Q1. Explain Tree in Data Structure?

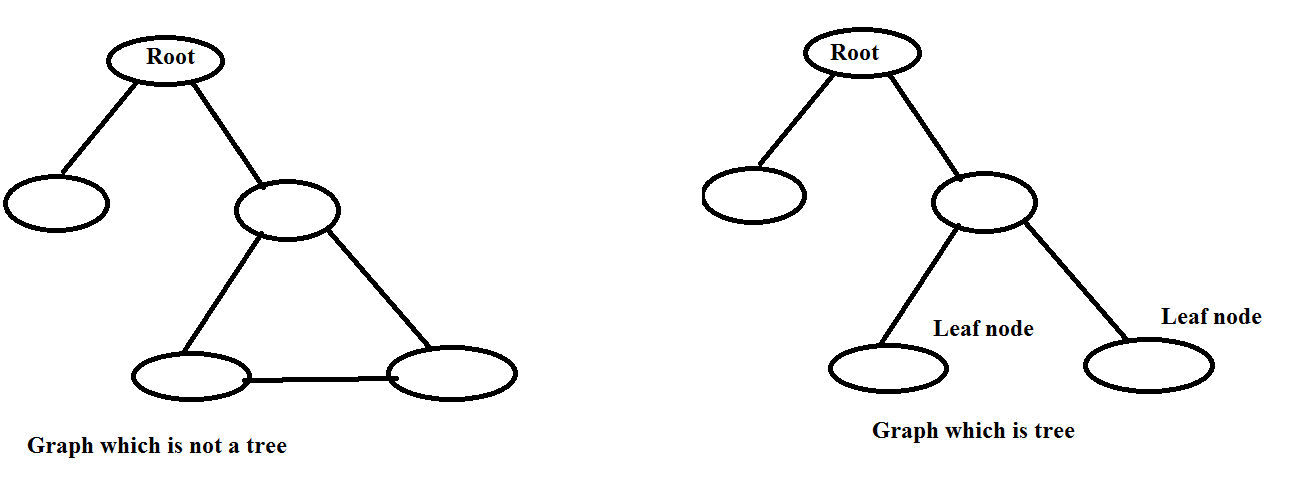
Ans-->Tree is a non-linear data structure which organize data in hierarchical structure and this is a recursive definition

OR

A tree is a connected graph without any circuits.

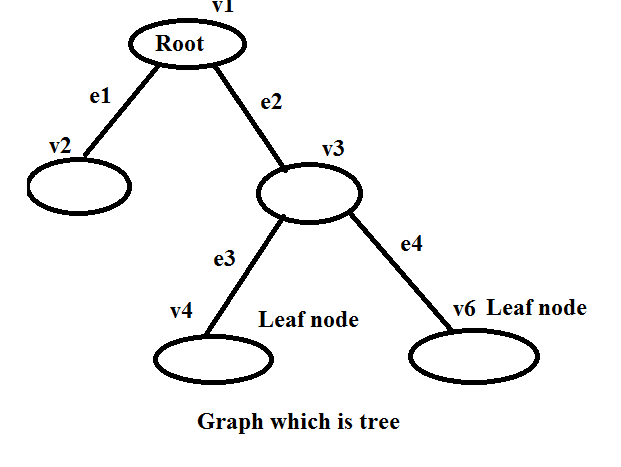
OR

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



Properties of trees:

1. A tree with n vertices has n-1 edges



2. A graph is a tree if and only if it is minimum connected

3. Any connected graph with n vertices and n-1 edges is a tree.

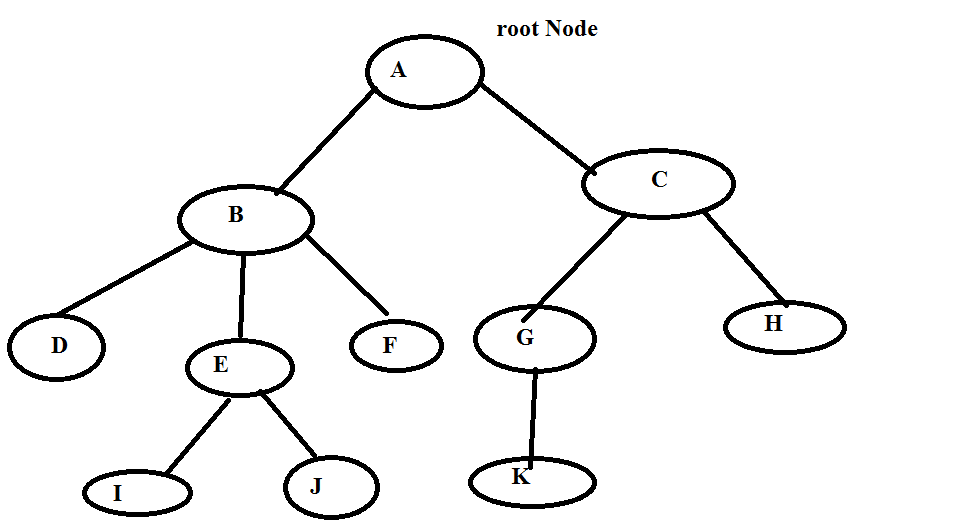
------------------------------------------------------------------------------------------------------------------------------------------

Q2. Explain Basic Terminology of tree

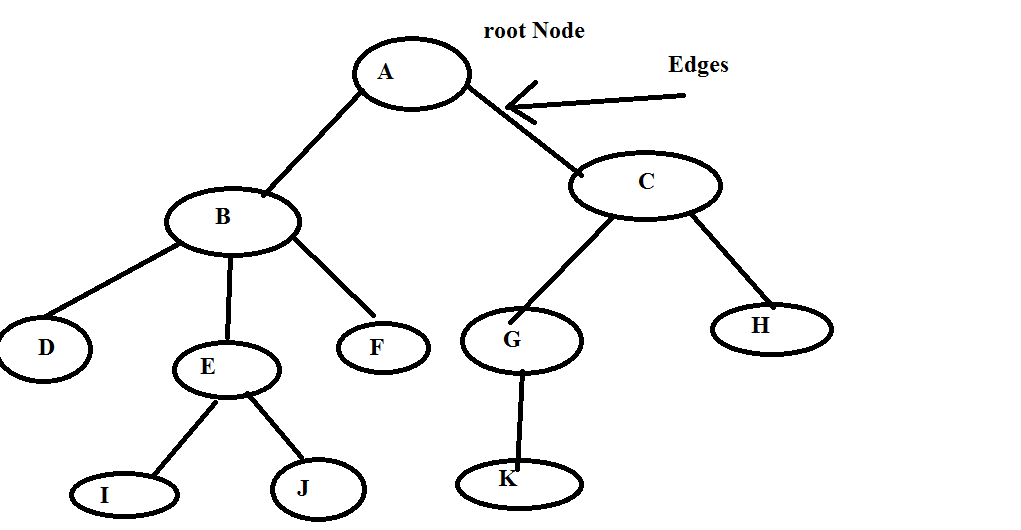
Ans-->

**1. ROOT:** Root node is the origin of tree data structure. it is the first node

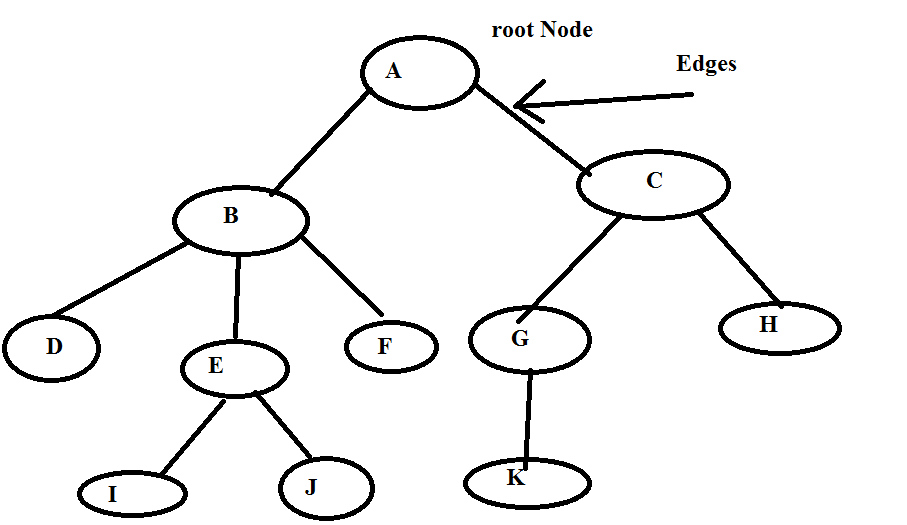
There must be only one root node and we can never have multiple root nodes in a tree.



2. Edge: The connecting link between only two nodes is called edges.

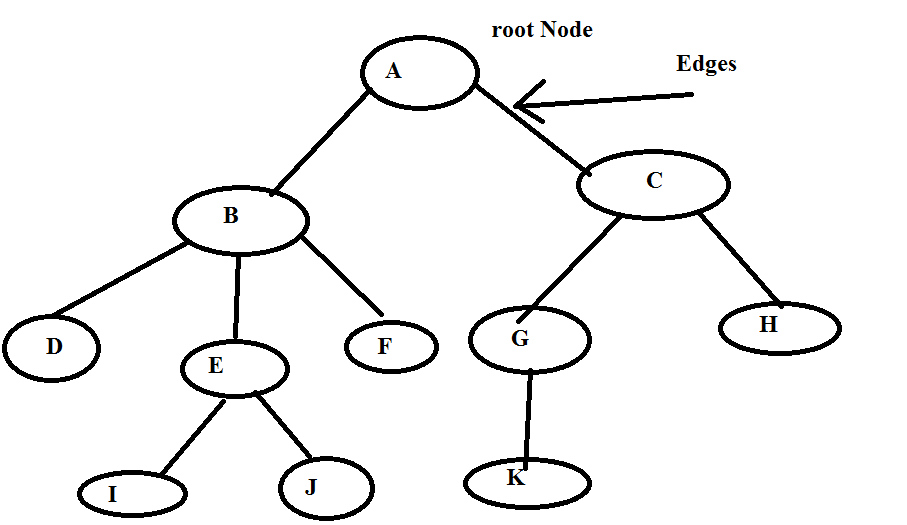


**3. Parent:**  The Node which has a branch form it to any other node is called as parent node.



Example: A,B,C,E and G are parent Nodes

4. Child: The node don't have any other node is called child node.



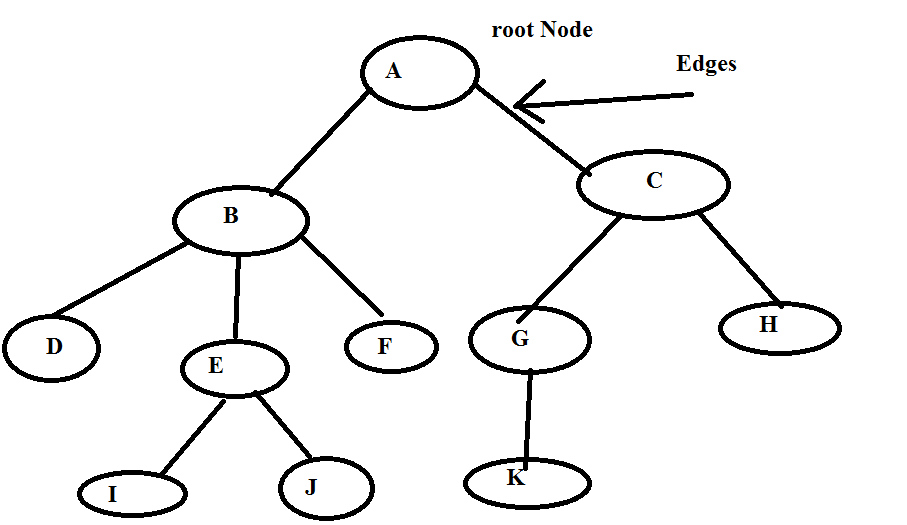
1. Here B and C are child node of A

2. G and H are child node of C

3. K is the only one child of G node

4. I and J is child node of E

**5. Siblings:** Nodes which belongs to the same parent are called as sibling. in other words node with the same parent is called sibling node

****

**1. B and C are sibling**

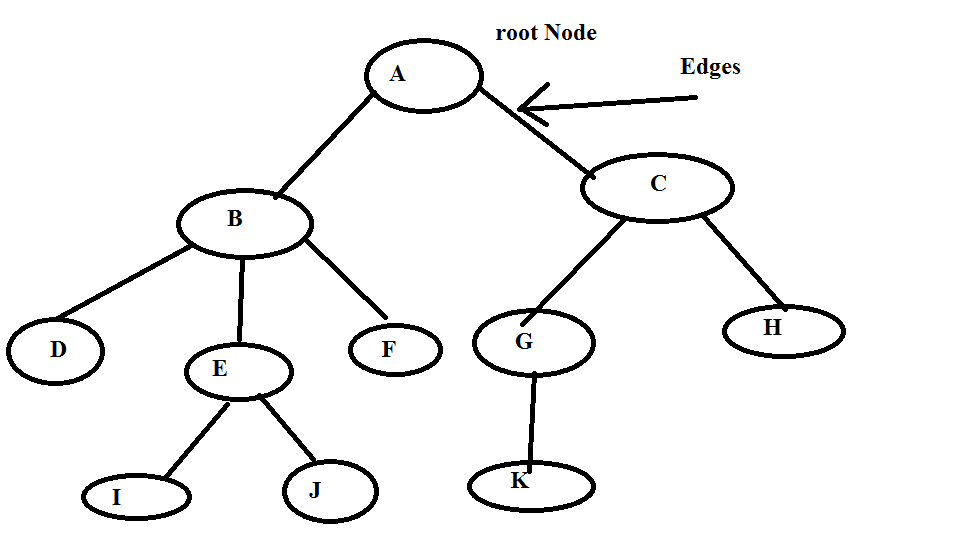
**2. D, E and F are sibling**

**3. G and H are sibling**

**4. I and J are sibling**

**6. Degree: The total number of child of a node is called as degree of that node**

**degree of a tree is the highest degree of a node among all the node in a tree.**

****

**here**

**1. Degree of B: 3**

**2. Degree of A: 2**

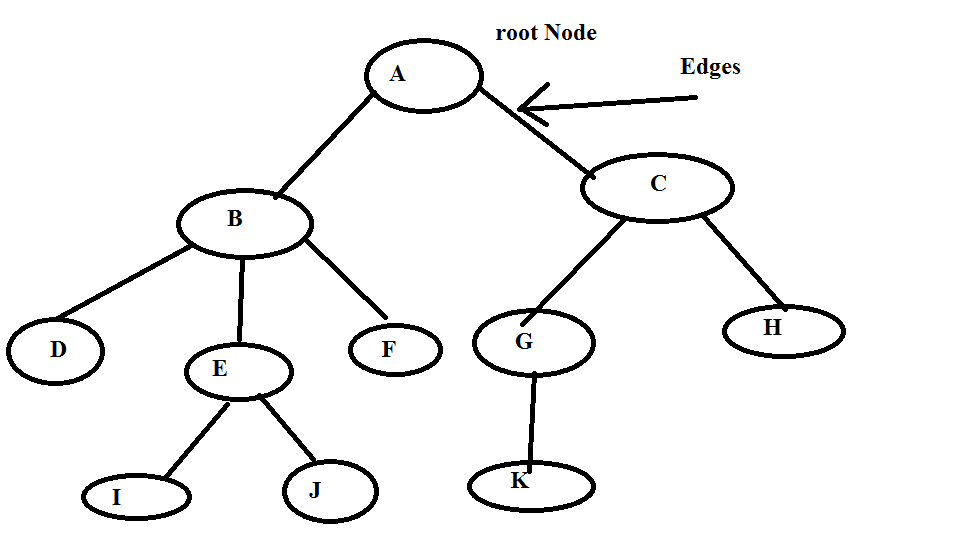
**3. Degree of F: 0**

**7. Internal Node:**  The node which has at least one child is called as internal node

They are also called non-terminal nodes.

Every non-leaf node is an internal node

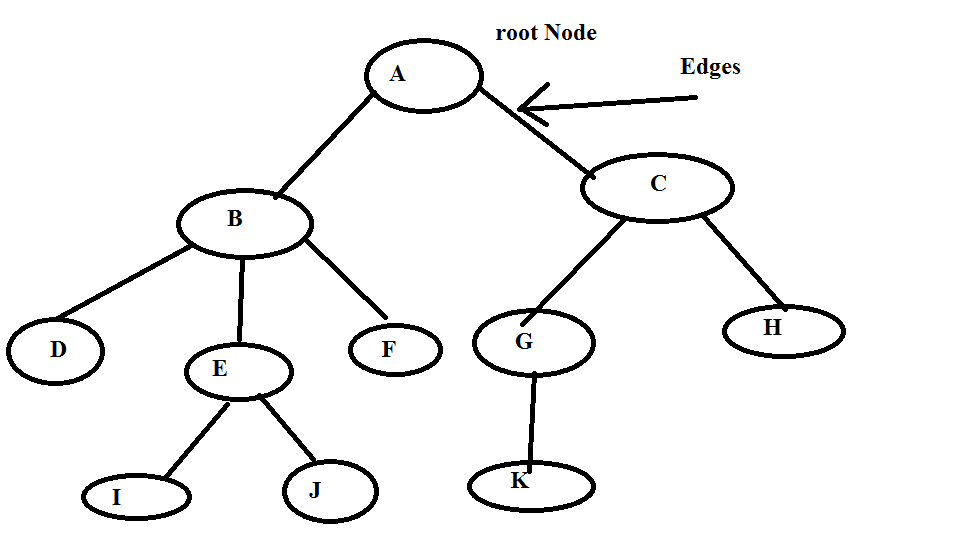
Example

****

**Example: A,B,C E and G**

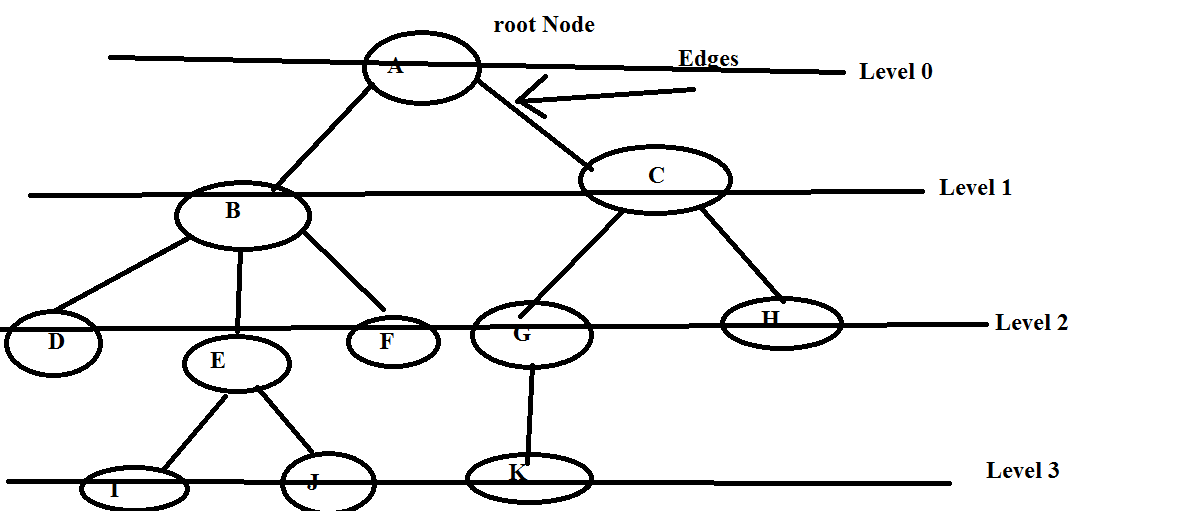
**8. Leaf Node: The node which does not have any child is called as leaf node**

**The leaf nodes are also called as External node/ terminal node**

****

**Example" D,I,J,F,K and H are leaf Nodes.**

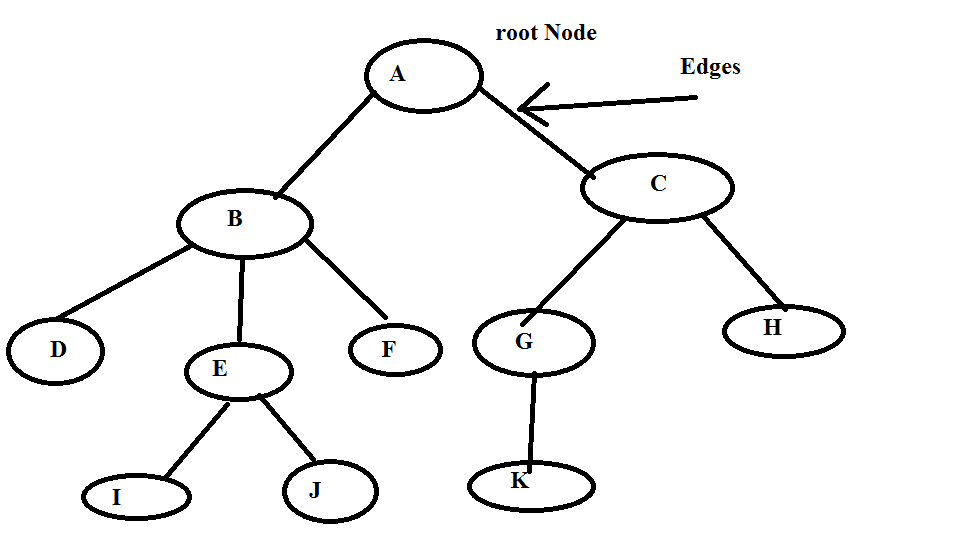
**9. Level: In a tree, each step from top to bottom is called level and level count starts with '0' and incremented by one at each level or step**

****

**10. Height :**  The total number of edges from leaf node to a particular node in the longest path is called height of that node

Example:

Height of the tree=Height of the root node



1. Height of K : 0

2. Height of B: 2

3. Height of A : 3

4. Height of G:1

5. Height of H: 0

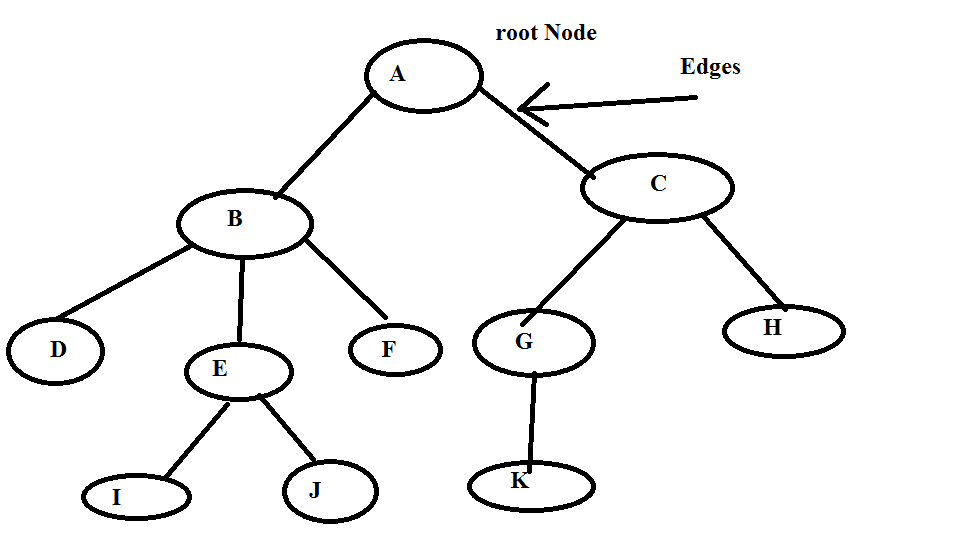
Height of tree is: 3(Height of root node)

Node: Height of all leaf node is 0.

**14. Depth:**

The total number of edges form root node to particular node is called as depth of that node

Depth of tree= Total Number of edges from root node to a leaf node in the longest path



Depth OF Node A: 0

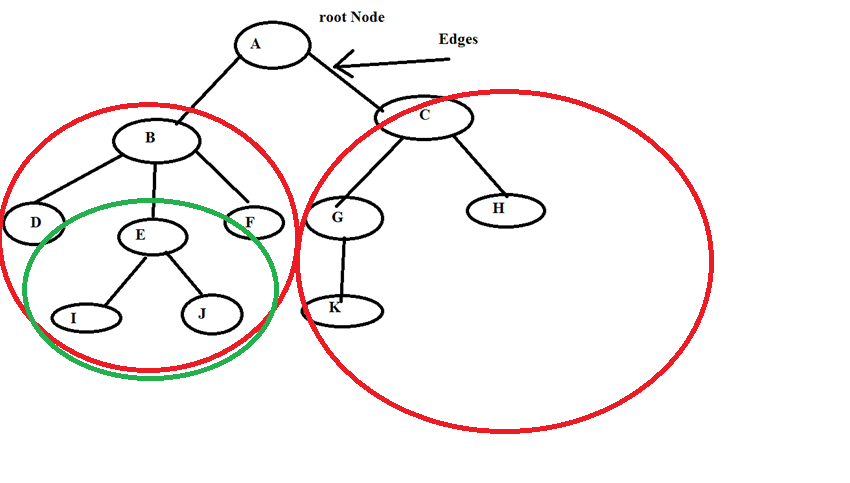
Depth OF B: 1

Depth OF D:2

Depth OF K: 3

Depth of tree: 3

**13: sub tree:** In a tree, each child from a node form a sub tree recursively. Every child node will form a sub tree on its parent node.



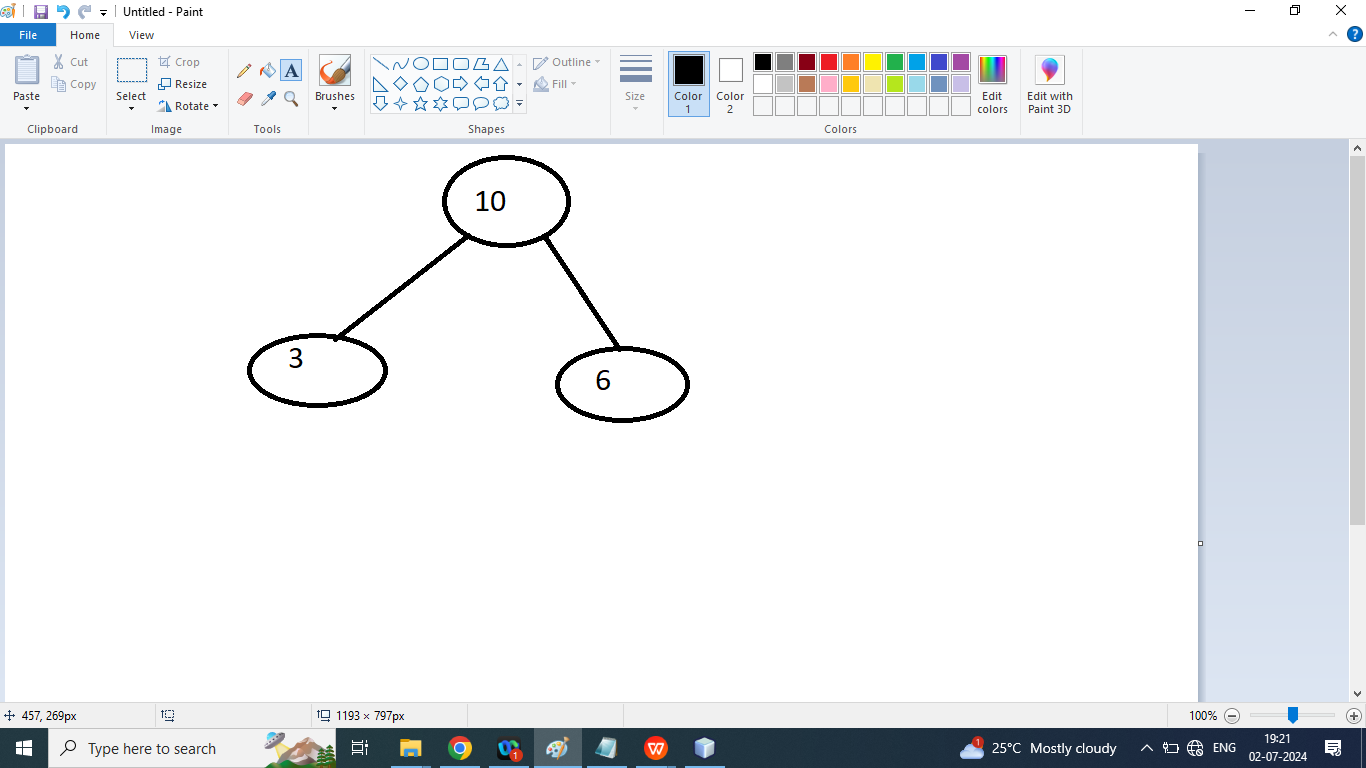
class Tree{

int data;

Node left;

Node right;

}



package dsausingjava;

public class Tree {

int data;

Tree left;

Tree right;

public Tree(int data){

this.data=data;

left=null;

right=null;

}

public static void main(String[] args) {

Tree t1=new Tree(10);

Tree t2=new Tree(2);

t1.left=t2;

Tree t3=new Tree(25);

t1.right=t3;

Tree t4=new Tree(1);

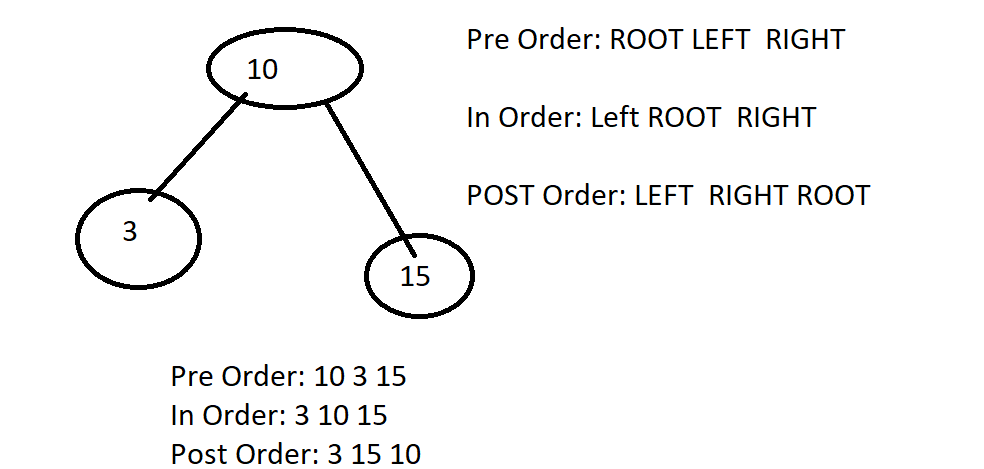
t1.left.left=t4;

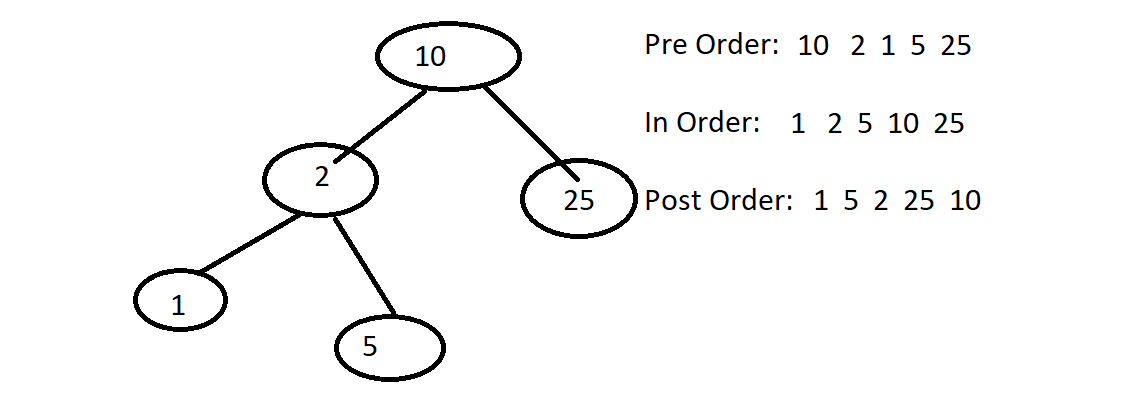
Tree t5=new Tree(5);

t1.left.right=t5;

}

}





Q2. Write a java program to print data of the tree using Pre order , Post Order ,In Order Tree Traversal Technique.

package dsausingjava;

public class Tree {

int data;

Tree left;

Tree right;

public Tree(int data) {

this.data = data;

left = null;

right = null;

}

public static void preOrder(Tree root) {

//Root Left Right

if (root == null) {

return;

}

System.out.print("\t" + root.data);

preOrder(root.left);

preOrder(root.right);

}

public static void InOrder(Tree root) {

//Root Left Right

if (root == null) {

return;

}

InOrder(root.left);

System.out.print("\t" + root.data);

InOrder(root.right);

}

public static void postOrder(Tree root) {

//Root Left Right

if (root == null) {

return;

}

postOrder(root.left);

postOrder(root.right);

System.out.print("\t" + root.data);

}

public static void main(String[] args) {

Tree t1 = new Tree(10);

Tree t2 = new Tree(2);

t1.left = t2;

Tree t3 = new Tree(25);

t1.right = t3;

Tree t4 = new Tree(1);

t1.left.left = t4;

Tree t5 = new Tree(5);

t1.left.right = t5;

System.out.println("Print Data using Pre order");

preOrder(t1);

System.out.println("\nPrint Data using In order");

InOrder(t1);

System.out.println("\nPrint Data using Post order");

postOrder(t1);

}

}

Binary Search Example:

package dsausingjava;

public class Tree {

int data;

Tree left;

Tree right;

public Tree(int data) {

this.data = data;

left = null;

right = null;

}

public static void preOrder(Tree root) {

//Root Left Right

if (root == null) {

return;

}

System.out.print("\t" + root.data);

preOrder(root.left);

preOrder(root.right);

}

public static void InOrder(Tree root) {

//Root Left Right

if (root == null) {

return;

}

InOrder(root.left);

System.out.print("\t" + root.data);

InOrder(root.right);

}

public static void postOrder(Tree root) {

//Root Left Right

if (root == null) {

return;

}

postOrder(root.left);

postOrder(root.right);

System.out.print("\t" + root.data);

}

public static boolean bst(Tree root, int element) {

boolean b = false;

if (root.data == element) {

b = true;

}

else if(element<root.data){

b=bst(root.left,element);

}

else if(element>root.data){

b=bst(root.right,element);

}

return b;

}

public static void main(String[] args) {

Tree t1 = new Tree(10);

Tree t2 = new Tree(2);

t1.left = t2;

Tree t3 = new Tree(25);

t1.right = t3;

Tree t4 = new Tree(1);

t1.left.left = t4;

Tree t5 = new Tree(5);

t1.left.right = t5;

System.out.println("Print Data using Pre order");

preOrder(t1);

System.out.println("\nPrint Data using In order");

InOrder(t1);

System.out.println("\nPrint Data using Post order");

postOrder(t1);

System.out.println("\nExample Binary Search Tree :\n");

boolean b = bst(t1, 10);

System.out.println("Search 10 : " + b);

b = bst(t1, 2);

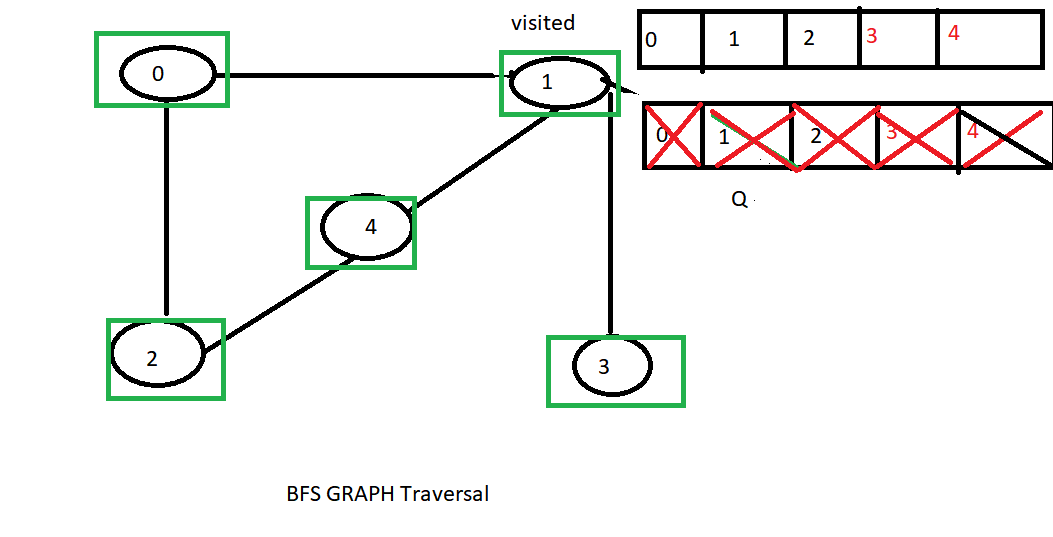
System.out.println("Search 2 : " + b);

b = bst(t1, 25);

System.out.println("Search 25 : " + b);

}

}



Graph Traversal Technique

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package dsa\_java\_doubt\_session;

import java.util.LinkedList;

import java.util.Queue;

// Class to represent a graph using adjacency list

class Graph {

int vertices;

LinkedList<Integer>[] adjList;

Graph(int vertices) {

this.vertices = vertices;

adjList = new LinkedList[vertices];

for (int i = 0; i < vertices; ++i) {

adjList[i] = new LinkedList<>();

}

}

// Function to add an edge to the graph

void addEdge(int u, int v) {

adjList[u].add(v);

}

// Function to perform Breadth First Search on a graph

// represented using adjacency list

void bfs(int startNode) {

// Create a queue for BFS

Queue<Integer> queue = new LinkedList<>();

boolean[] visited = new boolean[vertices];

// Mark the current node as visited and enqueue it

visited[startNode] = true;

queue.add(startNode);

// Iterate over the queue

while (!queue.isEmpty()) {

// Dequeue a vertex from queue and print it

int currentNode = queue.poll();

System.out.print(currentNode + " ");

// Get all adjacent vertices of the dequeued

// vertex currentNode If an adjacent has not

// been visited, then mark it visited and

// enqueue it

for (int neighbor : adjList[currentNode]) {

if (!visited[neighbor]) {

visited[neighbor] = true;

queue.add(neighbor);

}

}

}

}

}

class Main {

public static void main(String[] args) {

// Number of vertices in the graph

int vertices = 5;

// Create a graph

Graph graph = new Graph(vertices);

// Add edges to the graph

graph.addEdge(0, 1);

graph.addEdge(0, 2);

graph.addEdge(1, 3);

graph.addEdge(1, 4);

graph.addEdge(2, 4);

// Perform BFS traversal starting from vertex 0

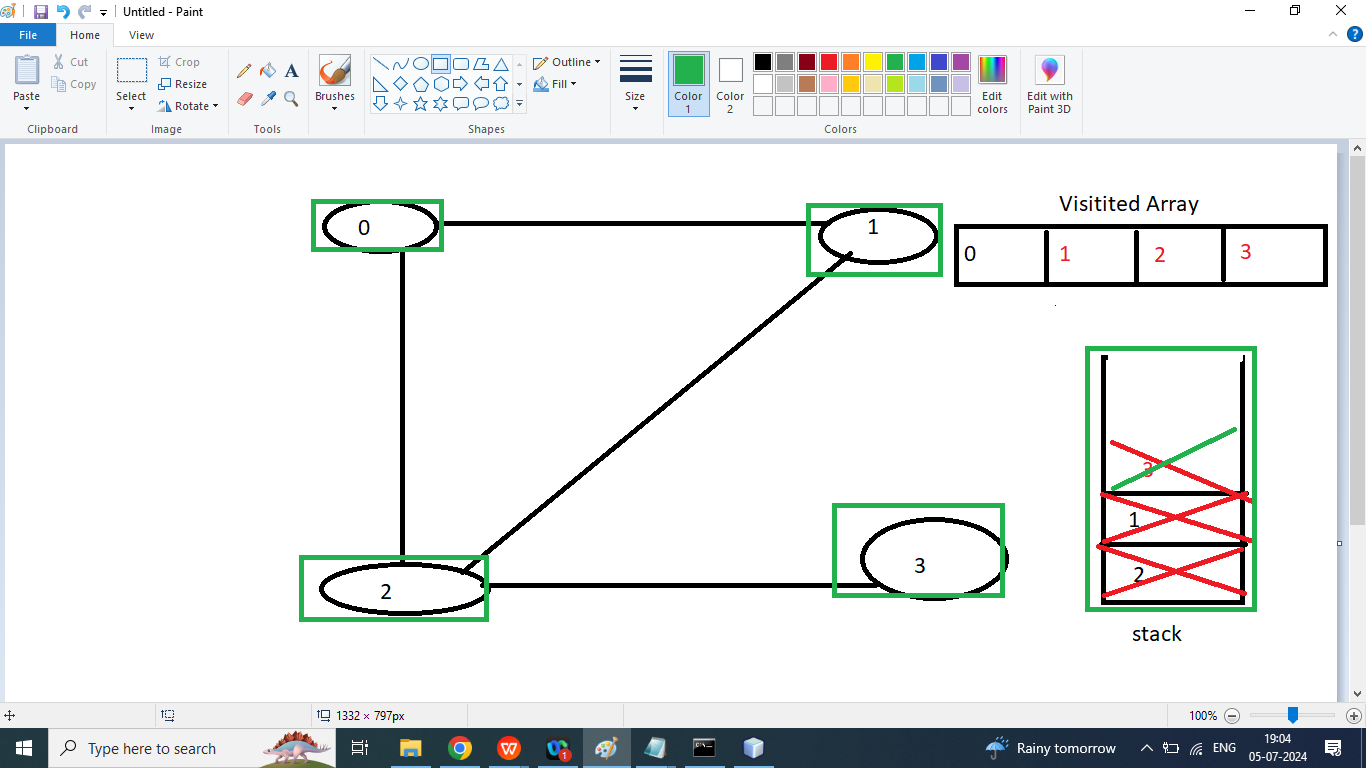
System.out.print("Breadth First Traversal starting from vertex 0: ");

graph.bfs(0);

}

}

DFS GRAPH TRAVERSAL



/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package dsa\_java\_doubt\_session;

// Java program to print DFS traversal

// from a given graph

import java.io.\*;

import java.util.\*;

// This class represents a

// directed graph using adjacency

// list representation

class Graph1 {

private int V;

// Array of lists for

// Adjacency List Representation

private LinkedList<Integer> adj[];

// Constructor

@SuppressWarnings("unchecked")

Graph1(int v) {

V = v;

adj = new LinkedList[v];

for (int i = 0; i < v; ++i) {

adj[i] = new LinkedList();

}

}

// Function to add an edge into the graph

void addEdge(int v, int w) {

// Add w to v's list.

adj[v].add(w);

}

// A function used by DFS

void DFSUtil(int v, boolean visited[]) {

// Mark the current node as visited and print it

visited[v] = true;

System.out.print(v + "\*\*\* ");

// Recur for all the vertices adjacent to this

// vertex

Iterator<Integer> i = adj[v].listIterator();

while (i.hasNext()) {

int n = i.next();

if (!visited[n]) {

DFSUtil(n, visited);

}

}

}

// The function to do DFS traversal.

// It uses recursive DFSUtil()

void DFS(int v) {

// Mark all the vertices as

// not visited(set as

// false by default in java)

boolean visited[] = new boolean[V];

// Call the recursive helper

// function to print DFS

// traversal

DFSUtil(v, visited);

}

// Driver Code

public static void main(String args[]) {

Graph1 g = new Graph1(4);

g.addEdge(0, 1);

g.addEdge(0, 2);

g.addEdge(1, 2);

g.addEdge(2, 0);

g.addEdge(2, 3);

System.out.println(

"Following is Depth First Traversal "

+ "(starting from vertex 2)");

// Function call

g.DFS(0);

}

}