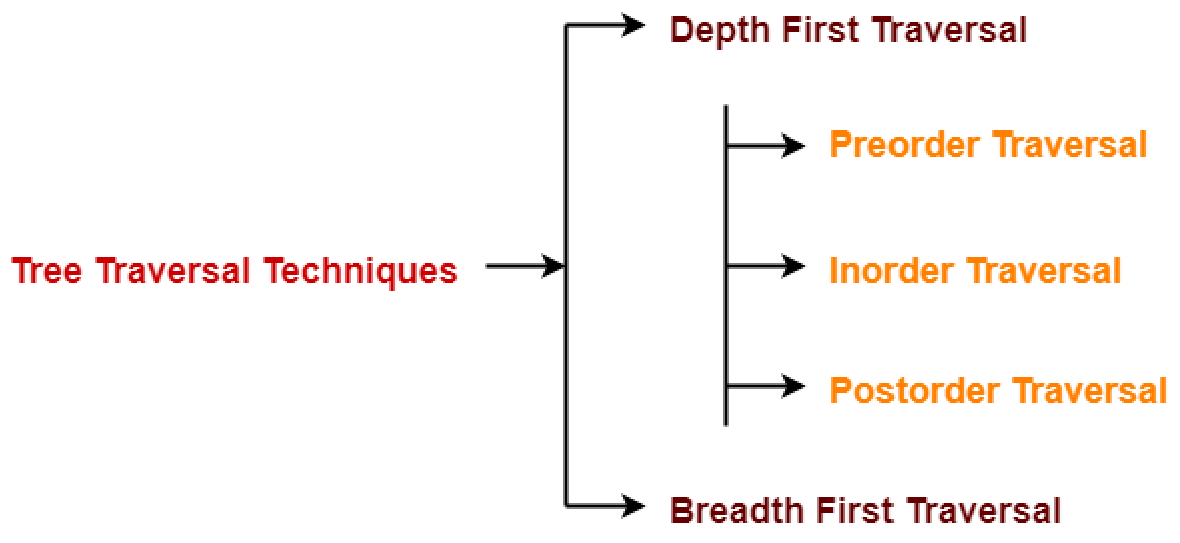
The above example of the binary tree represented using Linked list representation is shown as follows...



Tree Traversal

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

Various tree traversal techniques are-



<u>Depth First Traversal-</u>

Following three traversal techniques fall under Depth First Traversal-

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

<u>1. Preorder Traversal-</u>

<u>Algorithm-</u>

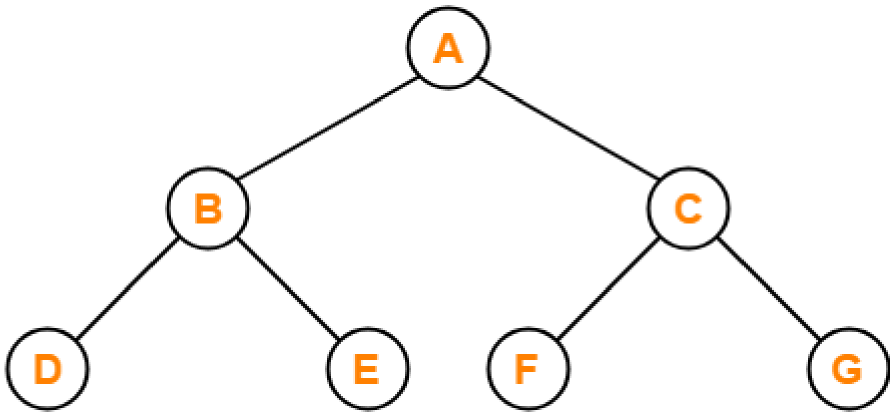
Visit the root

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

Root → Left → Right

<u>Example-</u>

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-

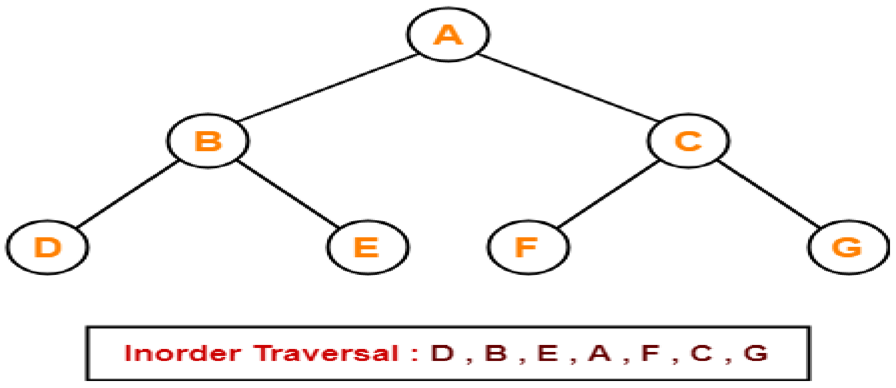
Algorithm-

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

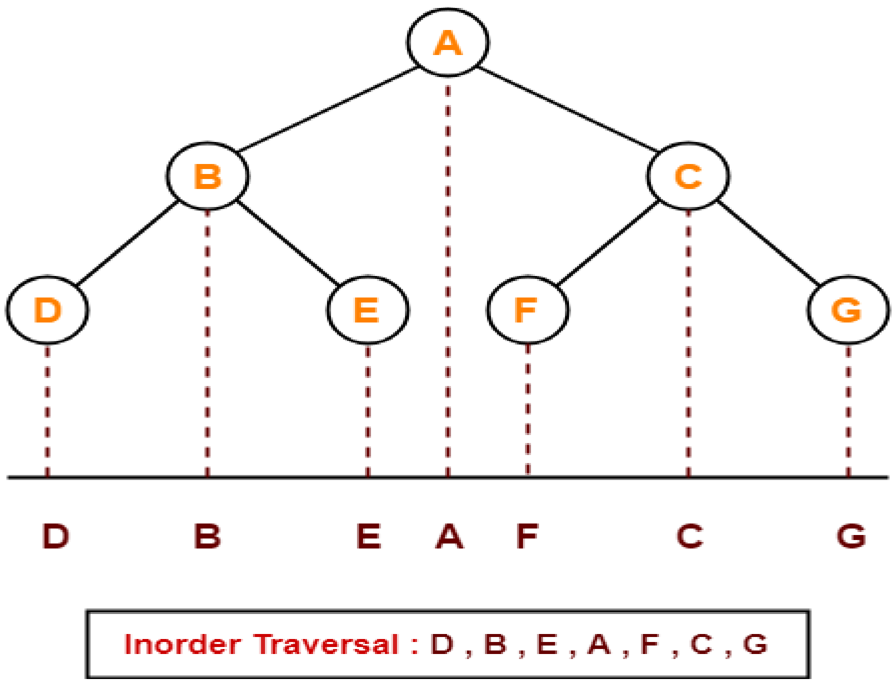
Example-

Consider the following example-



Inorder Traversal Shortcut

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



Application-

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

3. Postorder Traversal-

Algorithm-

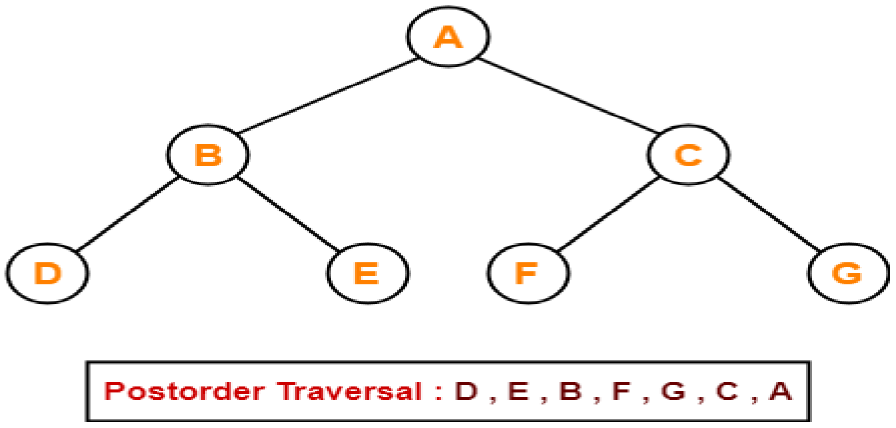
1. Traverse the left sub tree i.e. call Postorder (left sub tree)

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

Left → Right → Root

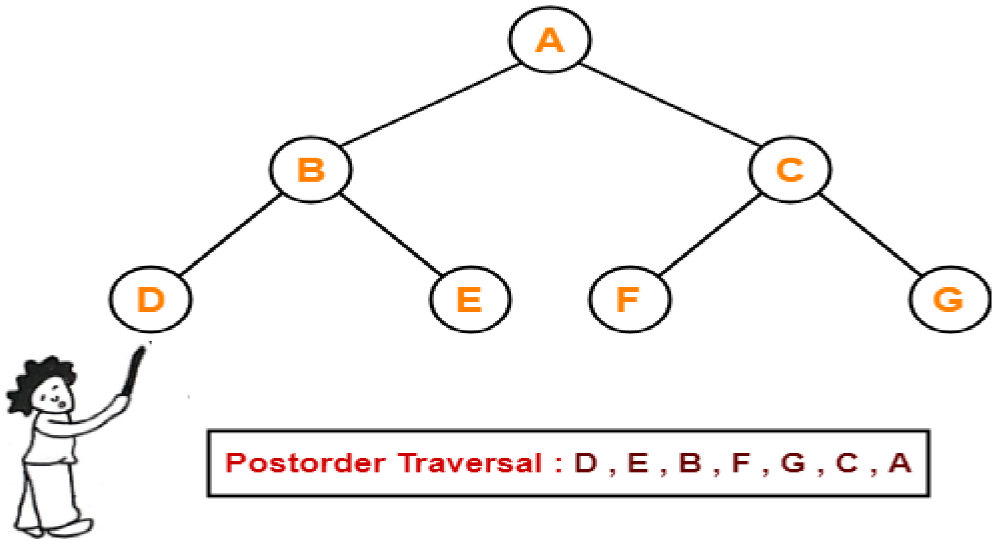
Example-

Consider the following example-



Postorder Traversal Shortcut

Pluck all the leftmost leaf nodes one by one.



Applications-

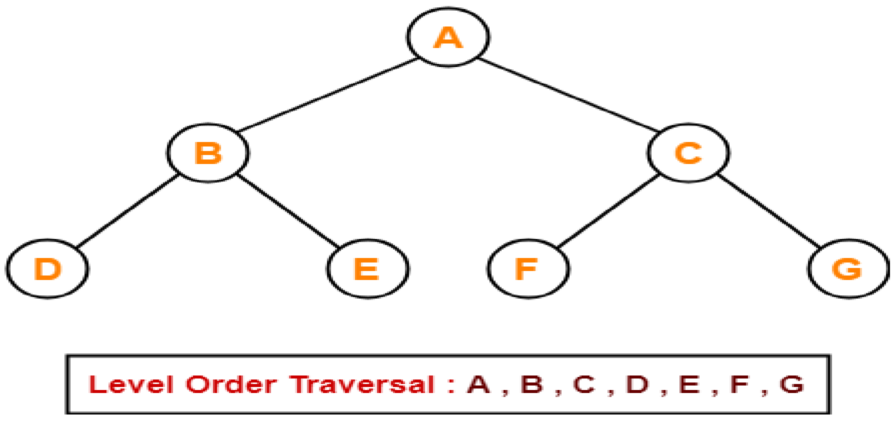
- Postorder traversal is used to get postfix expression of an expression tree.

- 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-



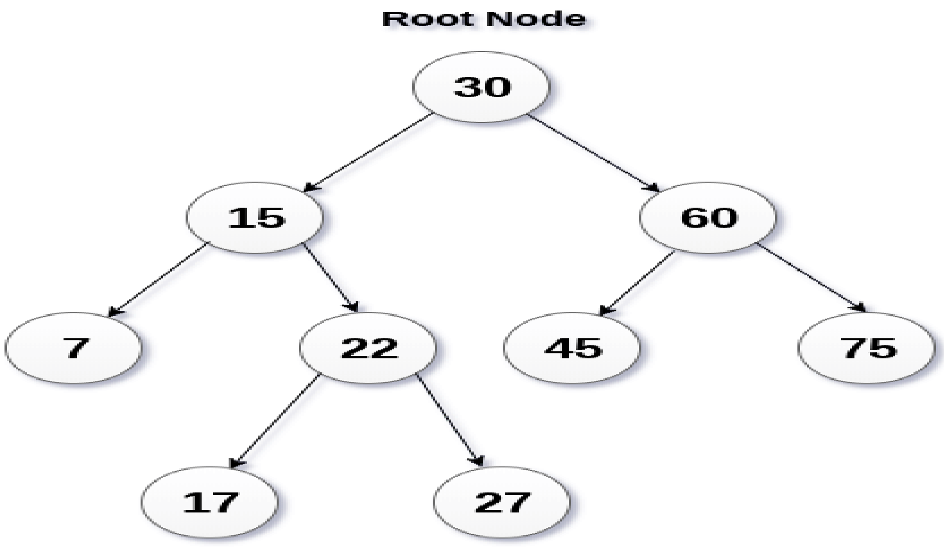
Application-

- 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

- Binary Search tree can be defined as a class of binary trees, in which the nodes are arranged in a specific order. This is also called ordered binary tree.
- In a binary search tree, the value of all the nodes in the left sub-tree is less than the value of the root.
- Similarly, value of all the nodes in the right sub-tree is greater than or equal to the value of the root.

- This rule will be recursively applied to all the left and right sub-trees of the root.



Binary Search Tree

A Binary search tree is shown in the above figure. As the constraint applied on the BST, we can see that the root node 30 doesn't contain any value greater than or equal to 30 in its left sub-tree and it also doesn't contain any value less than 30 in its right sub-tree.

Advantages of using binary search tree

- Searching become very efficient in a binary search tree since, we get a hint at each step, about which sub-tree contains the desired element.
- The binary search tree is considered as efficient data structure in compare to arrays and linked lists. In searching process, it removes half sub-tree at every step. Searching for an element in a binary search tree takes o($\log_2 n$) time. In worst case, the time it takes to search an element is 0(n).
- It also speed up the insertion and deletion operations as compare to that in array and linked list.

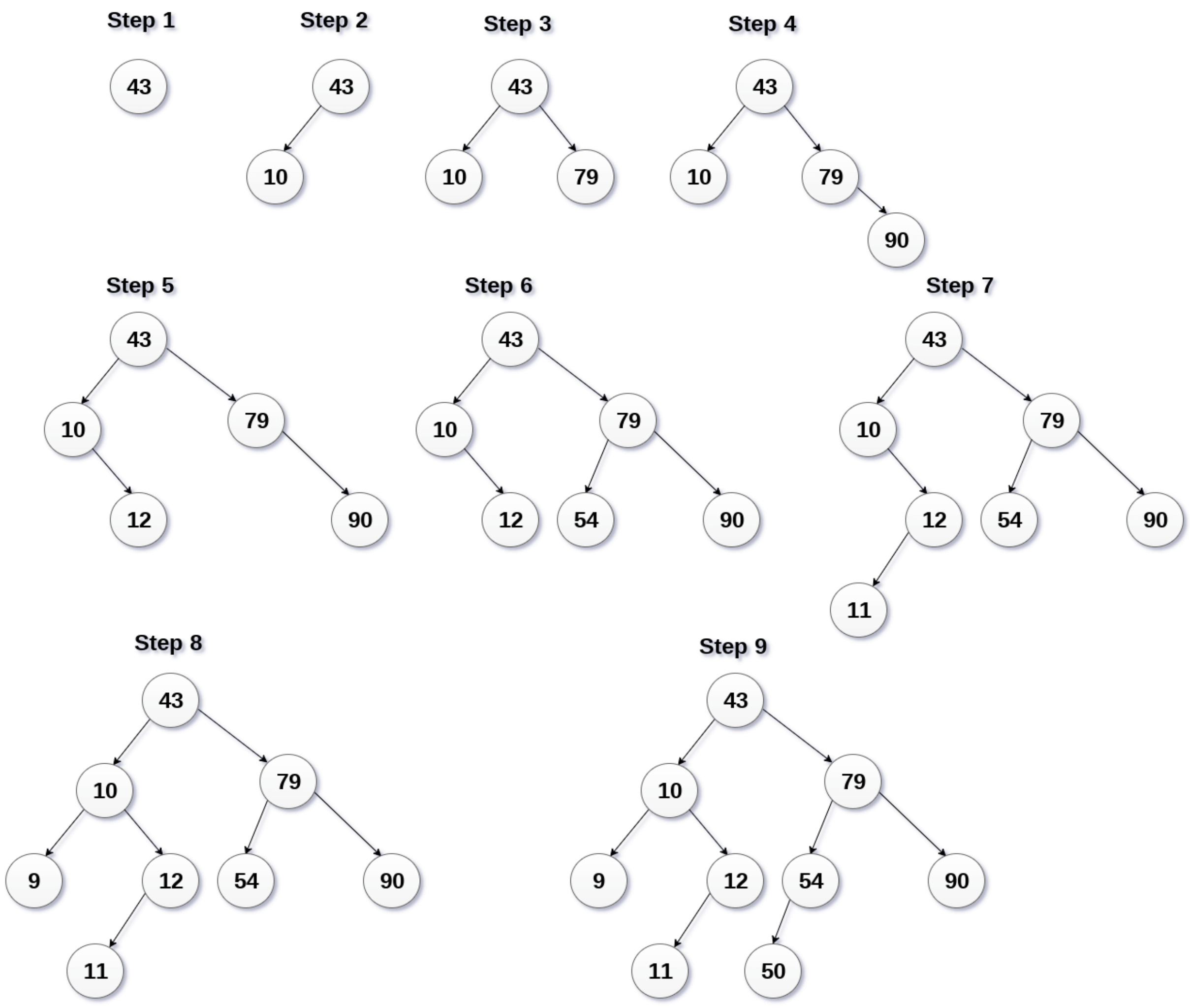
Example: Create the binary search tree using the following data elements.

43, 10, 79, 90, 12, 54, 11, 9, 50

- Insert 43 into the tree as the root of the tree.

- Read the next element, if it is lesser than the root node element, insert it as the root of the left sub-tree.
- Otherwise, insert it as the root of the right of the right sub-tree.

The process of creating BST by using the given elements, is shown in the image below.



Binary search Tree Creation

Operations on a Binary Search Tree

The following operations are performed on a binary search tree...

1. Search
2. Insertion

3. Deletion

Search Operation in BST

In a binary search tree, the search operation is performed with **O(log n)** time complexity. The search operation is performed as follows...

- Step 1 - Read the search element from the user.
- Step 2 - Compare the search element with the value of root node in the tree.
- Step 3 - If both are matched, then display "Given node is found!!!" and terminate the function
- Step 4 - If both are not matched, then check whether search element is smaller or larger than that node value.
- Step 5 - If search element is smaller, then continue the search process in left subtree.
- Step 6- If search element is larger, then continue the search process in right subtree.
- Step 7 - Repeat the same until we find the exact element or until the search element is compared with the leaf node
- Step 8 - If we reach to the node having the value equal to the search value then display "Element is found" and terminate the function.
- Step 9 - If we reach to the leaf node and if it is also not matched with the search element, then display "Element is not found" and terminate the function.

Insertion Operation in BST

In a binary search tree, the insertion operation is performed with **O(log n)** time complexity. In binary search tree, new node is always inserted as a leaf node. The insertion operation is performed as follows...

- Step 1 - Create a newNode with given value and set its **left** and **right** to **NULL**.
- Step 2 - Check whether tree is Empty.
- Step 3 - If the tree is **Empty**, then set **root** to **newNode**.
- Step 4 - If the tree is **Not Empty**, then check whether the value of newNode is **smaller** or **larger** than the node (here it is root node).
- Step 5 - If newNode is **smaller** than **or equal** to the node then move to its **left** child. If newNode is **larger** than the node then move to its **right** child.

- Step 6- Repeat the above steps until we reach to the **leaf** node (i.e., reaches to NULL).
- Step 7 - After reaching the leaf node, insert the newNode as **left child** if the newNode is **smaller or equal** to that leaf node or else insert it as **right child**.

Deletion Operation in BST

In a binary search tree, the deletion operation is performed with **O(log n)** time complexity. Deleting a node from Binary search tree includes following three cases...

- Case 1: **Deleting a Leaf node (A node with no children)**
- Case 2: **Deleting a node with one child**
- Case 3: **Deleting a node with two children**

Case 1: Deleting a leaf node

We use the following steps to delete a leaf node from BST...

- Step 1 - **Find** the node to be deleted using **search operation**
- Step 2 - Delete the node using **free** function (If it is a leaf) and terminate the function.

Case 2: Deleting a node with one child

We use the following steps to delete a node with one child from BST...

- Step 1 - **Find** the node to be deleted using **search operation**
- Step 2 - If it has only one child then create a link between its parent node and child node.
- Step 3 - Delete the node using **free** function and terminate the function.

Case 3: Deleting a node with two children

We use the following steps to delete a node with two children from BST...

- Step 1 - **Find** the node to be deleted using **search operation**
- Step 2 - If it has two children, then find the **largest** node in its **left subtree** (OR) the **smallest** node in its **right subtree**.
- Step 3 - **Swap** both **deleting node** and node which is found in the above step.
- Step 4 - Then check whether deleting node came to **case 1** or **case 2** or else goto step 2
- Step 5 - If it comes to **case 1**, then delete using case 1 logic.

- Step 6- If it comes to **case 2**, then delete using case 2 logic.
- Step 7 - Repeat the same process until the node is deleted from the tree.

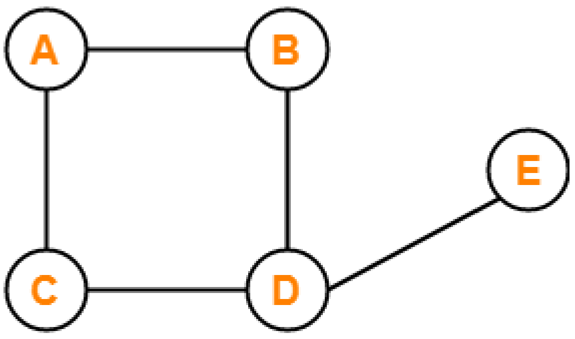
Graphs

- A graph is a collection of vertices connected to each other through a set of edges.
- A graph is defined as an ordered pair of a set of vertices and a set of edges.

G = (V, E)

Here, V is the set of vertices and E is the set of edges connecting the vertices.

Example-



Example of Graph

In this graph,

V = { A , B , C , D , E }

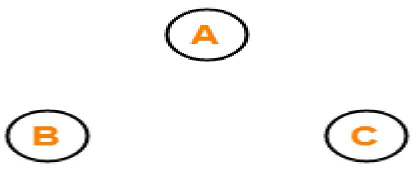
E = { AB , AC , BD , CD , DE }

Types of Graphs

1. Null Graph

- A graph whose edge set is empty is called as a null graph.
- In other words, a null graph does not contain any edges in it.

Example-



Example of Null Graph

Here,

- This graph consists only of the vertices and there are no edges in it.
- Since the edge set is empty, therefore it is a null graph.

2. Trivial Graph

- A graph having only one vertex in it is called as a trivial graph.
- It is the smallest possible graph.

Example

Example of Trivial Graph

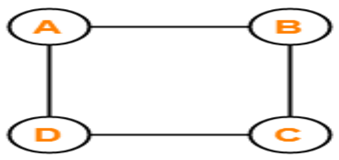
Here,

- This graph consists of only one vertex and there are no edges in it.
- Since only one vertex is present, therefore it is a trivial graph.

3. Non-Directed Graph

- A graph in which all the edges are undirected is called as a non-directed graph.
- In other words, edges of an undirected graph do not contain any direction.

Example-



Example of Non-Directed Graph

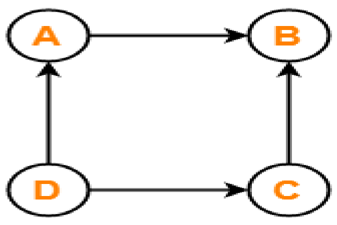
Here,

- This graph consists of four vertices and four undirected edges.
- Since all the edges are undirected, therefore it is a non-directed graph.

4. Directed Graph

- A graph in which all the edges are directed is called as a directed graph.
- In other words, all the edges of a directed graph contain some direction.
- Directed graphs are also called as **digraphs**.

Example-



Example of Directed Graph

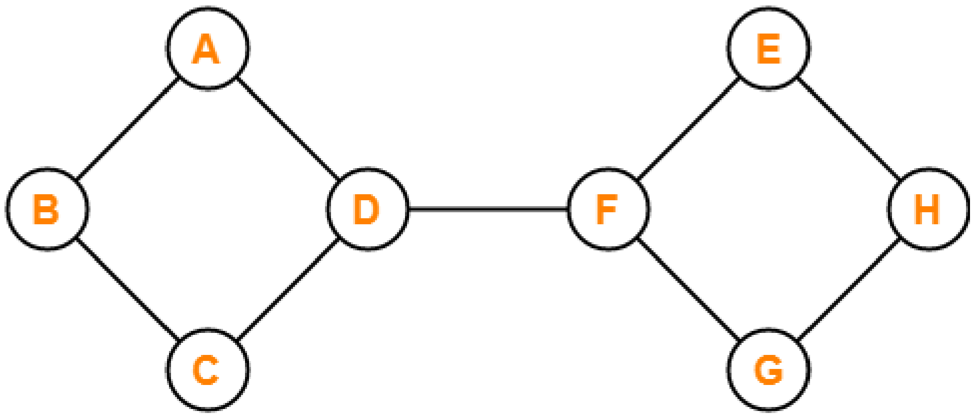
Here,

- This graph consists of four vertices and four directed edges.
- Since all the edges are directed, therefore it is a directed graph.

5. Connected Graph

- A graph in which we can visit from any one vertex to any other vertex is called as a connected graph.
- In connected graph, at least one path exists between every pair of vertices.

Example-



Example of Connected Graph

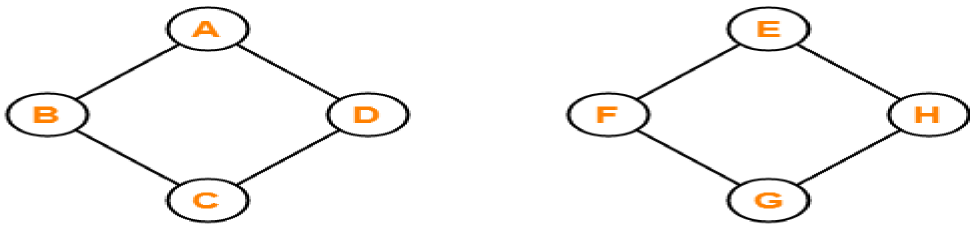
Here,

- In this graph, we can visit from any one vertex to any other vertex.
- There exists at least one path between every pair of vertices.
- Therefore, it is a connected graph.

6. Disconnected Graph

- A graph in which there does not exist any path between at least one pair of vertices is called as a disconnected graph.

Example-



Example of Disconnected Graph

Here,

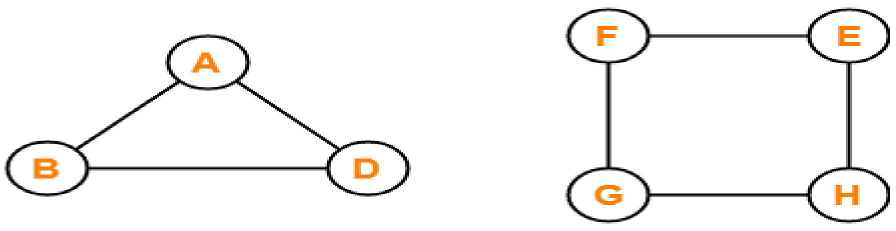
- This graph consists of two independent components which are disconnected.

- It is not possible to visit from the vertices of one component to the vertices of other component.
- Therefore, it is a disconnected graph.

7. Regular Graph

- A graph in which degree of all the vertices is same is called as a regular graph.
- If all the vertices in a graph are of degree ‘k’, then it is called as a “**k-regular graph**“.

Examples-



Examples of Regular Graph

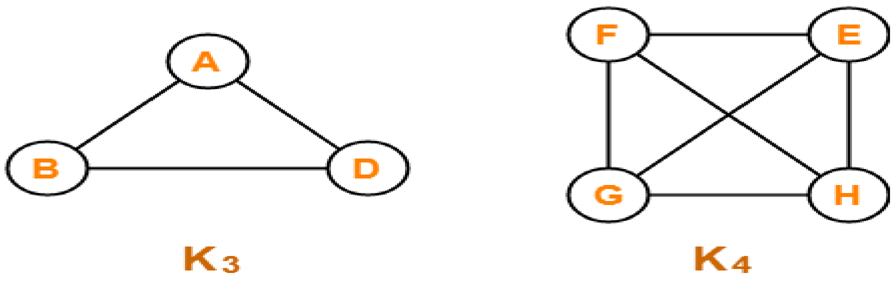
In these graphs,

- All the vertices have degree-2.
- Therefore, they are 2-Regular graphs.

8. Complete Graph

- A graph in which exactly one edge is present between every pair of vertices is called as a complete graph.
- A complete graph of ‘n’ vertices contains exactly $^nC_2$ edges.
- A complete graph of ‘n’ vertices is represented as K_n.

Examples-



Examples of Complete Graph

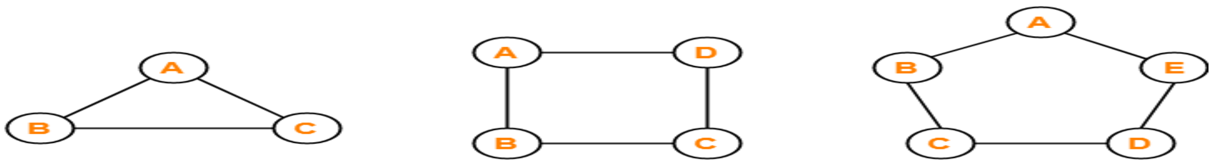
In these graphs,

- Each vertex is connected with all the remaining vertices through exactly one edge.
- Therefore, they are complete graphs.

9. Cycle Graph

- A simple graph of ‘n’ vertices (n>=3) and n edges forming a cycle of length ‘n’ is called as a cycle graph.
- In a cycle graph, all the vertices are of degree 2.

Examples-



Examples of Cycle Graph

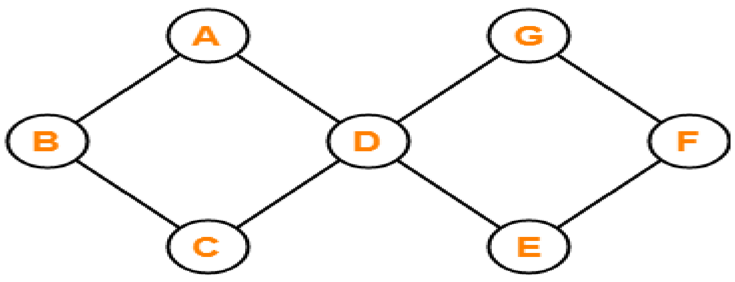
In these graphs,

- Each vertex is having degree 2.
- Therefore, they are cycle graphs.

10. Cyclic Graph

- A graph containing at least one cycle in it is called as a cyclic graph.

Example-



Example of Cyclic Graph

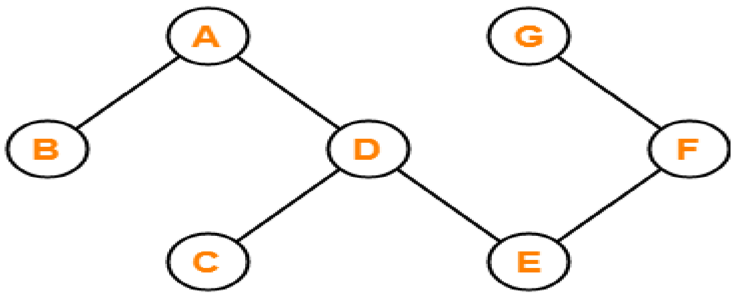
Here,

- This graph contains two cycles in it.
- Therefore, it is a cyclic graph.

11. Acyclic Graph

- A graph not containing any cycle in it is called as an acyclic graph.

Example-



Example of Acyclic Graph

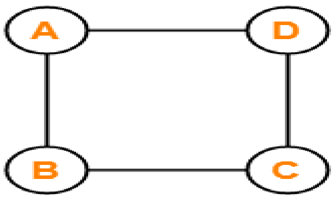
Here,

- This graph do not contain any cycle in it.
- Therefore, it is an acyclic graph.

12. Finite Graph

- A graph consisting of finite number of vertices and edges is called as a finite graph.

Example-



Example of Finite Graph

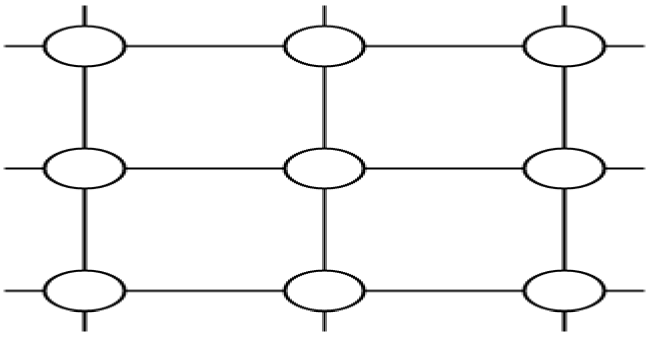
Here,

- This graph consists of finite number of vertices and edges.
- Therefore, it is a finite graph.

13. Infinite Graph

- A graph consisting of infinite number of vertices and edges is called as an infinite graph.

Example-


Example of Infinite Graph

Here,

- This graph consists of infinite number of vertices and edges.
- Therefore, it is an infinite graph.

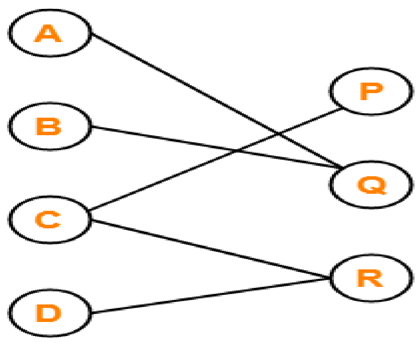
14. Bipartite Graph

A bipartite graph is a graph where-

- Vertices can be divided into two sets X and Y.
- The vertices of set X only join with the vertices of set Y.

- None of the vertices belonging to the same set join each other.

Example-

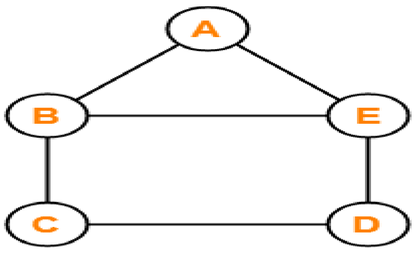


Example of Bipartite Graph

15. Planar Graph

- A planar graph is a graph that we can draw in a plane such that no two edges of it cross each other.

Example-



Example of Planar Graph

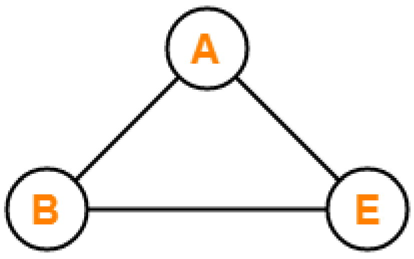
Here,

- This graph can be drawn in a plane without crossing any edges.
- Therefore, it is a planar graph.

16. Simple Graph

- A graph having no self loops and no parallel edges in it is called as a simple graph.

Example-



Example of Simple Graph

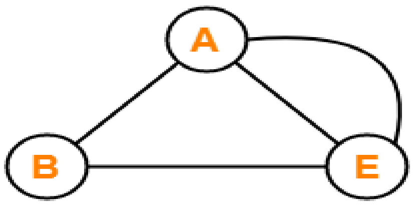
Here,

- This graph consists of three vertices and three edges.
- There are neither self loops nor parallel edges.
- Therefore, it is a simple graph.

17. Multi Graph

- A graph having no self loops but having parallel edge(s) in it is called as a multi graph.

Example-



Example of Multi Graph

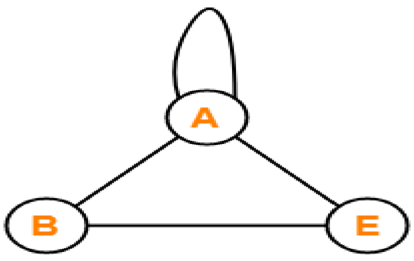
Here,

- This graph consists of three vertices and four edges out of which one edge is a parallel edge.
- There are no self loops but a parallel edge is present.
- Therefore, it is a multi graph.

18. Pseudo Graph

- A graph having no parallel edges but having self loop(s) in it is called as a pseudo graph.

Example-



Example of Pseudo Graph

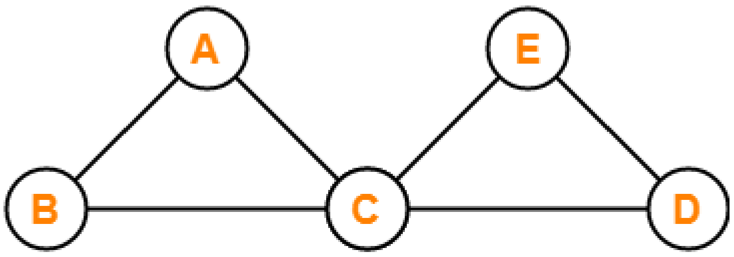
Here,

- This graph consists of three vertices and four edges out of which one edge is a self loop.
- There are no parallel edges but a self loop is present.
- Therefore, it is a pseudo graph.

19. Euler Graph

- Euler Graph is a connected graph in which all the vertices are even degree.

Example-



Example of Euler Graph

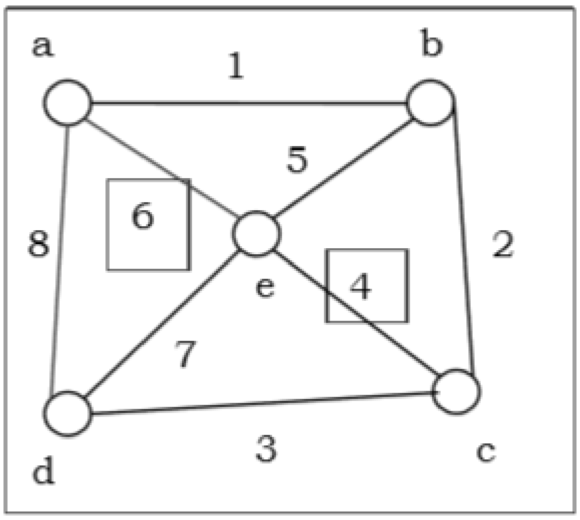
Here,

- This graph is a connected graph.
- The degree of all the vertices is even.
- Therefore, it is an Euler graph.

20. Hamiltonian Graph

- A connected graph G is called Hamiltonian graph if there is a cycle which includes every vertex of G and the cycle is called Hamiltonian cycle. Hamiltonian walk in graph G is a walk that passes through each vertex exactly once.

Example-



In above example, sum of degree of a and c vertices is 6 and is greater than total vertices, 5 using Ore's theorem, it is an Hamiltonian Graph.

Graph Representations

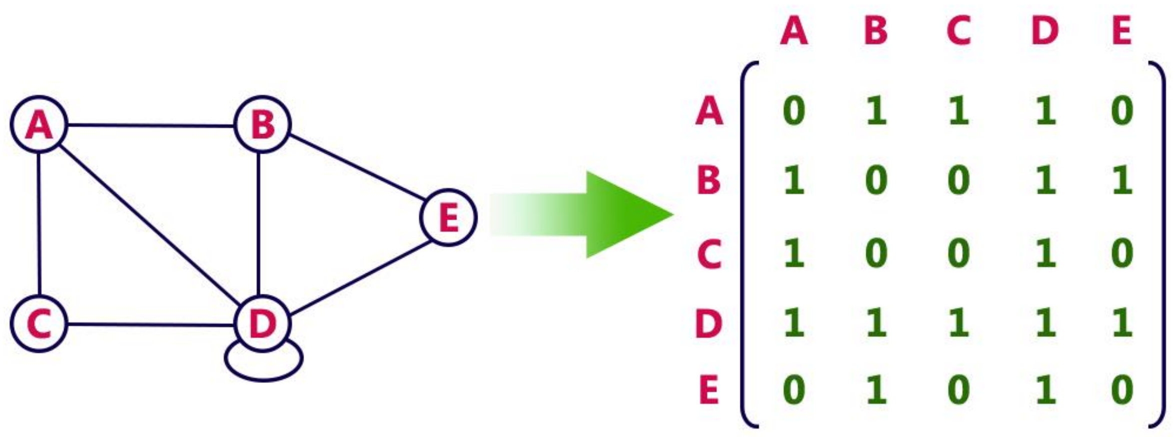
Graph data structure is represented using following representations...

1. Adjacency Matrix
2. Incidence Matrix
3. Adjacency List

Adjacency Matrix

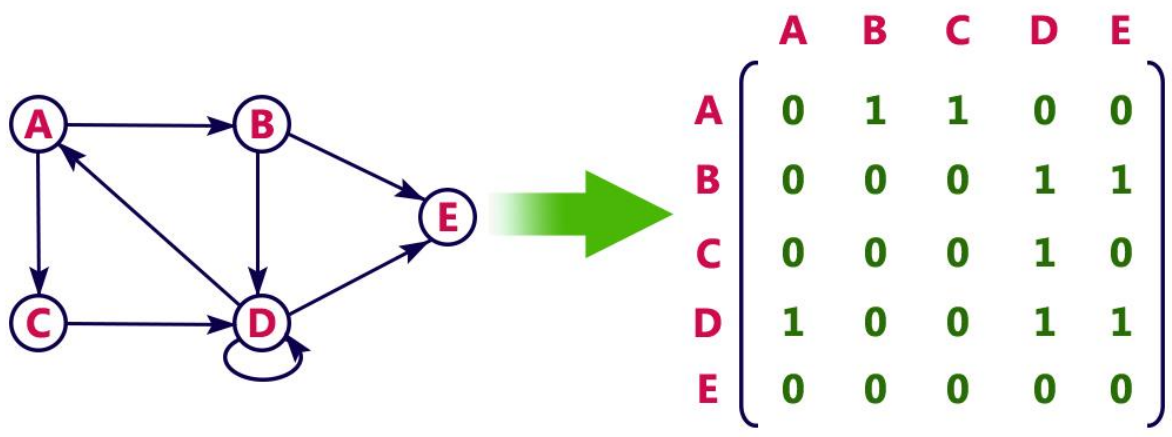
- In this representation, the graph is represented using a matrix of size total number of vertices by a total number of vertices.
- That means a graph with 4 vertices is represented using a matrix of size 4X4.
- In this matrix, both rows and columns represent vertices.
- This matrix is filled with either 1 or 0.
- Here, 1 represents that there is an edge from row vertex to column vertex and 0 represents that there is no edge from row vertex to column vertex.

For example, consider the following undirected graph representation...



$$\begin{array}{c|ccccc|} & A & B & C & D & E \\ A & 0 & 1 & 1 & 1 & 0 \\ B & 1 & 0 & 0 & 1 & 1 \\ C & 1 & 0 & 0 & 1 & 0 \\ D & 1 & 1 & 1 & 1 & 1 \\ E & 0 & 1 & 0 & 1 & 0 \end{array}$$

Directed graph representation...

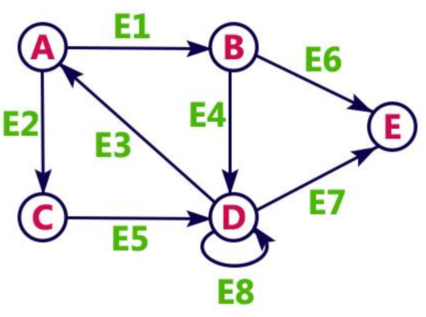


$$\begin{array}{c|ccccc|} & A & B & C & D & E \\ A & 0 & 1 & 1 & 0 & 0 \\ B & 0 & 0 & 0 & 1 & 1 \\ C & 0 & 0 & 0 & 1 & 0 \\ D & 1 & 0 & 0 & 1 & 1 \\ E & 0 & 0 & 0 & 0 & 0 \end{array}$$

Incidence Matrix

- In this representation, the graph is represented using a matrix of size total number of vertices by a total number of edges.
- That means graph with 4 vertices and 6 edges is represented using a matrix of size 4X6.
- In this matrix, rows represent vertices and columns represents edges.
- This matrix is filled with 0 or 1 or -1.
- Here, 0 represents that the row edge is not connected to column vertex, 1 represents that the row edge is connected as the outgoing edge to column vertex and -1 represents that the row edge is connected as the incoming edge to column vertex.

For example, consider the following directed graph representation...

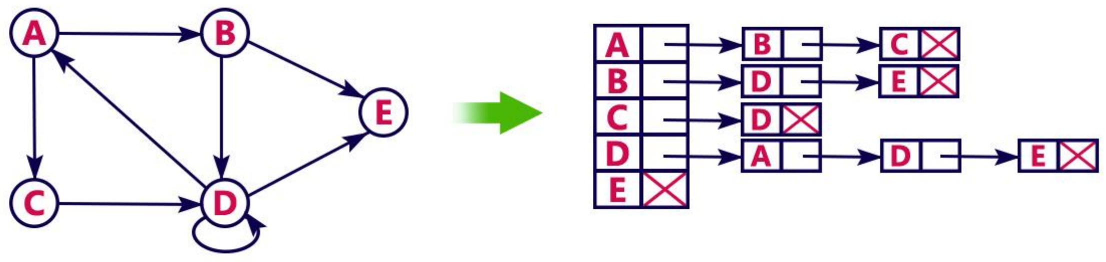


$$\begin{array}{c|cccccccc}
 & E1 & E2 & E3 & E4 & E5 & E6 & E7 & E8 \\
\hline
A & 1 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\
B & -1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
C & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 \\
D & 0 & 0 & 1 & -1 & -1 & 0 & 1 & 1 \\
E & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 0
\end{array}$$

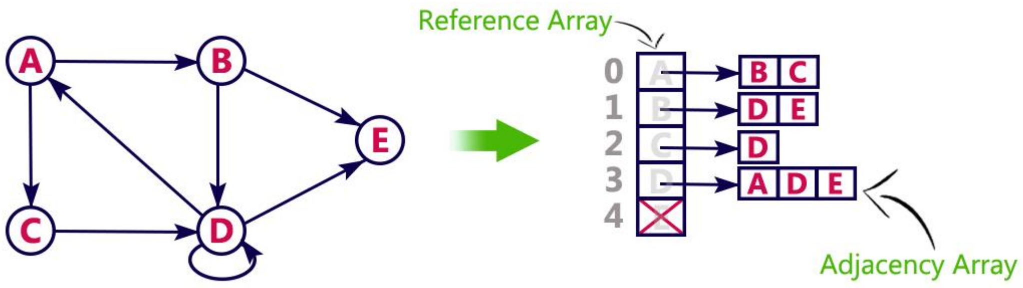
Adjacency List

In this representation, every vertex of a graph contains list of its adjacent vertices.

For example, consider the following directed graph representation implemented using linked list...



This representation can also be implemented using an array as follows..


Reference Array
Adjacency Array
A
B
C
D
E
0
1
2
3
4
B C
D E
D
A D E