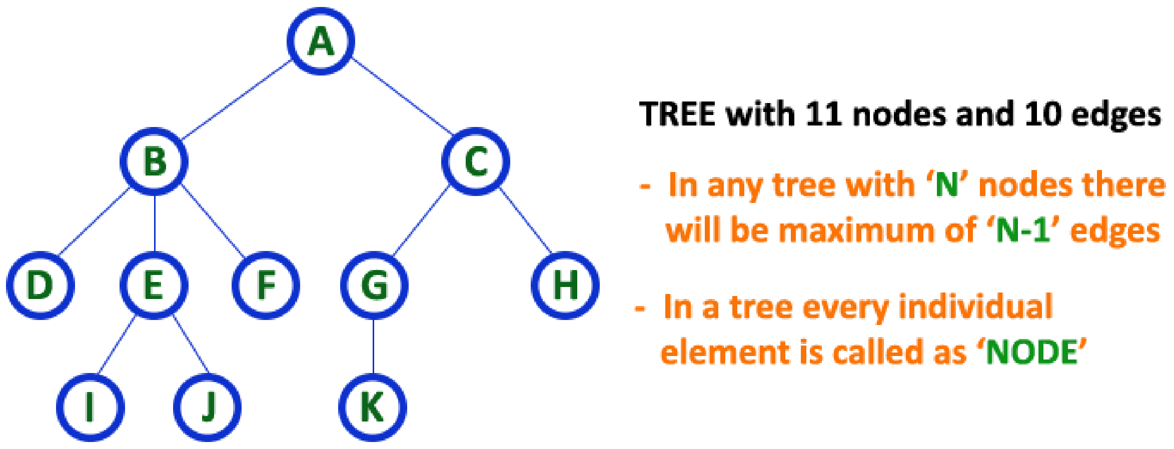
Tree Terminology

A tree is a hierarchical data structure defined as a collection of nodes. Nodes represent value and nodes are connected by edges. A tree has the following properties:

1. The tree has one node called root. The tree originates from this, and hence it does not have any parent.
2. Each node has one parent only but can have multiple children.
3. Each node is connected to its children via edge.

Following diagram explains various terminologies used in a tree structure.

Example

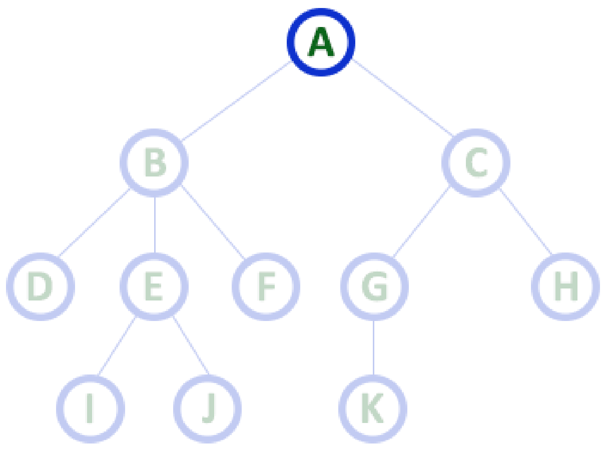

A
B
C
D
E
F
G
H
I
J
K
TREE with 11 nodes and 10 edges
- In any tree with 'N' nodes there will be maximum of 'N-1' edges
- In a tree every individual element is called as 'NODE'

Terminology

In a tree data structure, we use the following terminology...

1. Root

In a tree data structure, the first node is called as **Root Node**. Every tree must have a root node. We can say that the root node is the origin of the tree data structure. In any tree, there must be only one root node. We never have multiple root nodes in a tree.

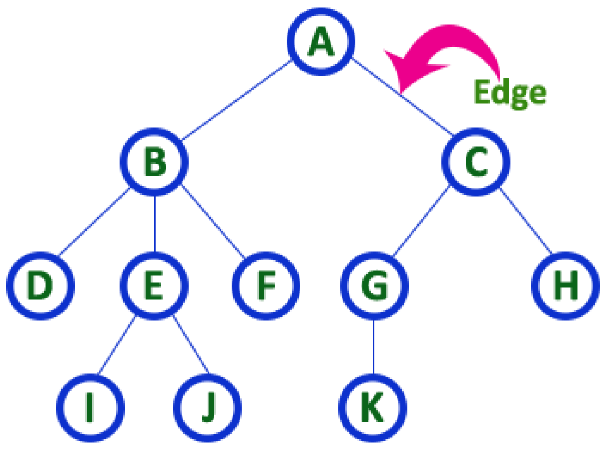


Here 'A' is the 'root' node

- In any tree the first node is called as ROOT node

2. Edge

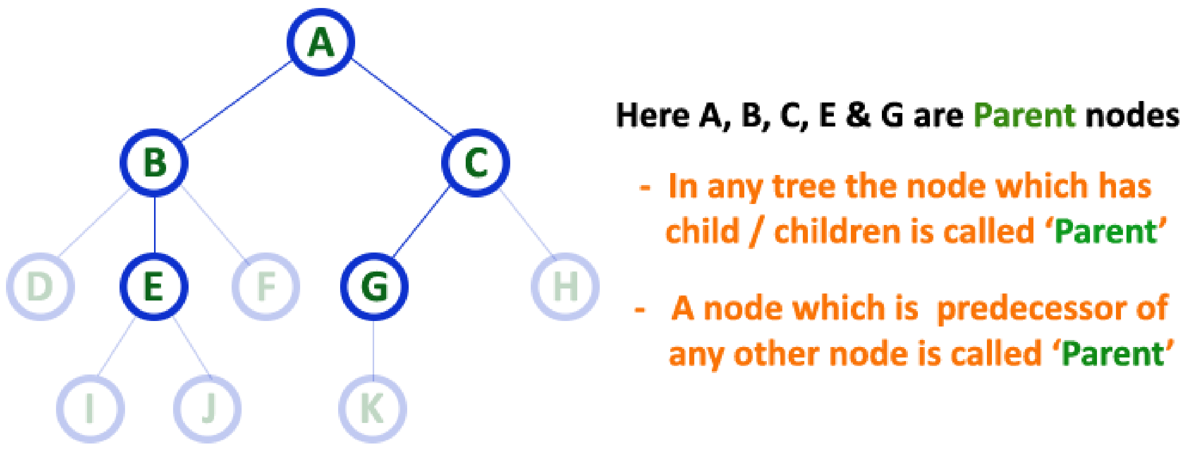
In a tree data structure, the connecting link between any two nodes is called as **EDGE.** In a tree with '**N**' number of nodes there will be a maximum of '**N-1**' number of edges.



- In any tree, 'Edge' is a connecting link between two nodes.

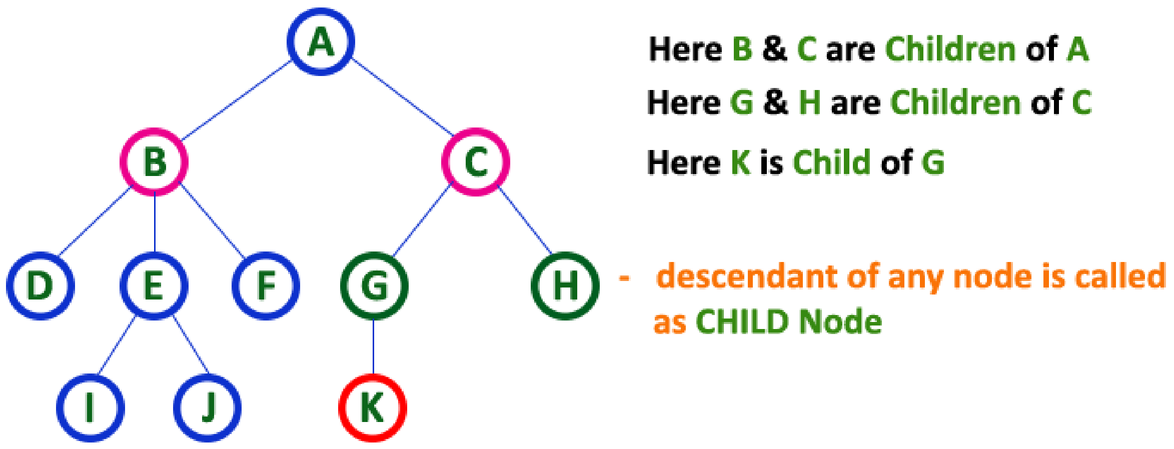
3. Parent

In a tree data structure, the node which is a predecessor of any node is called as **PARENT NODE.** In simple words, the node which has a branch from it to any other node is called a parent node. Parent node can also be defined as "**The node which has child / children**".


A
B
C
D
E
F
G
H
I
J
K
Here A, B, C, E & G are Parent nodes
- In any tree the node which has child / children is called 'Parent'
- A node which is predecessor of any other node is called 'Parent'

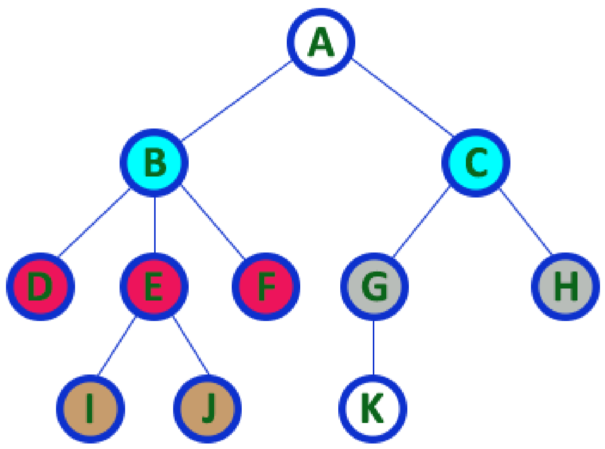
4. Child

In a tree data structure, the node which is descendant of any node is called as **CHILD Node**. In simple words, the node which has a link from its parent node is called as child node. In a tree, any parent node can have any number of child nodes. In a tree, all the nodes except root are child nodes.


A
B
C
D
E
F
G
H
I
J
K
Here B & C are Children of A
Here G & H are Children of C
Here K is Child of G
- descendant of any node is called as CHILD Node

5. Siblings

In a tree data structure, nodes which belong to same Parent are called as **SIBLINGS**. In simple words, the nodes with the same parent are called Sibling nodes.



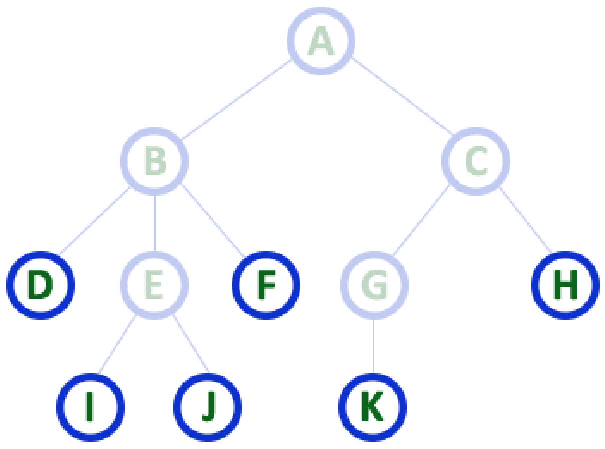
Here B & C are Siblings
Here D E & F are Siblings
Here G & H are Siblings
Here I & J are Siblings

- In any tree the nodes which has same Parent are called 'Siblings'
- The children of a Parent are called 'Siblings'

6. Leaf

In a tree data structure, the node which does not have a child is called as **LEAF Node**. In simple words, a leaf is a node with no child.

In a tree data structure, the leaf nodes are also called as **External Nodes**. External node is also a node with no child. In a tree, leaf node is also called as '**Terminal**' node.



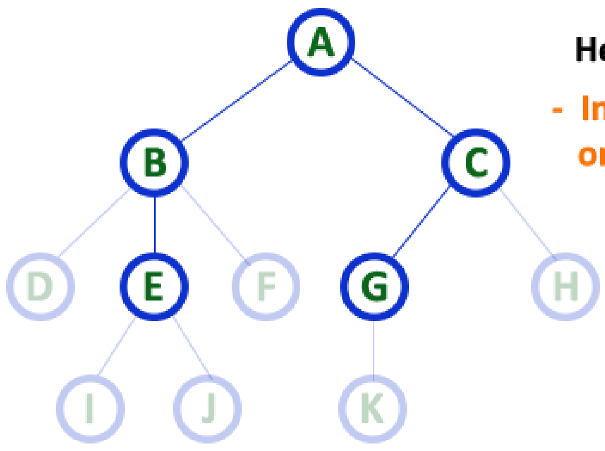
Here D, I, J, F, K & H are Leaf nodes

- In any tree the node which does not have children is called 'Leaf'
- A node without successors is called a 'leaf' node

7. Internal Nodes

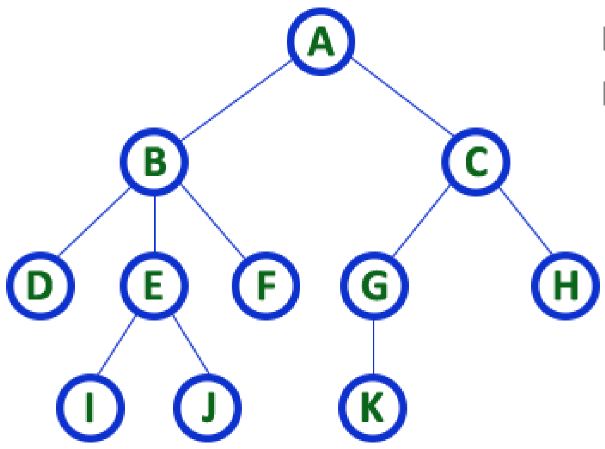
In a tree data structure, the node which has at least one child is called as **INTERNAL Node**. In simple words, an internal node is a node with atleast one child.

In a tree data structure, nodes other than leaf nodes are called as **Internal Nodes. The root node is also said to be Internal Node** if the tree has more than one node. Internal nodes are also called as '**Non-Terminal**' nodes.


A
B
C
D
E
F
G
H
I
J
K
Here A, B, C, E & G are Internal nodes
- In any tree the node which has atleast one child is called 'Internal' node
- Every non-leaf node is called as 'Internal' node

8. Degree

In a tree data structure, the total number of children of a node is called as **DEGREE** of that Node. In simple words, the Degree of a node is total number of children it has. The highest degree of a node among all the nodes in a tree is called as '**Degree of Tree**'



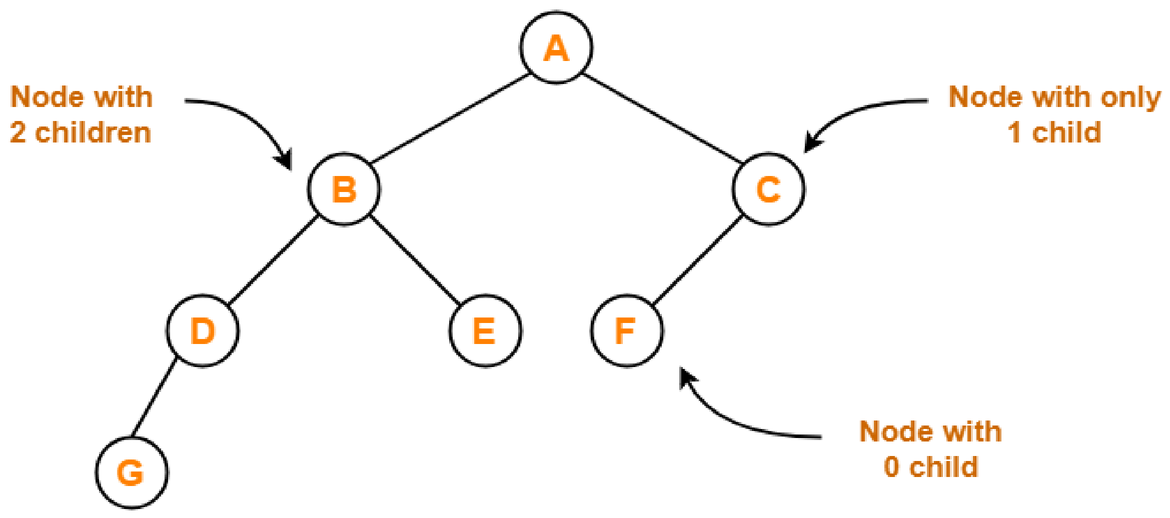
Binary Tree-

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.

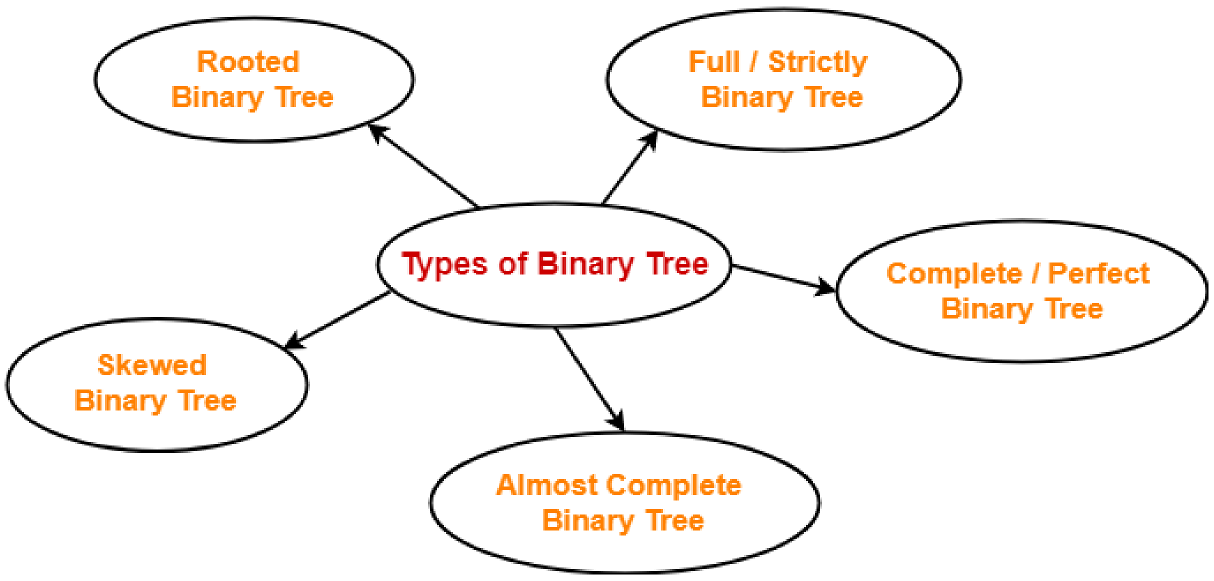
Example-



Binary Tree Example

Types of Binary Trees-

Binary trees can be of the following types-

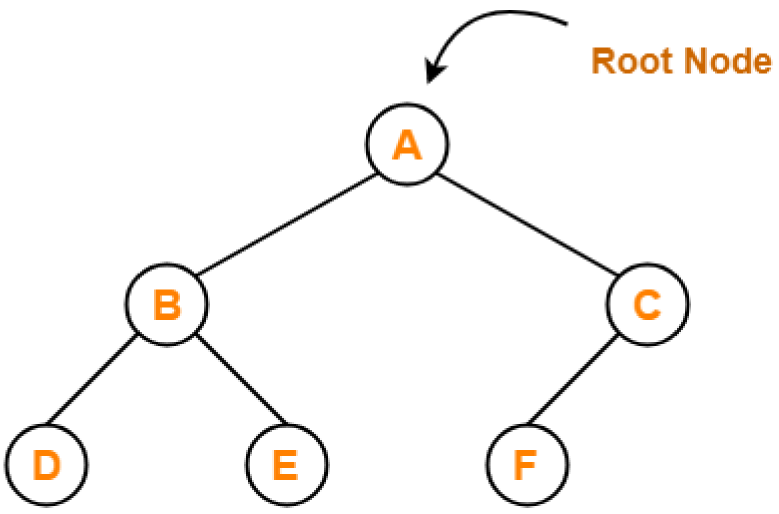

Types of Binary Tree
Rooted Binary Tree
Full / Strictly Binary Tree
Complete / Perfect Binary Tree
Almost Complete Binary Tree
Skewed Binary Tree

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-

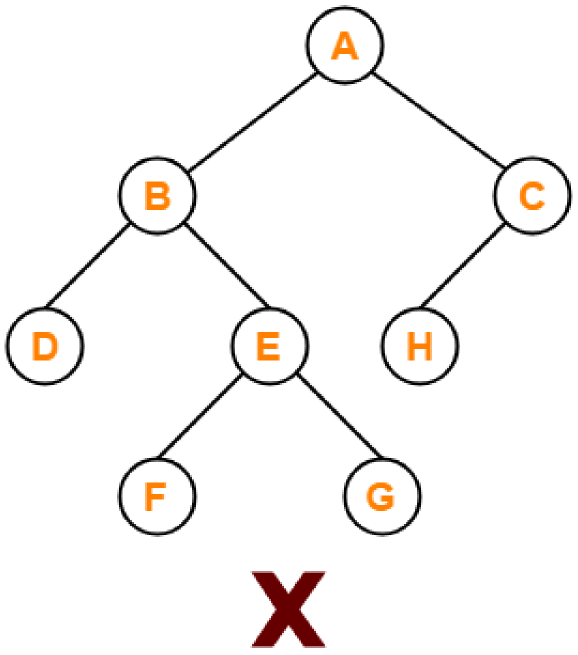


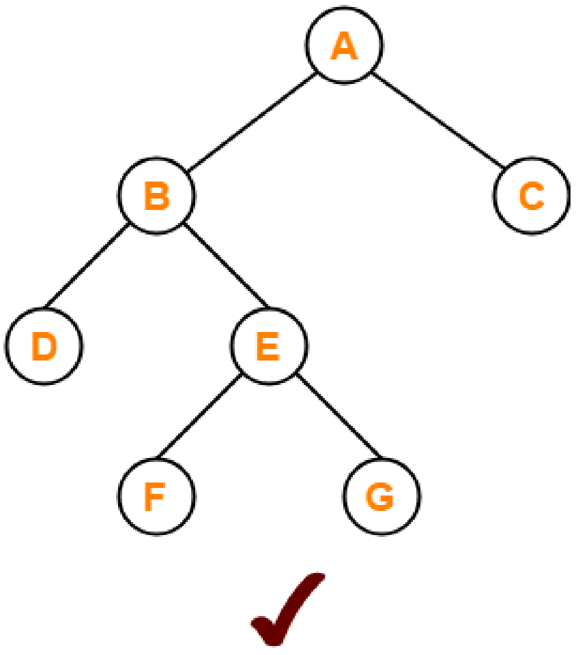
Rooted Binary Tree

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.

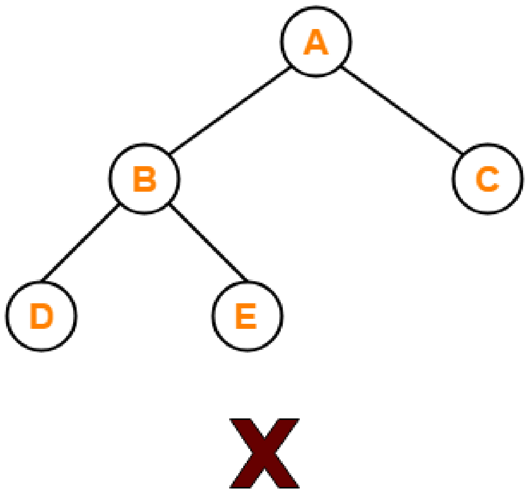
3. Complete / Perfect Binary Tree-

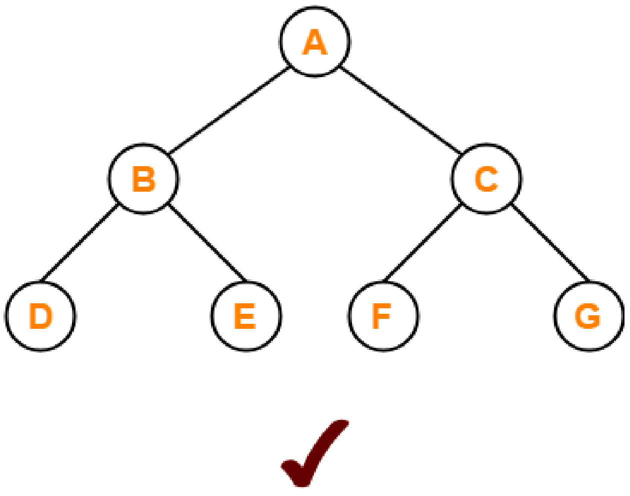
A complete binary tree is a binary tree that satisfies the following 2 properties-

- 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,

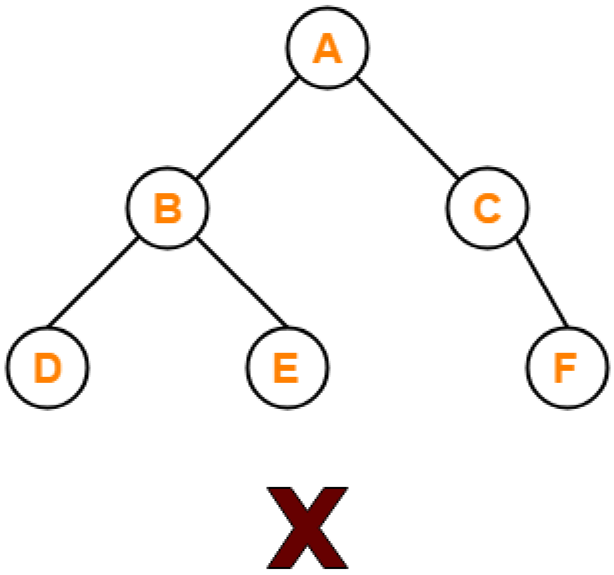
- First binary tree is not a complete binary tree.
- This is because all the leaf nodes are not at the same level.

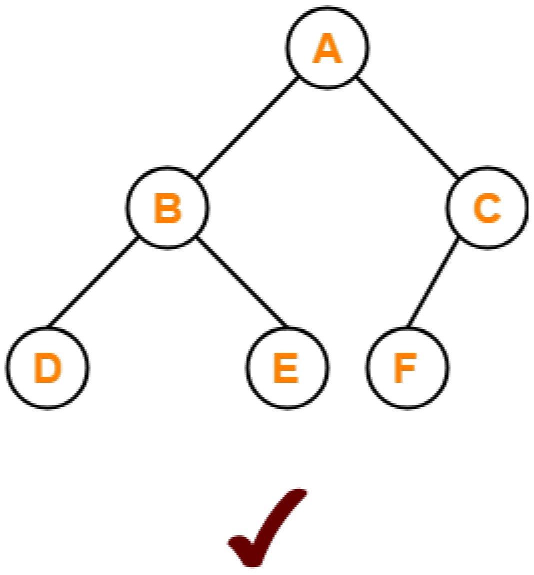
4. Almost Complete Binary Tree-

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-





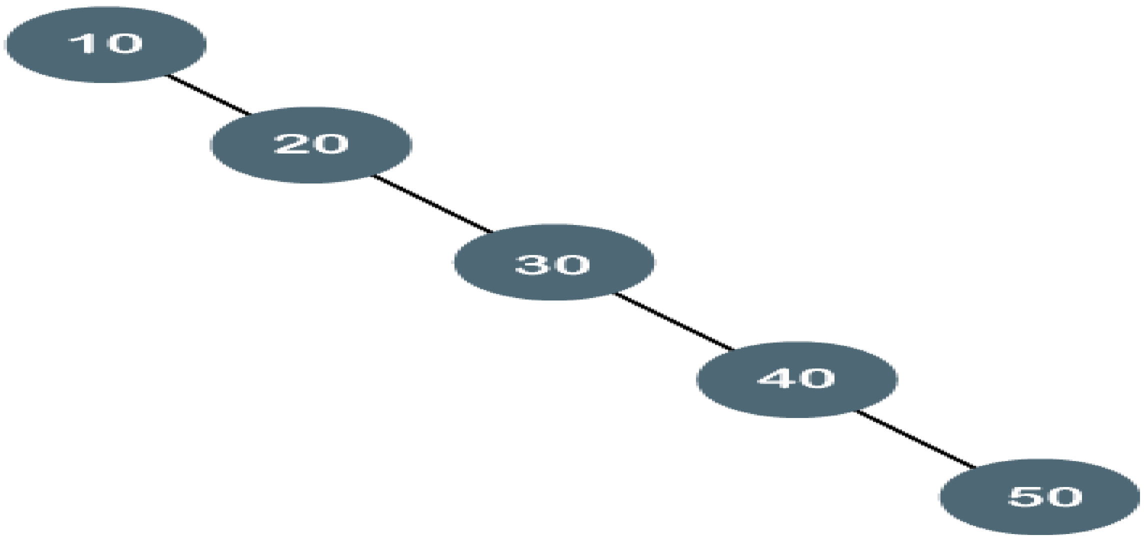
Here,

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

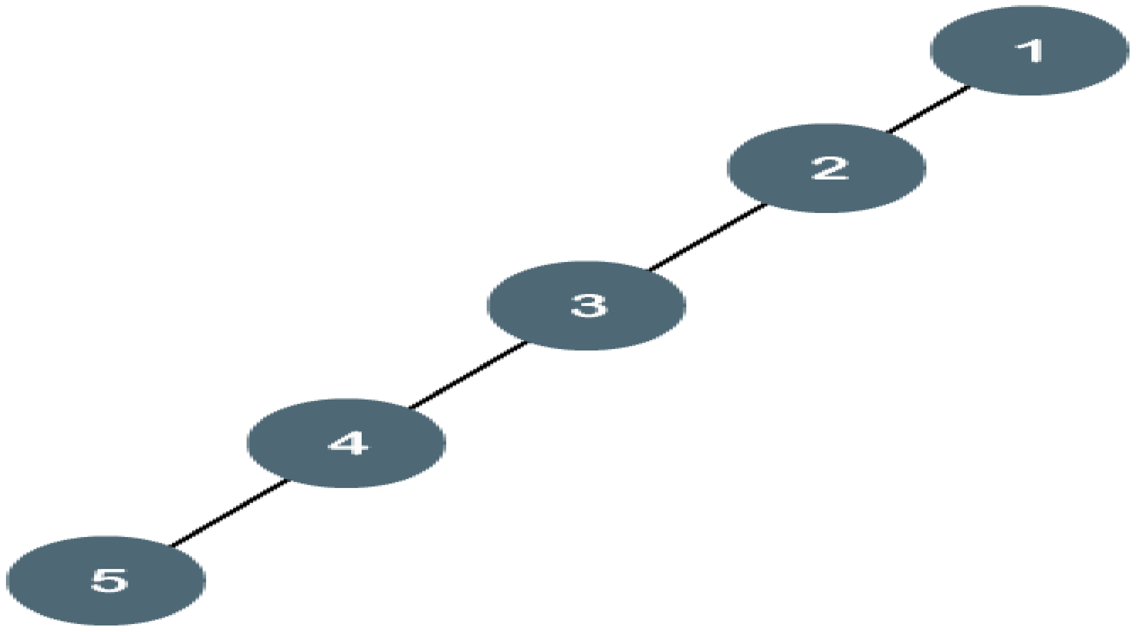
5. Skewed Binary Tree-

A skewed binary tree is a pathological/degenerate tree in which the tree is either dominated by the left nodes or the right nodes. Thus, there are two types of skewed binary tree: left-skewed binary tree and right-skewed binary tree.

Example-



The above tree is a degenerate binary tree because all the nodes have only one child. It is also known as a right-skewed tree as all the nodes have a right child only.



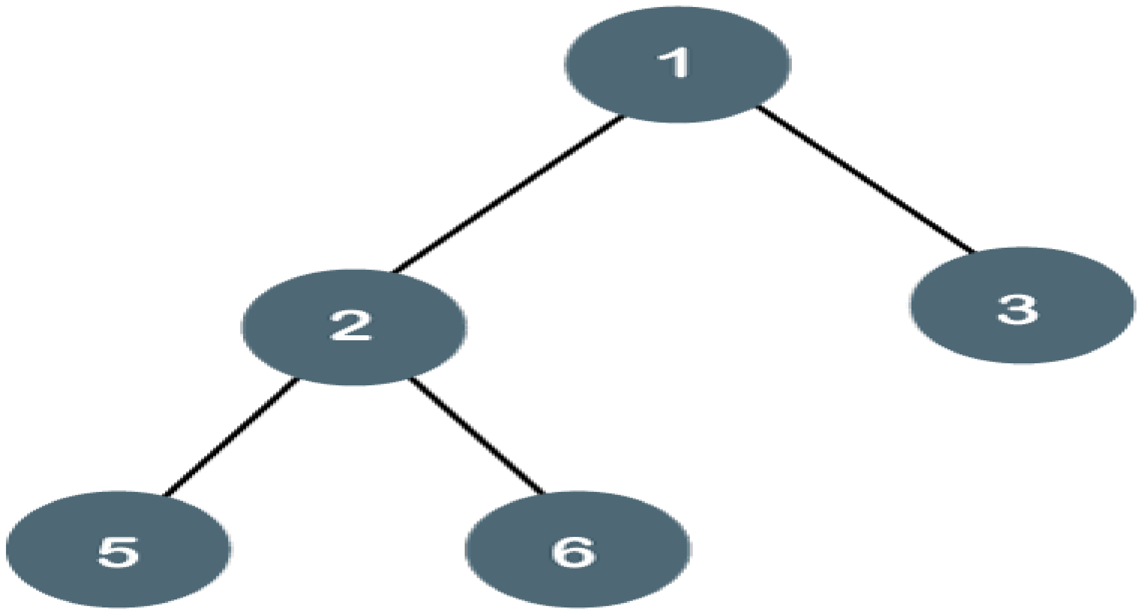
The above tree is also a degenerate binary tree because all the nodes have only one child. It is also known as a left-skewed tree as all the nodes have a left child only.

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...



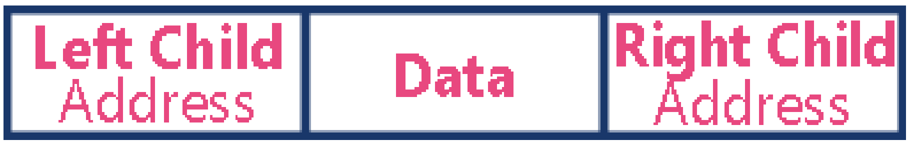
1. Array Representation of Binary Tree

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...

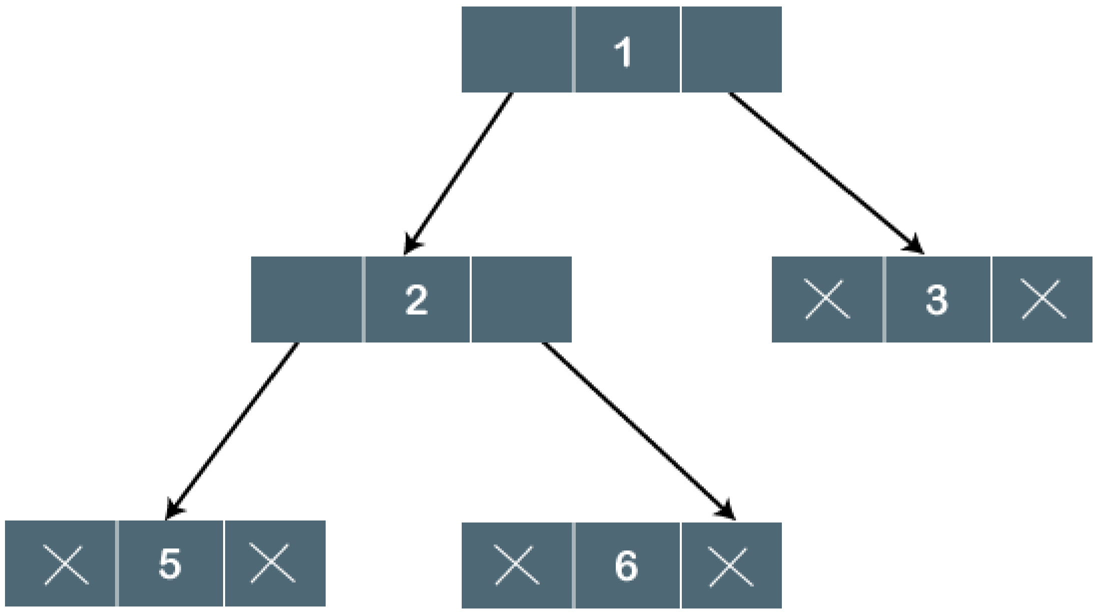
1	2	3	5	6

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 Traversals

Tree traversal means **traversing** or **visiting** each node of a tree.

The following are the three different ways of traversal:

- **Inorder traversal**
- **Preorder traversal**
- **Postorder traversal**

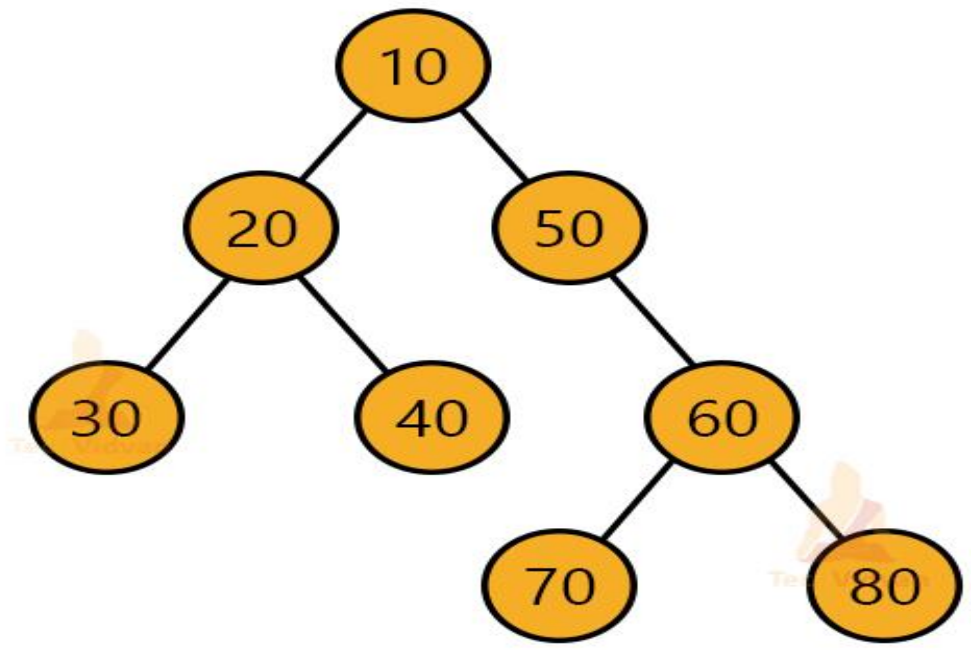
Inorder Traversal

An inorder traversal is a traversal technique that follows the policy, i.e., Left Root Right

Algorithm:

1. First, visit all the nodes in the left subtree
2. Then the root node
3. Visit all the nodes in the right subtree

Consider the following tree:



After performing the inorder traversal, the order of nodes will be 30→20→40→10→50→70→60→80.

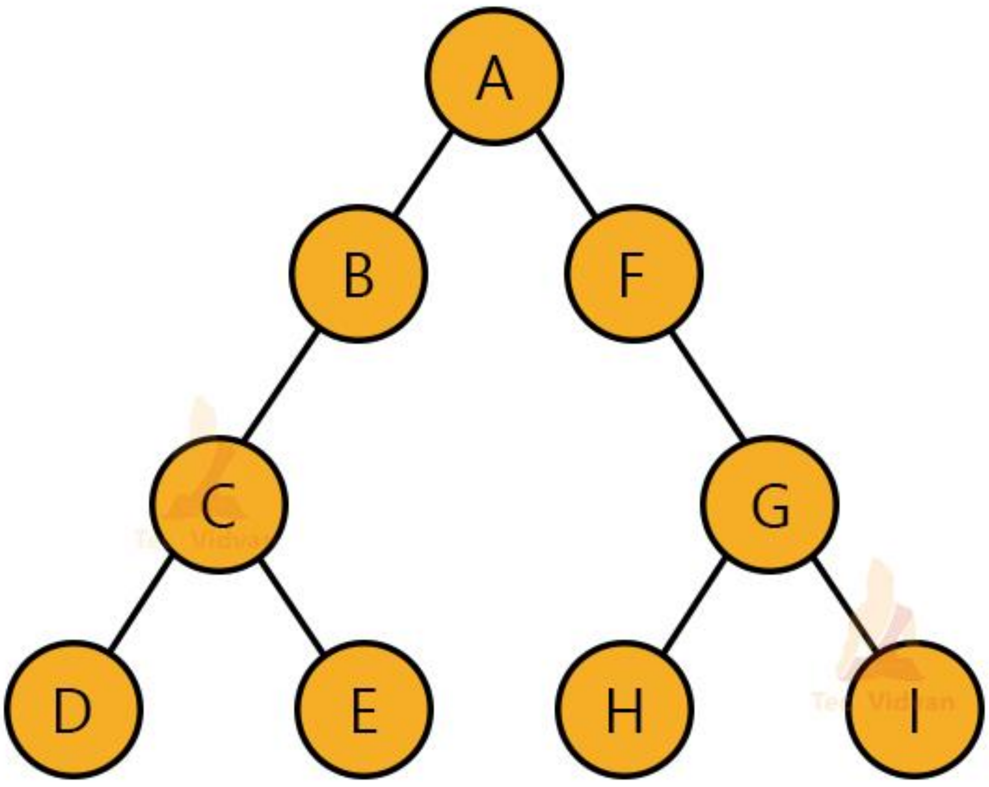
Preorder Traversal

A preorder traversal is a traversal technique that follows the policy, i.e., **Root Left Right**.

Algorithm:

1. Visit root node
2. Visit all the nodes in the left subtree
3. Visit all the nodes in the right subtree

Consider the following tree:



The order of visiting the nodes will be: A→B→C→D→E→F→G→H→I.

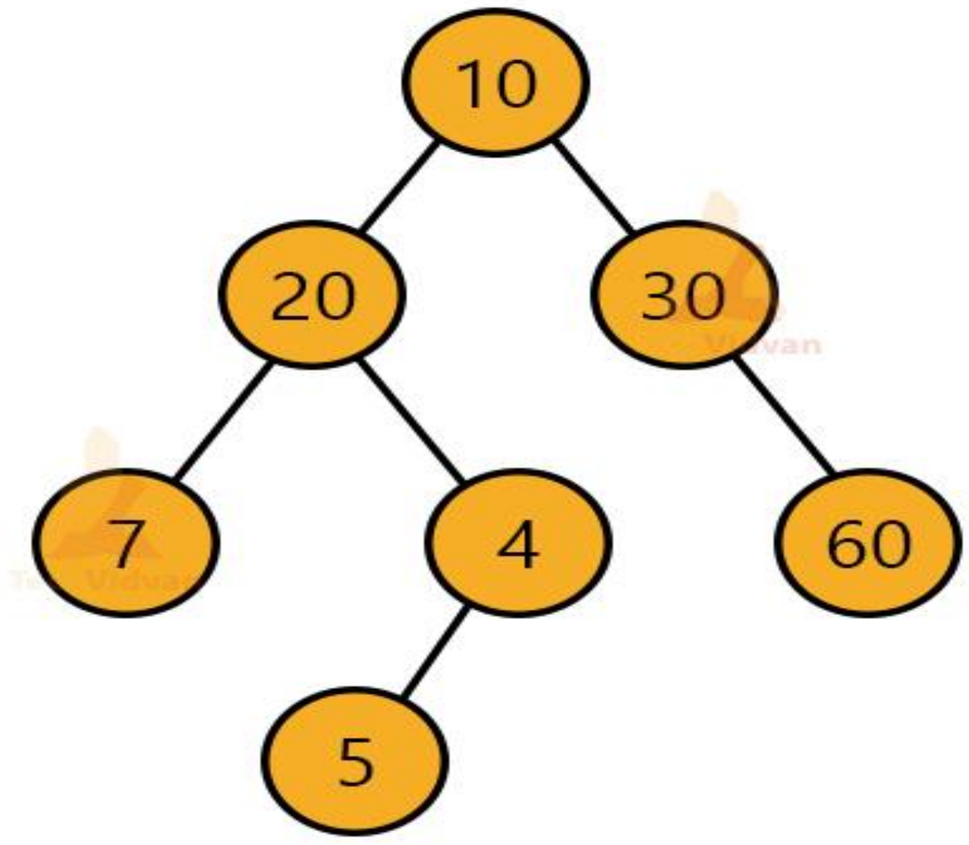
Postorder Traversal

A Postorder traversal is a traversal technique that follows the policy, i.e., **Left Right Root**.

Algorithm:

1. Visit all the nodes in the left subtree
2. Visit all the nodes in the right subtree
3. Visit the root node

Consider the following tree:



The postorder traversal will be 7→5→4→20→60→30→10.

Graph

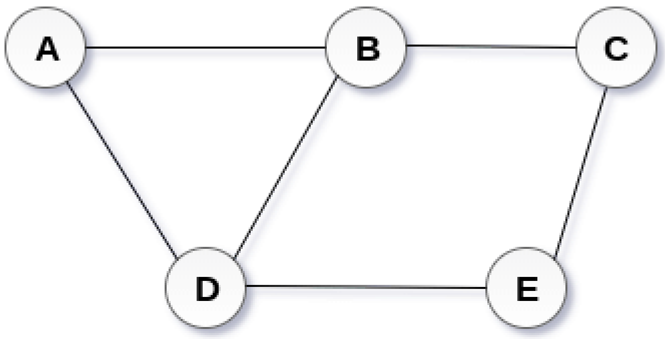
A graph can be defined as group of vertices and edges that are used to connect these vertices.

Definition

A graph G can be defined as an ordered set G(V, E) where V(G) represents the set of vertices and E(G) represents the set of edges which are used to connect these vertices.

Directed and Undirected Graph

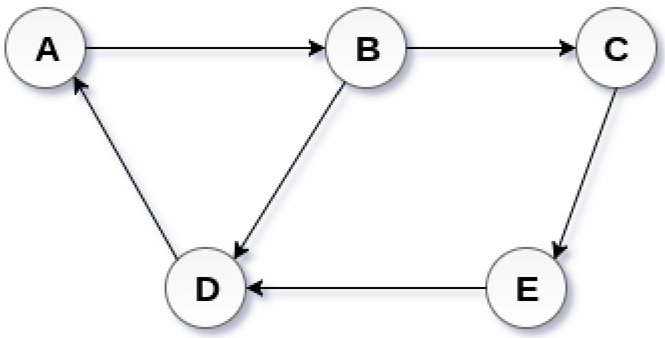
A graph can be directed or undirected. However, in an undirected graph, edges are not associated with the directions with them. An undirected graph is shown in the above figure since its edges are not attached with any of the directions. If an edge exists between vertex A and B then the vertices can be traversed from B to A as well as A to B.



Undirected Graph

In a directed graph, edges form an ordered pair. Edges represent a specific path from some vertex A to another vertex B. Node A is called initial node while node B is called terminal node.

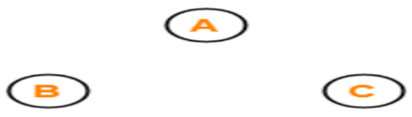
A directed graph is shown in the following figure.



Directed Graph

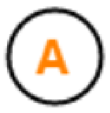
Null Graph

The Null Graph is also known as the order zero graph. The term "null graph" refers to a graph with an empty edge set. In other words, a null graph has no edges, and the null graph is present with only isolated vertices in the graph.



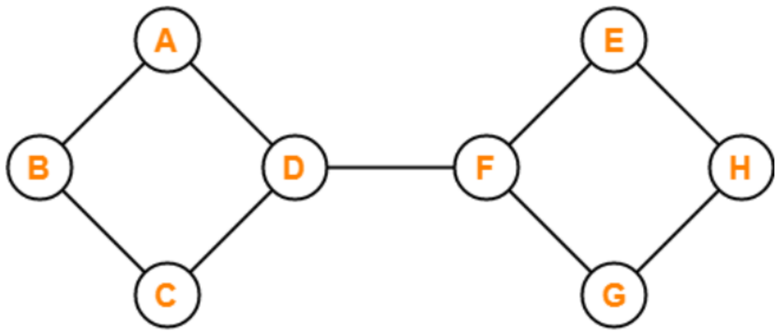
Trivial Graph

A graph is called a trivial graph if it has only one vertex present in it. The trivial graph is the smallest possible graph that can be created with the least number of vertices that is one vertex only**.**



Connected Graph

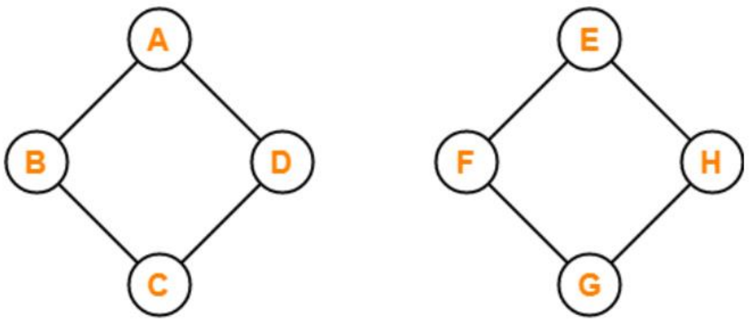
For a graph to be labelled as a connected graph, there must be at least a single path between every pair of the graph's vertices. In other words, we can say that if we start from one vertex, we should be able to move to any of the vertices that are present in that particular graph, which means there exists at least one path between all the vertices of the graph.



Disconnected Graph

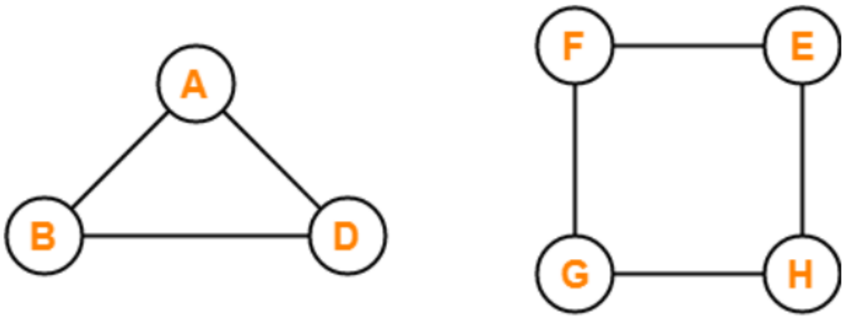
A graph is said to be a disconnected graph where there does not exist any path between at least one pair of vertices**.**

In other words, we can say that if we start from any one of the vertices of the graph and try to move to the remaining present vertices of the graph and there exists not even a single path to move to that vertex, then it is the case of the disconnected graph. If any one of such a pair of vertices doesn't have a path between them, it is called a disconnected graph.



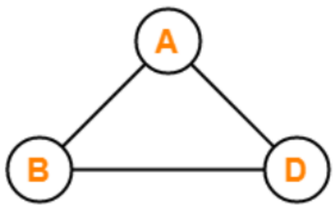
Regular Graph

For a graph to be called a regular, it should satisfy one primary condition: all graph vertices should have the same degree. By the degree of vertices, we mean the number of nodes associated with a particular vertex. If all the graph nodes have the same degree value, then the graph is called a regular graph. If all the vertices of a graph have the degree value of 6, then the graph is called a 6-regular graph. If all the vertices in a graph are of degree 'k', then it is called a "k-regular graph".

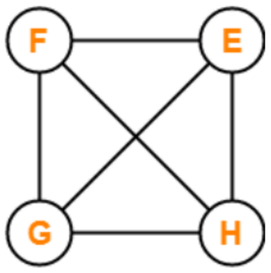


Complete Graph

A graph is said to be a complete graph if, for all the vertices of the graph, there exists an edge between every pair of the vertices. In other words, we can say that all the vertices are connected to the rest of all the vertices of the graph.



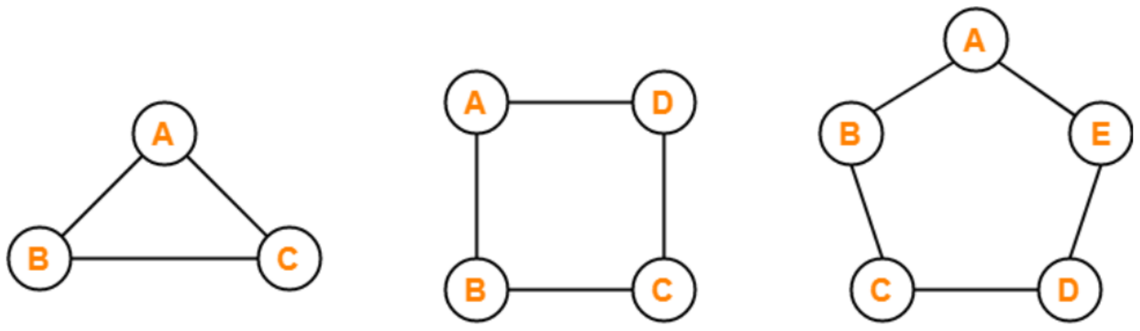
K_3



K_4

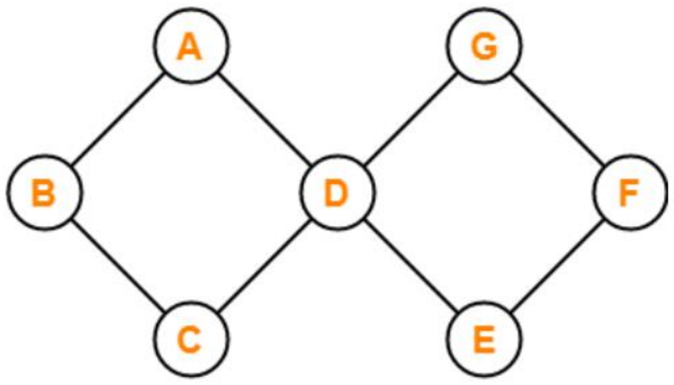
Cycle Graph

If a graph with many vertices greater than three and edges form a cycle, then the graph is called a cycle graph. In a graph of cycle type, the degree of all the vertices of the cycle graph will be 2.



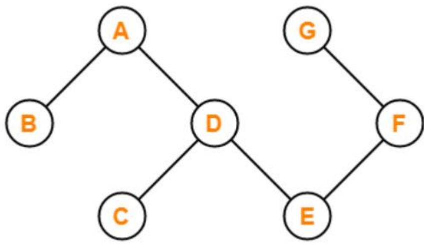
Cyclic Graph

For a graph to be called a cyclic graph, it should consist of at least one cycle. If a graph has a minimum of one cycle present, it is called a cyclic graph**.**



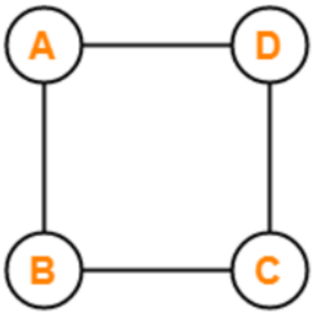
Acyclic Graph

A graph is called an acyclic graph if zero cycles are present, and an acyclic graph is the complete opposite of a cyclic graph.



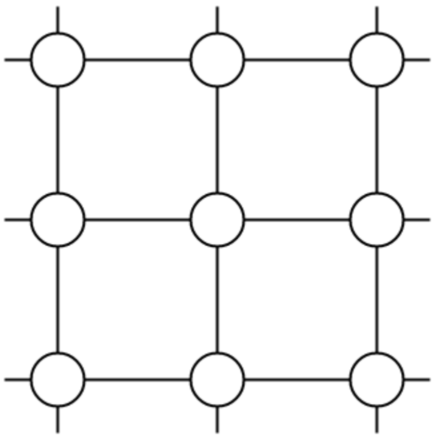
Finite Graph

If the number of vertices and the number of edges that are present in a graph are finite in number, then that graph is called a finite graph.



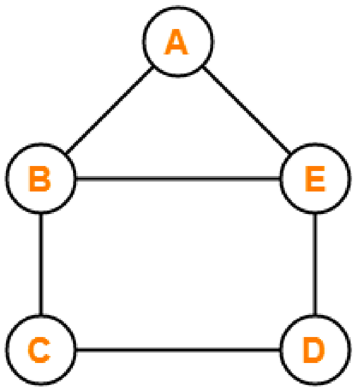
Infinite Graph

If the number of vertices in the graph and the number of edges in the graph are infinite in number, that means the vertices and the edges of the graph cannot be counted, then that graph is called an infinite graph.



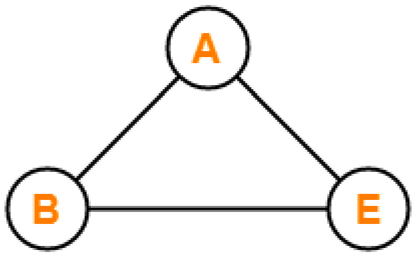
Planar Graph

A graph is called a planar graph if that graph can be drawn in a single plane with any two of the edges intersecting each other.



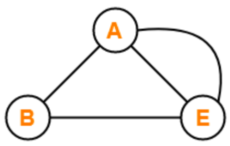
Simple Graph

A graph is said to be a simple graph if the graph doesn't consist of no self-loops and no parallel edges in the graph.



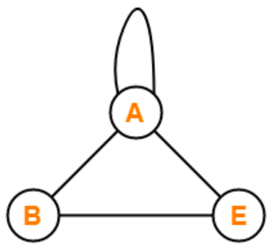
Multi Graph

A graph is said to be a multigraph if the graph doesn't consist of any self-loops, but parallel edges are present in the graph. If there is more than one edge present between two vertices, then that pair of vertices is said to be having parallel edges.



Pseudo Graph

If a graph consists of no parallel edges, but self-loops are present in a graph, it is called a pseudo graph



Graph Terminology

Path

A path can be defined as the sequence of nodes that are followed in order to reach some terminal node V from the initial node U.

Closed Path

A path will be called as closed path if the initial node is same as terminal node. A path will be closed path if $V_0=V_N$.

Weighted Graph

In a weighted graph, each edge is assigned with some data such as length or weight. The weight of an edge e can be given as w(e) which must be a positive (+) value indicating the cost of traversing the edge.

Loop

An edge that is associated with the similar end points can be called as Loop.

Adjacent Nodes

If two nodes u and v are connected via an edge e, then the nodes u and v are called as neighbours or adjacent nodes.

Degree of the Node

A degree of a node is the number of edges that are connected with that node. A node with degree 0 is called as isolated node.

Graph Representation

By Graph representation, we simply mean the technique which is to be used in order to store some graph into the computer's memory.

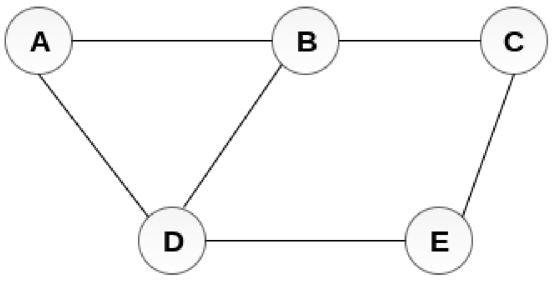
There are two ways to store Graph into the computer's memory.

1. Sequential Representation

In sequential representation, we use adjacency matrix to store the mapping represented by vertices and edges. In adjacency matrix, the rows and columns are represented by the graph vertices. A graph having n vertices, will have a dimension **n x n**.

An entry M_{ij} in the adjacency matrix representation of an undirected graph G will be 1 if there exists an edge between V_i and V_j.

An undirected graph and its adjacency matrix representation is shown in the following figure.

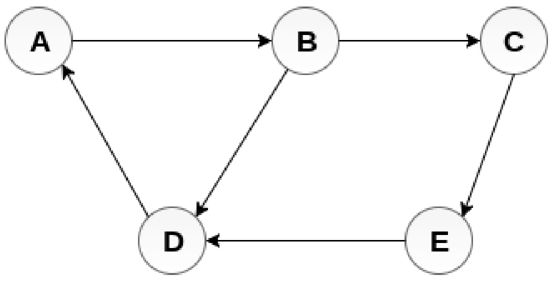


Undirected Graph

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

Adjacency Matrix

A directed graph and its adjacency matrix representation is shown in the following figure.

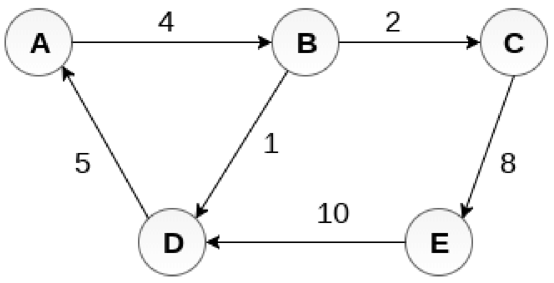


Directed Graph

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

Adjacency Matrix

The weighted directed graph along with the adjacency matrix representation is shown in the following figure.



Weighted Directed Graph

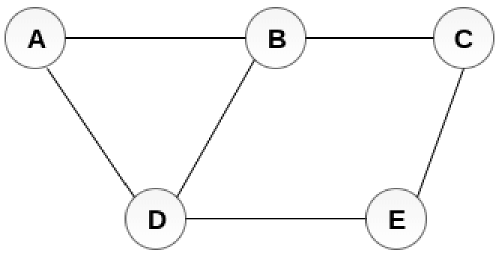
$$\begin{array}{c|ccccc|} & A & B & C & D & E \\ \hline A & 0 & 4 & 0 & 0 & 0 \\ B & 0 & 0 & 2 & 1 & 0 \\ C & 0 & 0 & 0 & 0 & 8 \\ D & 5 & 0 & 0 & 0 & 0 \\ E & 0 & 0 & 0 & 10 & 0 \end{array}$$

Adjacency Matrix

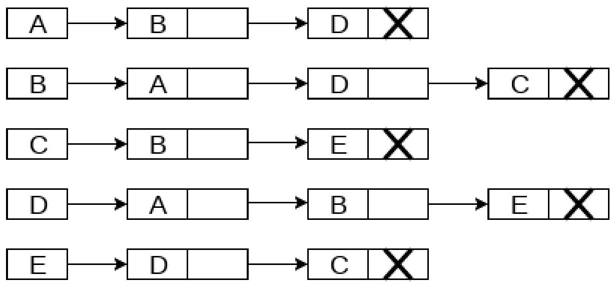
Linked Representation

In the linked representation, an adjacency list is used to store the Graph into the computer's memory.

Consider the undirected graph shown in the following figure and check the adjacency list representation.



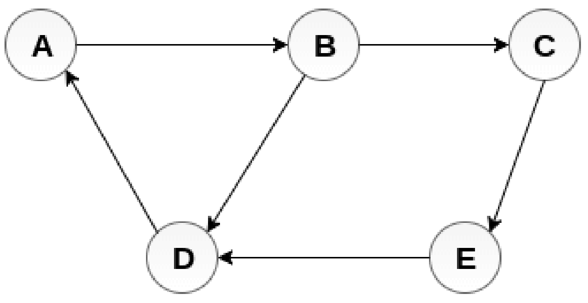
Undirected Graph



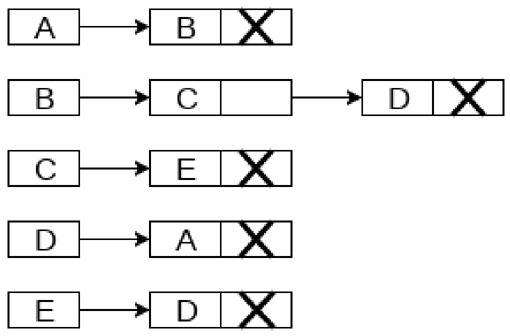
Adjacency List

An adjacency list is maintained for each node present in the graph which stores the node value and a pointer to the next adjacent node to the respective node. If all the adjacent nodes are traversed then store the NULL in the pointer field of last node of the list. The sum of the lengths of adjacency lists is equal to the twice of the number of edges present in an undirected graph.

Consider the directed graph shown in the following figure and check the adjacency list representation of the graph.



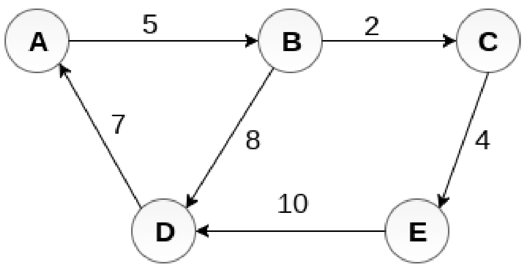
Directed Graph



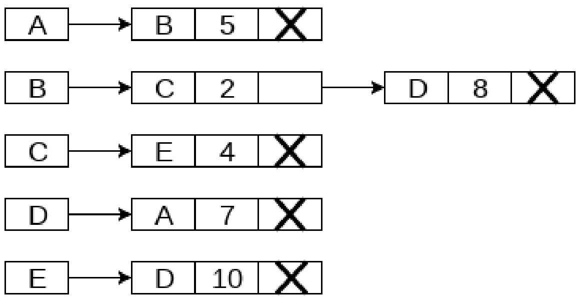
Adjacency List

In a directed graph, the sum of lengths of all the adjacency lists is equal to the number of edges present in the graph.

In the case of weighted directed graph, each node contains an extra field that is called the weight of the node. The adjacency list representation of a directed graph is shown in the following figure.



Weighted Directed Graph



Adjacency List

Graph Traversal

Graph traversal is a technique used for searching a vertex in a graph. The graph traversal is also used to decide the order of vertices is visited in the search process. A graph traversal finds the edges to be used in the search process without creating loops. That means using graph traversal we visit all the vertices of the graph without getting into looping path.

There are two graph traversal techniques and they are as follows...

1. **DFS (Depth First Search)**
2. **BFS (Breadth First Search)**

BFS (Breadth First Search)

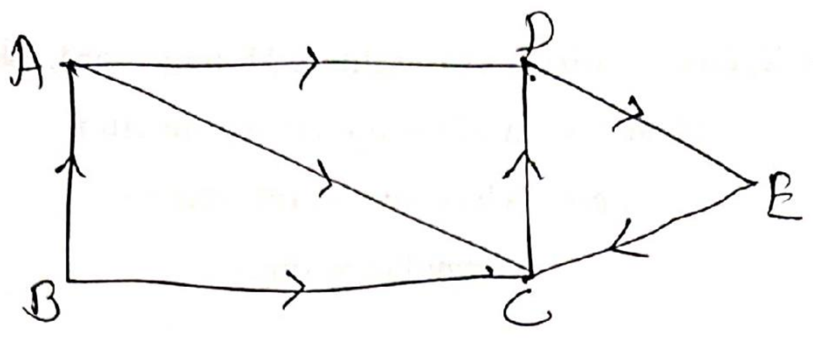
BFS traversal of a graph produces a **spanning tree** as final result. **Spanning Tree** is a graph without loops. We use **Queue data structure** with maximum size of total number of vertices in the graph to implement BFS traversal.

These is an application of queue following is a algorithm which in used to traverse a graph using a BFS

Method .

I. Initialize all nodes to ready state (status=1)
II. Insert a starting node a in a queue change its state to waiting state (radius=2)
III. Repeat step 4&5 until queue in node empty
IV. Delete front node n of queue process it, change its state to process state (radius=3)
V. Insert all neighbour of node n which are in ready state into a queue. And change their state to waiting state.

Example



i. Consider node a as starting node.

ii. Insert in queue and change its states to waiting state queue=A

F=1

R=1

iii. Delete the front element A, from queue print it change its status to process state. Insert all its neighbor which are in ready state in a queue.

Queue: C, D

F=2

R=1, 2, 3

Print A

iv. Delete front element C process it change its status and insert all its neighbor which are in a ready state in a queue.

Q=E

E=3

R=4

Print A, C, D

V . Q:- empty

F=4

R=4

Print :- A, C, D, E

These gives the following as, A, C, D, E

In these traversing order the node B is not present .

Because these is no edge from any node to node B

i.e , node B in a source.

DFS (Depth First Search)

DFS traversal of a graph produces a **spanning tree** as final result. **Spanning Tree** is a graph without loops. We use **Stack data structure** with maximum size of total number of vertices in the graph to implement DFS traversal. It is an application of a stack

1.Initialize the all the nodes to ready state (radius =1)

2. Push the starting node A on to stack. And change it to the waiting state . (status=2)

3.Repeat the steps 4 & 5 until stack is not empty .

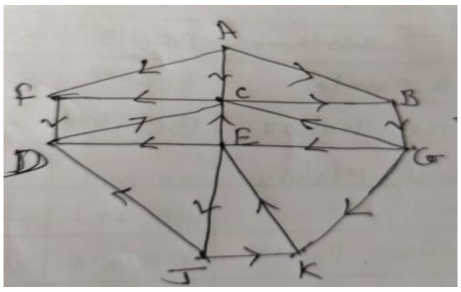
4.Pop the top element of the stack (N) process N. change its status to process state .(Status=3)

5. push all the neighbours of N that are in the ready state on top stack and change there state to waiting state.

6.EXIT.

Back tracking is coming back to the vertex from which we reached the current vertex.

Example



1.Push starting node A on top of the stack and change its state from ready to waiting state. STACK= A

2.Pop the top element of stack. Print it and change its status to process state.

3.Push all the neighbours of A which are in waiting state on the top of the stack.

PRINT=A

STACK=B,C,F

4. Pop the top element F, print it change of F which are in ready state of stack

PRINT=A,F

STACK=B,C,D

5.Pop the top element D. print it change its status. Push all neighbour which are in ready state an top of stack.

PRINT=A,F,D

STACK=B,C

Similarly we perform following steps.

6. print= A,F,D,C

Stack=B

7. Print=A,F,D,C,B

Stack=G

8. Print=A,F,D,C,B,G

Stack=E,K

9. Print=A,F,D,C,B,G,K

Stack=E

10. Print=A,F,D,C,B,G,K,E

Stack=J

11.Print=A,F,D,C,B,G,K,E,J

Stack=Empty

At the end of step stack is empty. The DFS Traversal of above graph is

A, F, D ,C ,B ,G ,K ,E, J