II - I

DATA STRUCTURES

Non Linear Data Structures

Module 4 QB Solutions

Ujjwal • Vishal • Ruthvik

PART-A

Q1)Let G be a graph with n vertices and m edges. Find the tightest upper bound on the runningtimeondepthfirstsearchofgraphG. Assume that the graph is represented using an adjacency matrix.

A. Depth First Search of a graph takes O(m+n) time when the graph is represented using an adjacency list. In adjacency matrix representation, the graph is represented as an ‘n x n’ matrix. To do DFS, for every vertex, we traverse the row corresponding to that vertex to find all adjacent vertices In adjacency list representation we traverse only the adjacent vertices of the vertex). Therefore time complexity becomes O(n2).

Q2) Let Gbeanundirectedgraphwithnverticesand 25 Edges Such that each vertex has degree at least 3. Find the maximum possible value of n?

A. As per handshaking lemma, Sum of degree of all vertices < 2 * no. of edges. Let v be the number of vertices in the graph. 3 * v 2 25 Maximum value of v 16

Q3) In Binary Tree, forever node the differencebetween thenumberofnodesinthe leftandrightsubtreesisat mosttwo. Iftheheightof thetreeishisgreaterthan 0,then find the minimum number of nodes in the tree?

A. 2h-1 1 Let there be n(h) nodes at height h. In a perfect tree where every node has exactly two children, except leaves, following recurrence holds.

n(h) 2n(h-1) 1

In given case, the numbers of nodes are two less, therefore

n(h) 2n(h-1) 1 -2

2n(h-1) -1

The above equation is only satisfied by 2h-1 +1

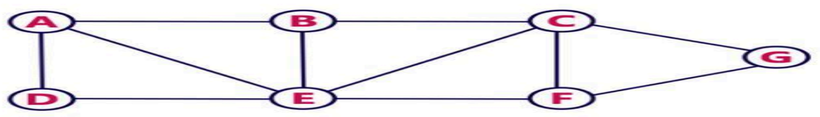
Q4) Write a function to find the number of occurrence sofa number in a tree of numbers?

A. A tree of numbers is stored as an array in any programming language. Number of occurrences of a number can be counted using a simple count increment program.

Q5) Write breadth first search (BFS)traversal algorithm, based on a queue,to traverse a directed graph of n vertices and m edges?

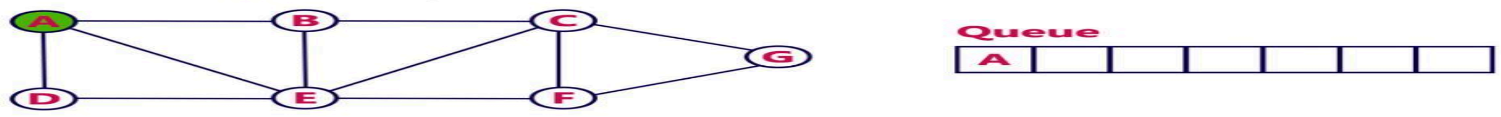
A.

Consider the following example graph to perform BFS traversal



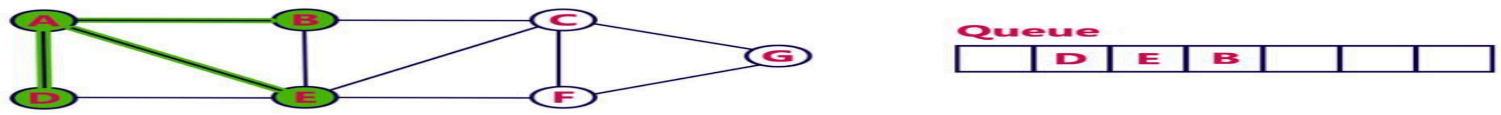
Step 1:

- Select the vertex **A** as starting point (visit **A**).
- Insert **A** into the Queue.



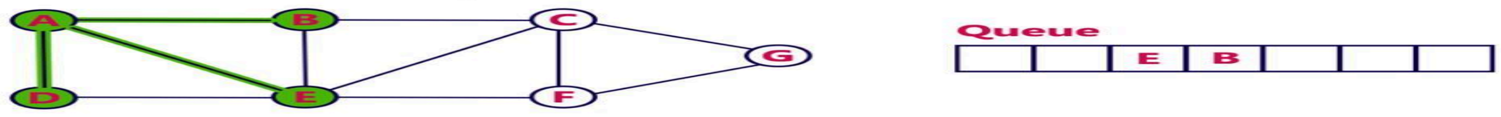
Step 2:

- Visit all adjacent vertices of **A** which are not visited (**D**, **E**, **B**).
- Insert newly visited vertices into the Queue and delete A from the Queue..



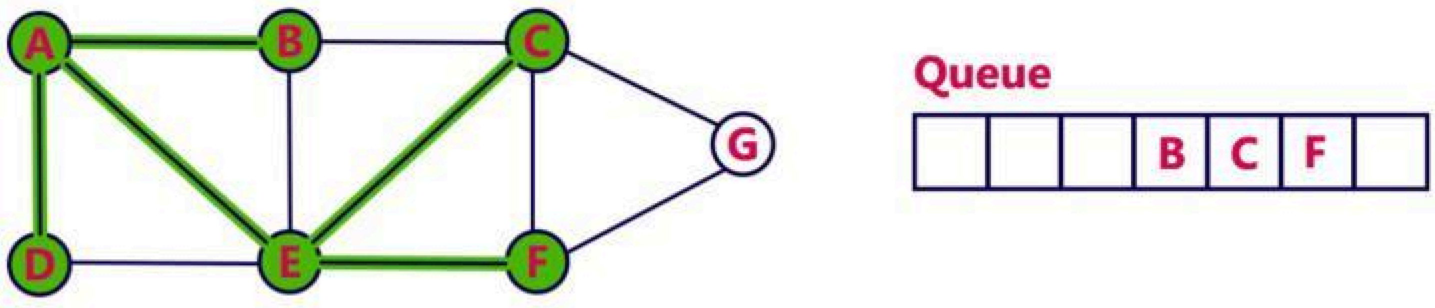
Step 3:

- Visit all adjacent vertices of **D** which are not visited (there is no vertex).
- Delete D from the Queue.



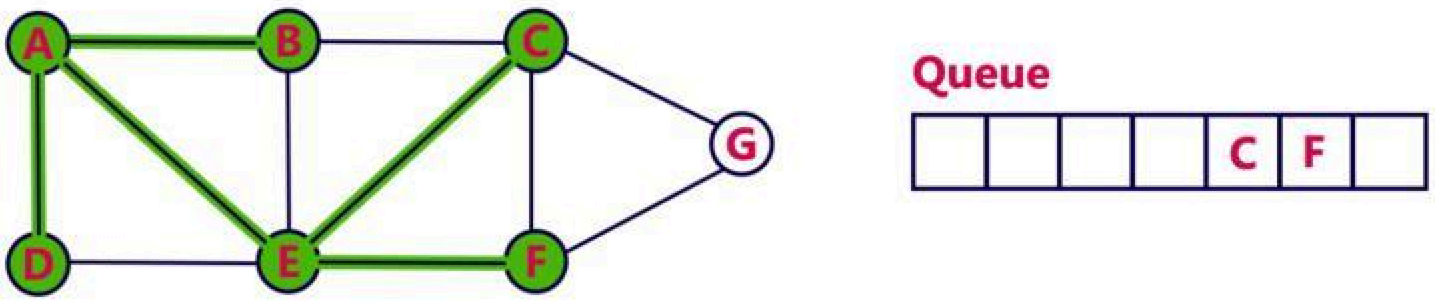
Step 4:

- Visit all adjacent vertices of **E** which are not visited (**C**, **F**).
- Insert newly visited vertices into the Queue and delete E from the Queue.



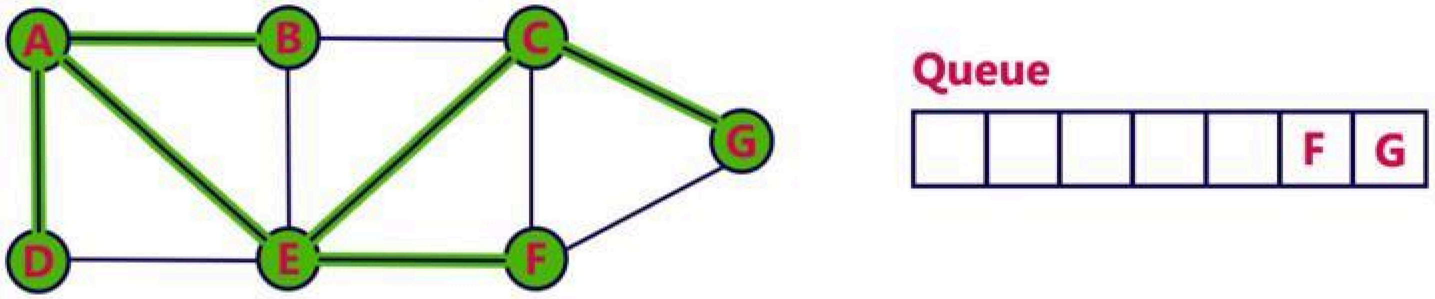
Step 5:

- Visit all adjacent vertices of **B** which are not visited (**there is no vertex**).
- Delete **B** from the Queue.



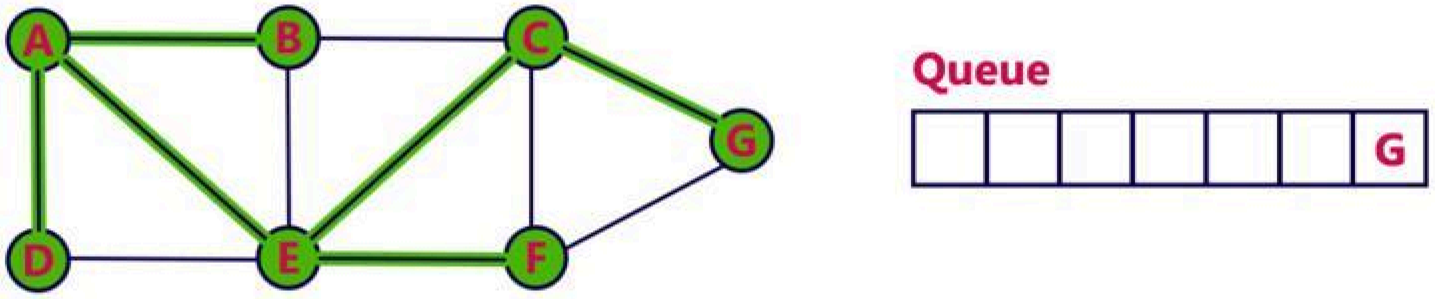
Step 6:

- Visit all adjacent vertices of **C** which are not visited (**G**).
- Insert newly visited vertex into the Queue and delete **C** from the Queue.



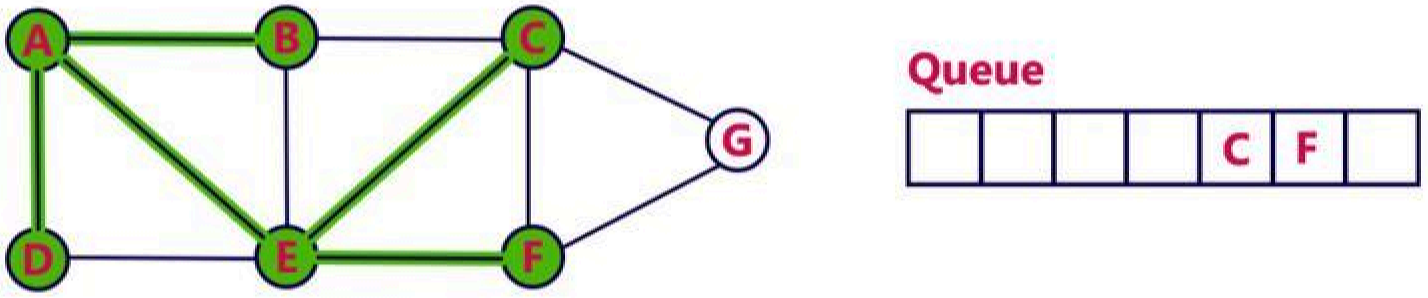
Step 7:

- Visit all adjacent vertices of **F** which are not visited (**there is no vertex**).
- Delete **F** from the Queue.



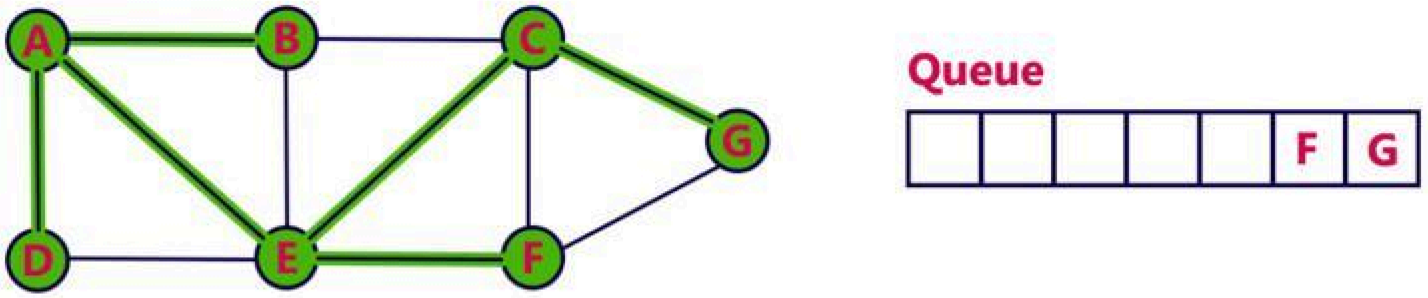
Step 5:

- Visit all adjacent vertices of **B** which are not visited (**there is no vertex**).
- Delete **B** from the Queue.



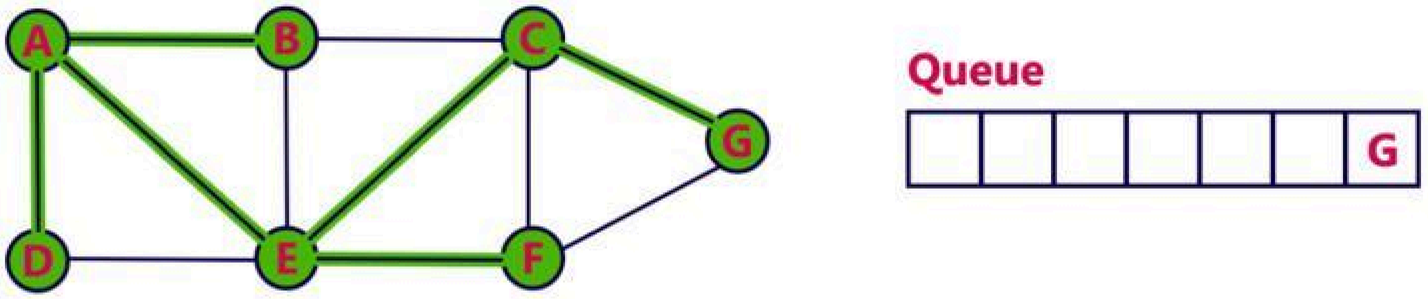
Step 6:

- Visit all adjacent vertices of **C** which are not visited (**G**).
- Insert newly visited vertex into the Queue and delete **C** from the Queue.



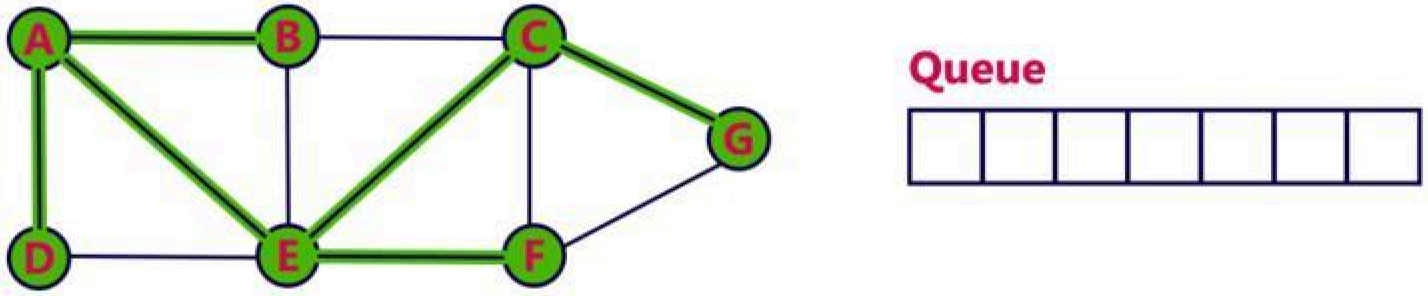
Step 7:

- Visit all adjacent vertices of **F** which are not visited (**there is no vertex**).
- Delete **F** from the Queue.

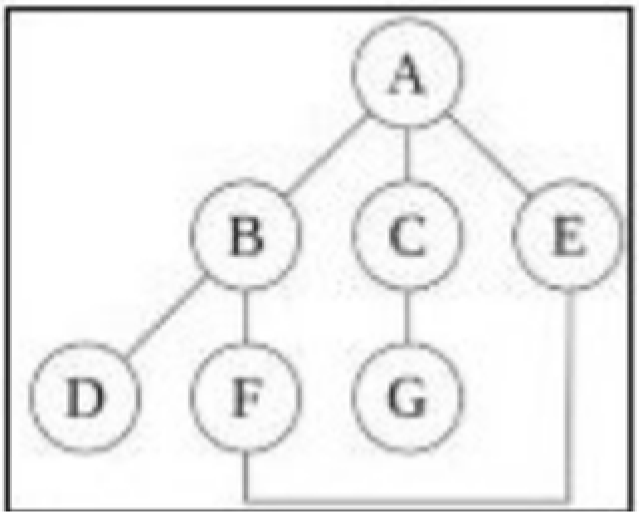


Step 8:

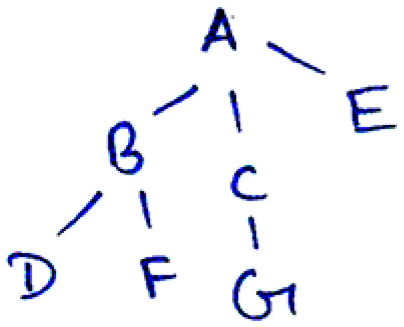
- Visit all adjacent vertices of **G** which are not visited (**there is no vertex**).
- Delete **G** from the Queue.



Q6) Consider the example. Find BFS and DFS.

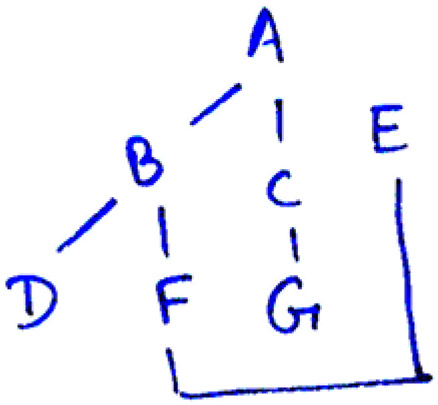


A.



BFS

A BCEDFG

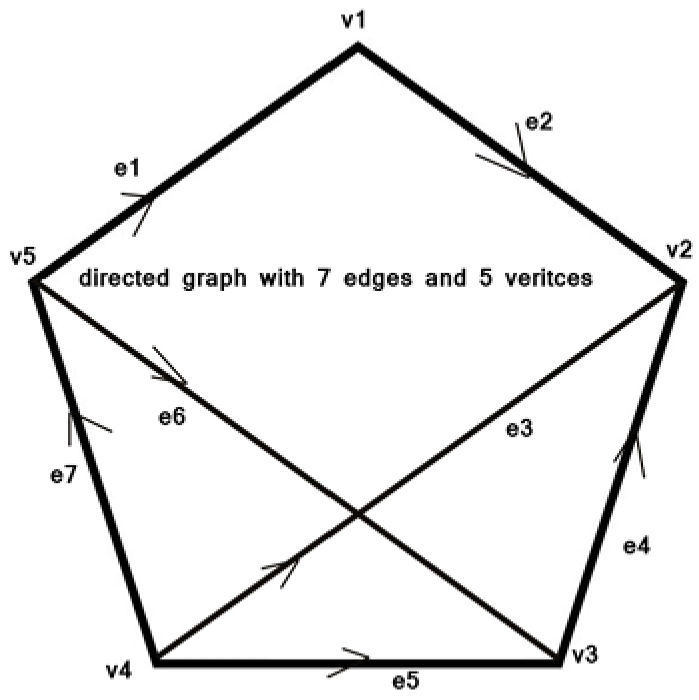


DFS

ABDFECG

Q7) Draw a directed graph with five vertices and seven edges. Exactly one of the edges should be a loop, and do not have any multiple edges.

A.



Q8) Given A Binary Tree. Write an efficient algorithm to delete the entire binary tree.

A. To delete a tree, we must traverse all the nodes of the tree and delete them one by one. We should use the postorder transversal because before deleting the parent node, we should delete its child nodes first.

Algorithm Postorder Tree Deletion:

1. Traverse the left subtree, i.e., call Postorder(left-subtree)
2. Traverse the right subtree, i.e., call Postorder(right-subtree)
3. Visit the root and delete the node

Q9) Given A Binary Tree. Write an efficient algorithm to print a left view of a binary tree.

A. This is solved by recursively traversing each level in the tree(visiting the left node before the right node). Whenever we move to the next level, print the leftmost node in that level Note: we traverse the left subtree before the right subtree). The variable maxlevel is used to track the maximum level in the tree that is traversed so far. Each time a node with a level greater than max level is encountered, its value is printed, and the value of max level is updated to that value.

Algorithm: Left View of a Binary Tree

1. **Initialize**:
    - Create an empty queue for level order traversal.
    - Enqueue the root node along with its level (0).
2. **Level Order Traversal**:
    - Initialize a variable to track the maximum level visited so far.
3. **Traversal Loop**:
    - While the queue is not empty:
        1. Dequeue the front node and its level from the queue.
        2. If the current level is greater than the maximum level visited so far:
            - Print the node (this is the first node at this level).
            - Update the maximum level visited.
        3. Enqueue the left and right children of the node along with their levels (current level + 1).

Q10) Given a binary tree, write a recursive solution to traverse the tree using post order traversal.

A. **Algorithm: Post-Order Traversal of a Binary Tree**

Post-order traversal involves visiting the left subtree, then the right subtree, and finally the root node. Here's a step-by-step recursive algorithm for postorder traversal:

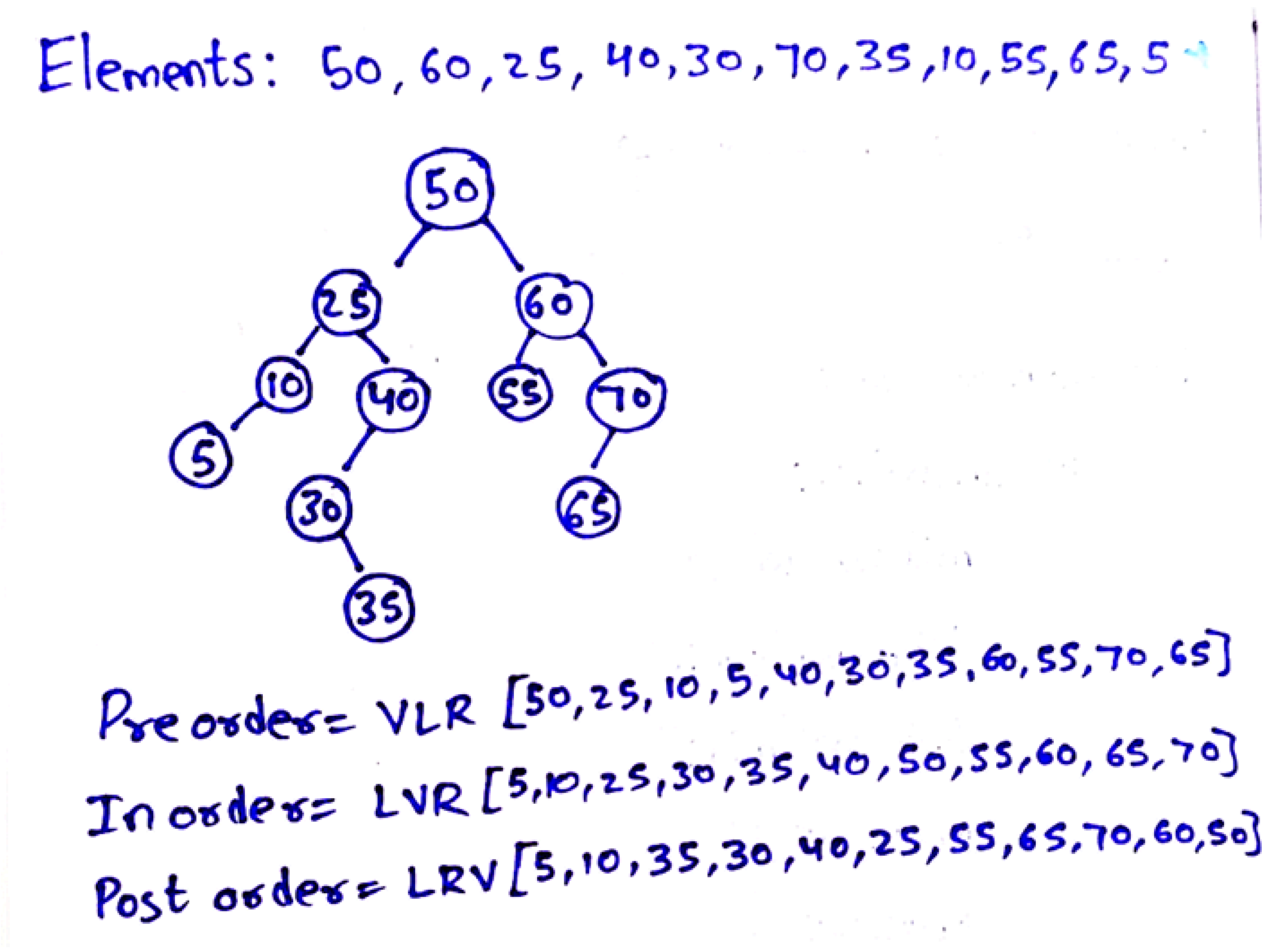
Steps:

1. Base Case: If the current node is `null`, return.
2. **Recursive Case**:
   - ○ Recursively traverse the left subtree.
   - ○ Recursively traverse the right subtree.
   - ○ Visit the root node (process the current node).

PART-B

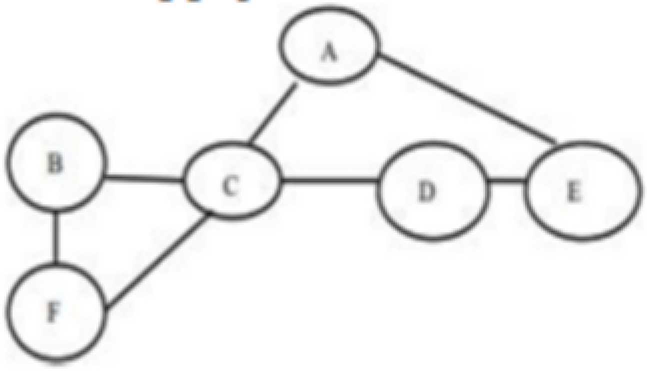
Q1) ConstructaBinarySearch Treeforthefollowingdata and doin-order,Preorder andPost-ordertraversalof thetree.
50,60,25,40,30, 70,35,10,55,65,5

A. Lec-56: Preorder, Inorder and Postorder in 5 minute | Tree Tr…


Elements: 50, 60, 25, 40, 30, 70, 35, 10, 55, 65, 5

Preorder= VLR [50, 25, 10, 5, 40, 30, 35, 60, 55, 70, 65]
Inorder= LVR [5, 10, 25, 30, 35, 40, 50, 55, 60, 65, 70]
Post order= LRV [5, 10, 35, 30, 40, 25, 55, 65, 70, 60, 50]

Q2) Explain the breadth first search and depth first search tree traversal on the following graph.



A.

Graph Traversals: Depth First Search Traversal

- We follow a path through the graph until we reach a dead end.
- We then back up until we reach a node with an edge to an unvisited node.
- We take this edge and again follow it until we reach a dead end.
- This process continues until we back up to the starting node and it has no edges to unvisited nodes.

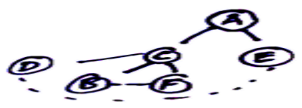
Breadth First Search Traversal

- From the starting node, we follow all paths of length one.
- Then we follow paths of length two that go to unvisited nodes.
- We continue increasing the length of the paths until there are no unvisited nodes along any of the paths.

- B.F.S : A, C, E, B, F, D

Q:

~~A~~	~~C~~	~~E~~	~~B~~	~~F~~	~~D~~



D.F.S : A, C, B, F, D, E

REFERENCE

5.1 Graph Traversals - BFS & DFS -Breadth First Search and D…

Q3) Illustrate The Output obtained after pre-order, in-orderandpost-order traversal of the following tree

A.

```
     1
    / \
   2   3
  / \   \
 4   5   6
```

Pre-Order Traversal (Root, Left, Right):

1. Visit the root.
2. Traverse the left subtree.
3. Traverse the right subtree.

For the given tree, the pre-order traversal is:

1,2,4,5,3,61, 2, 4, 5, 3, 6

In-Order Traversal (Left, Root, Right):

1. Traverse the left subtree.
2. Visit the root.
3. Traverse the right subtree.

For the given tree, the in-order traversal is:

4,2,5,1,3,6

Post-Order Traversal (Left, Right, Root):

1. Traverse the left subtree.

2. Traverse the right subtree.
3. Visit the root.

For the given tree, the post-order traversal is:

4,5,2,6,3,14, 5, 2, 6, 3, 1

Summary of Traversals:

- **Pre-Order**: 1,2,4,5,3,61, 2, 4, 5, 3, 6
- **In-Order**: 4,2,5,1,3,64, 2, 5, 1, 3, 6
- **Post-Order**: 4,5,2,6,3,1

Q4) Develop an algorithm to implement Depth First Search traversal of a graph using AdjacencyMatrix.

A. **Algorithm: Depth-First Search (DFS) Using Adjacency Matrix**

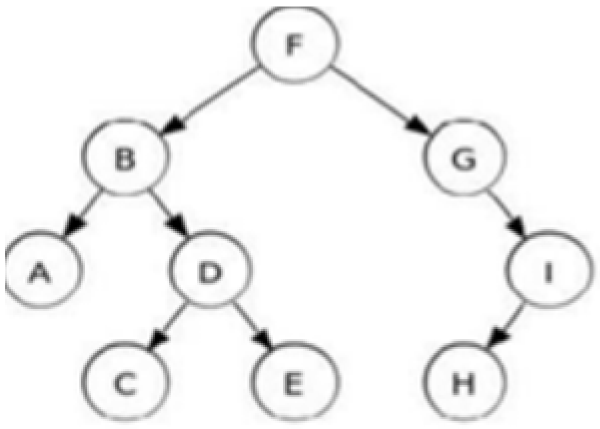
Steps:

1. **Initialize Data Structures**:
   - Create an array `visited` to keep track of visited nodes, initialized to `False` for all nodes.
   - Use a stack to manage the traversal order.
2. **Define the DFS Function**:
   - Use a recursive function (or iterative with stack) to visit nodes.
3. **Implementation**:
   - Start from a given node (source node).
   - Mark the node as visited.
   - Recursively visit all adjacent nodes that are not yet visited.

Q5) Construct a binary search tree by inserting the following nodes insequence:68,85,23,38,44,80,30,108,26,5,92,60.Write in-order, pre-order and post-order traversal of the above generated Binary search tree.

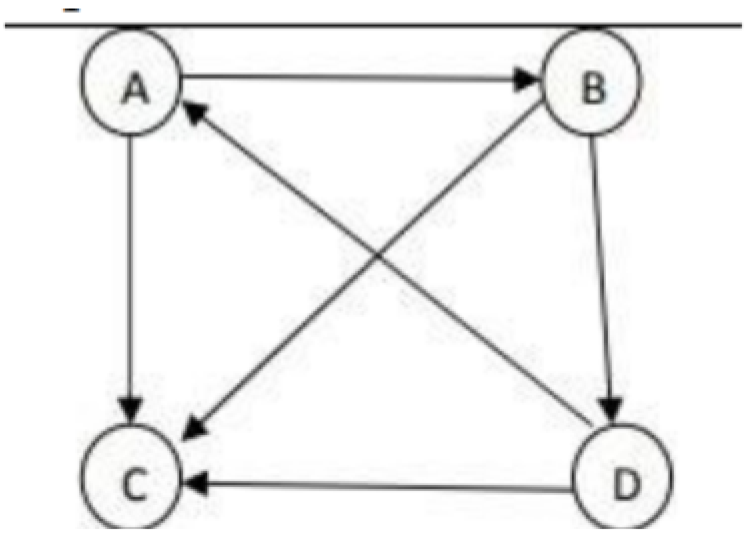
A. Follow same procedure as 1st Question

Q6) Write the in-order, pre-order and post-order traversals for the given binary tree.



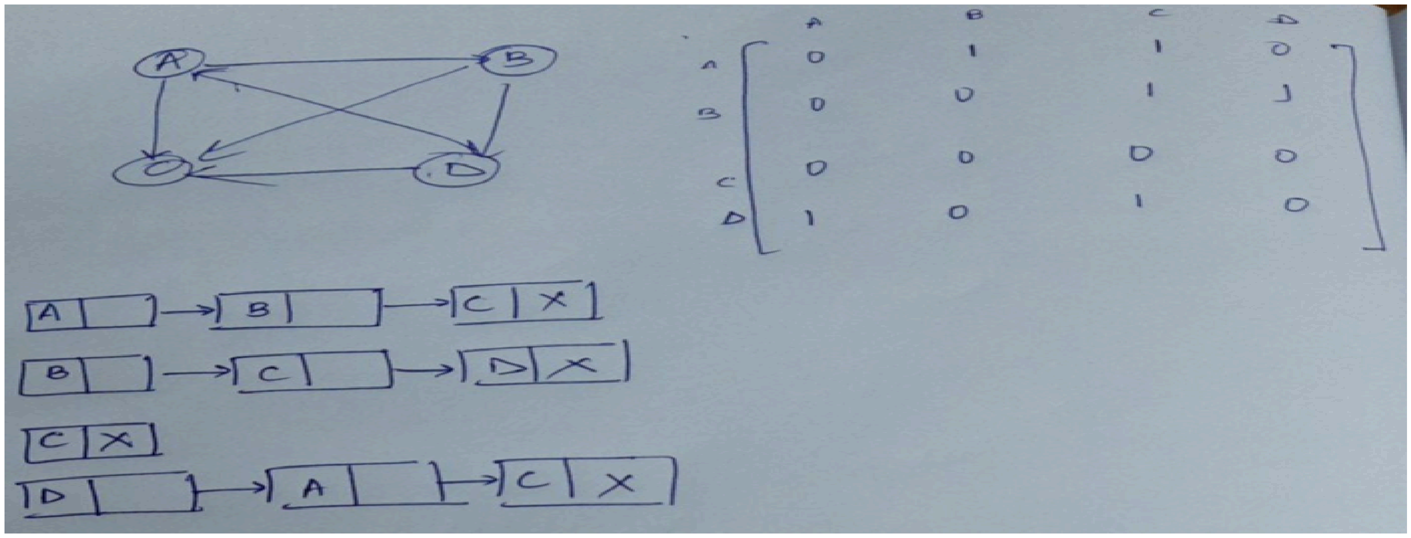
Follow same procedure as 1st question

Q7) Define Adjacency Matrix? Draw the Adjacency Matrix of the following graph. Also give adjacency list representation for the same.



A. Adjacency Matrix:

● Adjacency Matrix is a 2D array of size V x V where V is the number of vertices in a graph.
● Let the 2D array be adj[ ][ ], a slot adj[i][j] = 1 indicates that there is an edge from vertex i to vertex j.
● Adjacency matrix for an undirected graph is always symmetric. Adjacency Matrix is also used to represent weighted graphs. If adj[i][j] = w, then there is an edge from vertex i to vertex j with weight w.

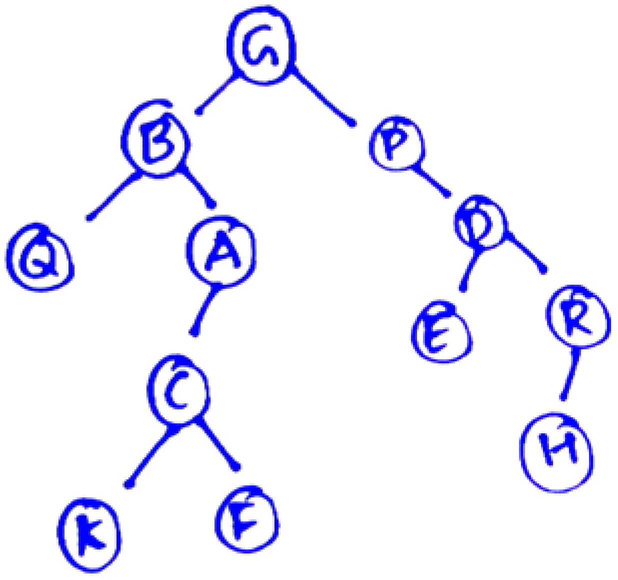


Q8) Explain the array and linked representation of a binary tree using a suitable example?
A. LINK:[Data Structures Tutorials - Binary Tree Representations with an example]()

Q9) Define a binary tree? Construct a binary tree given the pre-order traversal and in-order traversals as follows:
Pre-Order Traversal: G B Q A C K F P D E R H
In-Order Traversal: Q B K C F A G P E D H R

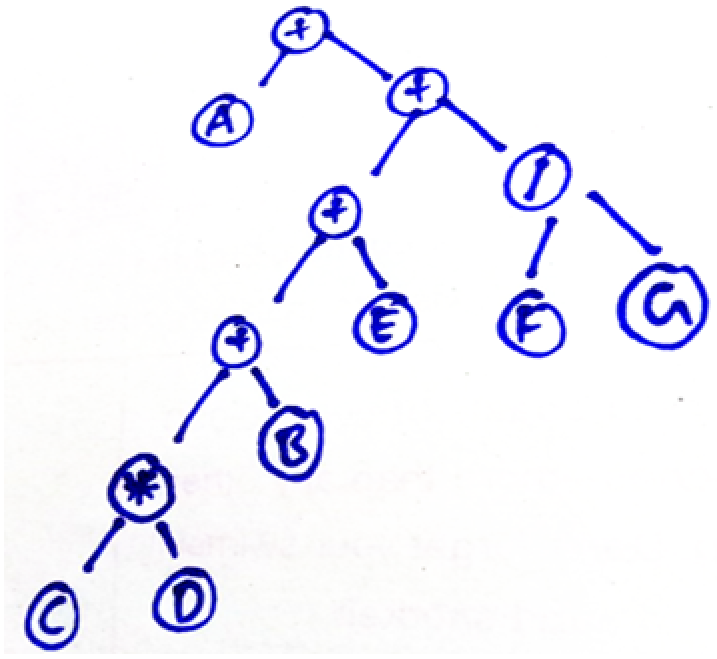
A. A binary tree is a tree data structure in which each node has at most two children, which are referred to as the left child and the right child.

5.7 Construct Binary Tree from Preorder and Inorder Traversal |…



Q10) Construct an expression tree for the following expression. A + (B +C*D+E)+F/G.Make Preorder Traversal of the resultant tree.

A.



PRE ORDER A *CDBE/FG (DOUBT)

Q11) Explain the binary tree traversal algorithms with a suitable example?

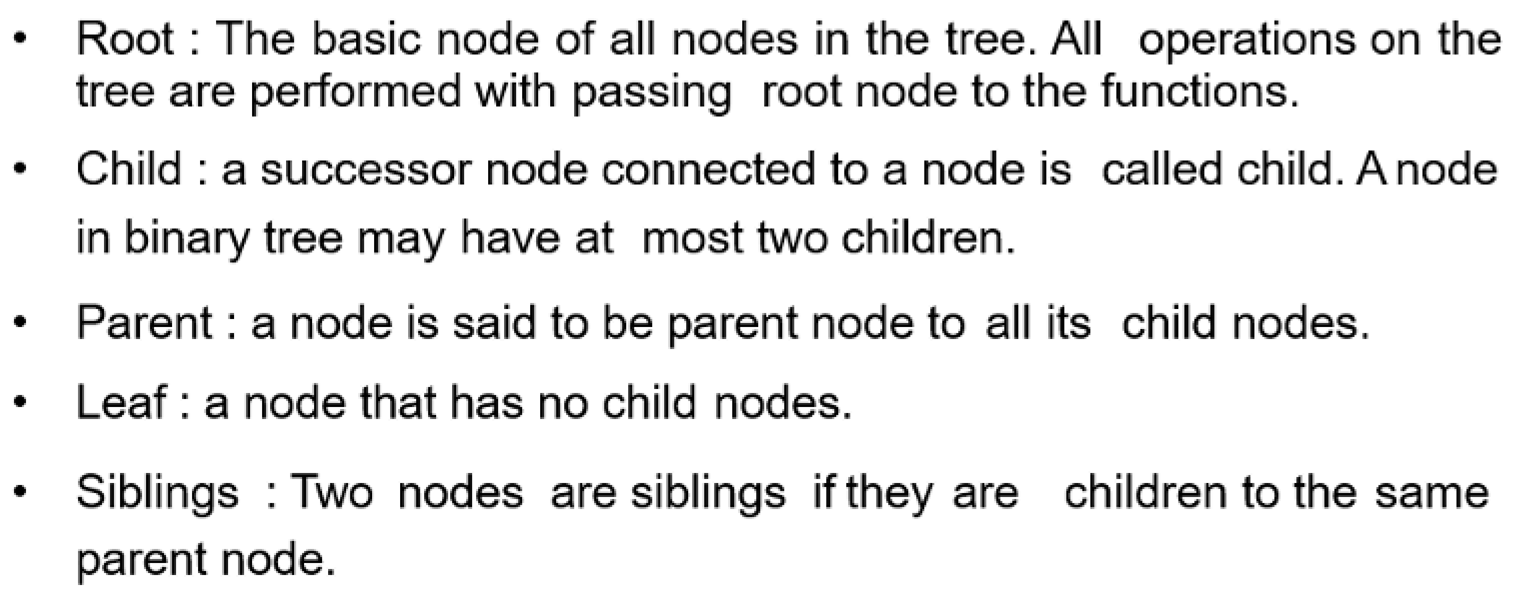
A. Pre order (VLR):In this traversal method, the root node is visited first, then the left subtree and finally the right subtree. Give any example

In order (LVR):In this traversal method, the left subtree is visited first, then the root and later the right subtree. We should always remember that every node may represent a subtree itself. If a binary tree is traversed in-order, the output will produce sorted key values in an ascending order. Give any example.

Post Order (LRV): In this traversal method, the root node is visited last, hence the name. First we traverse the left subtree, then the right subtree and finally the root node. Give any example.

Q12) Write the basic tree terminologies and the properties of binary trees?

A.



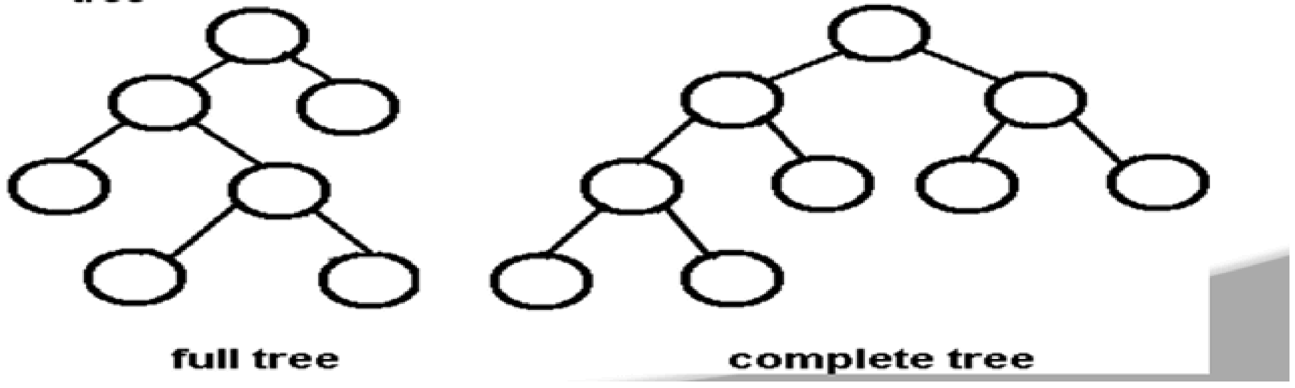
- Root : The basic node of all nodes in the tree. All operations on the tree are performed with passing root node to the functions.
- Child : a successor node connected to a node is called child. A node in binary tree may have at most two children.
- Parent : a node is said to be parent node to all its child nodes.
- Leaf : a node that has no child nodes.
- Siblings : Two nodes are siblings if they are children to the same parent node.

- Ancestor : a node which is parent of parent node (A is ancestor node to D,E and F).
- Descendent : a node which is child of child node (D, E and F are descendent nodes of node A)
- Level : The distance of a node from the root node, The root is at level – 0,(B and C are at Level 1 and D, E, F have Level 2 (highest level of tree is called height of tree)
- Degree : The number of nodes connected to a particular parent node.

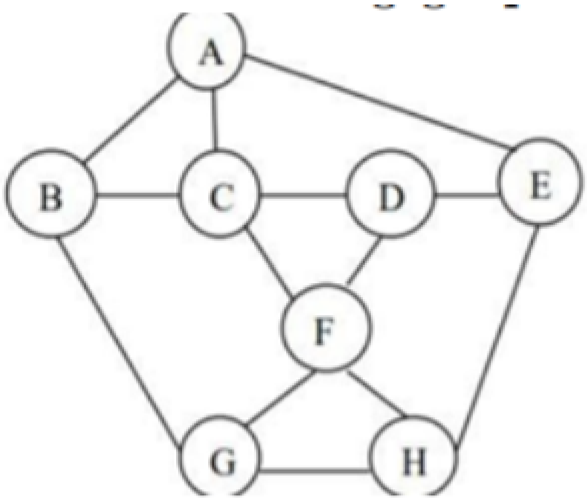
Binary Tree

- A binary tree is a hierarchy of nodes, where every parent node has at most two child nodes. There is a unique node, called the root, that does not have a parent.
- A binary tree can be defined recursively as
- Root node
- Left subtree: left child and all its descendants
- Right subtree: right child and all its descendants

- A full tree is a binary tree in which
  - Number of nodes at level *l is 2l–1*
  - Total nodes in a full tree of height *n is*
- A complete tree of height *n is a binary tree*
  - Number of nodes at level 1 *l n–1 is 2l–1*
  - Leaf nodes at level *n occupy the* leftmost positions in the tree

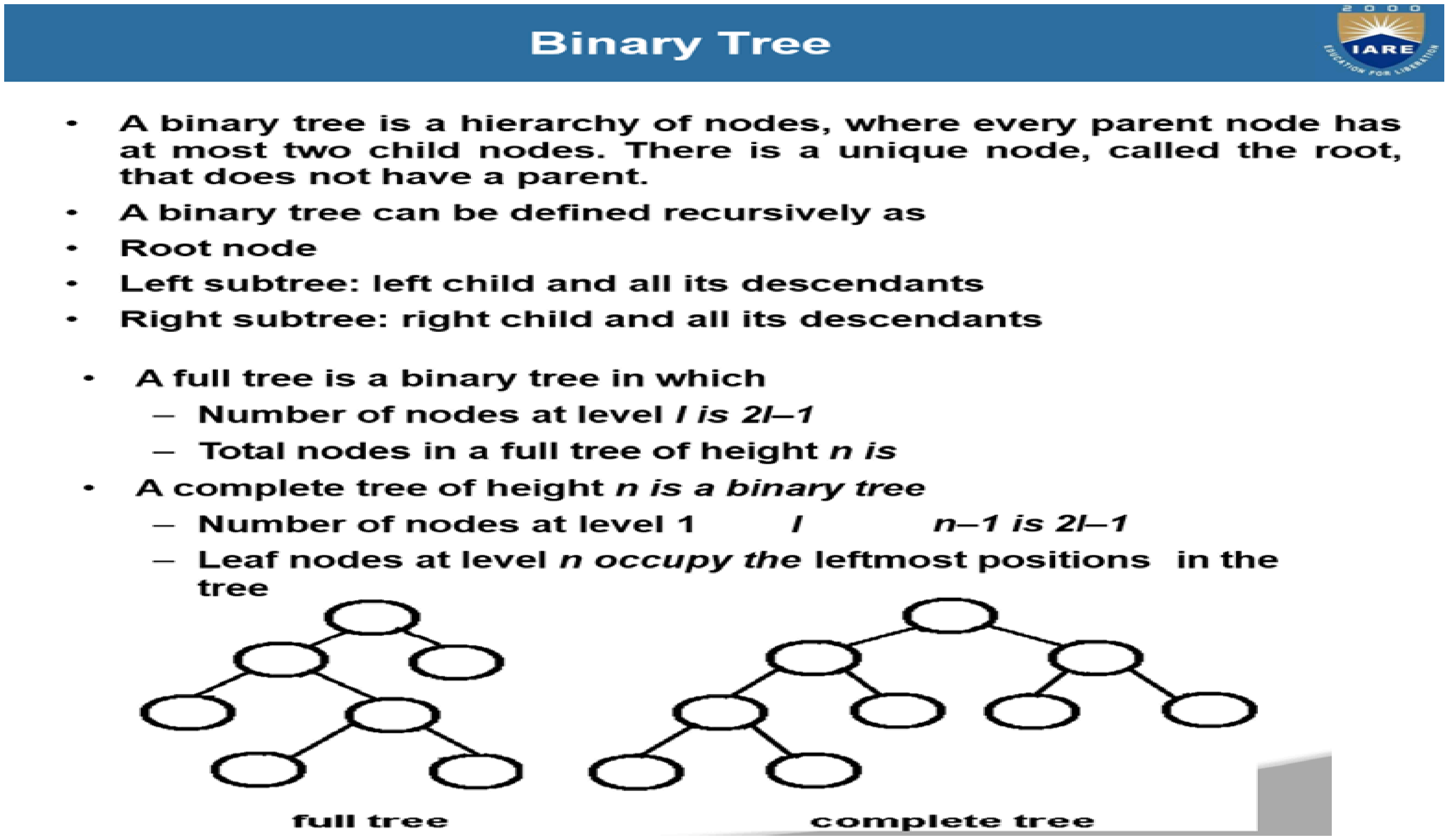


Q13) Explain the breadth first search and depth first search graph traversal algorithms for the following graph?



A. BFS: ABC GFDH
DFS: A B C D E H G F

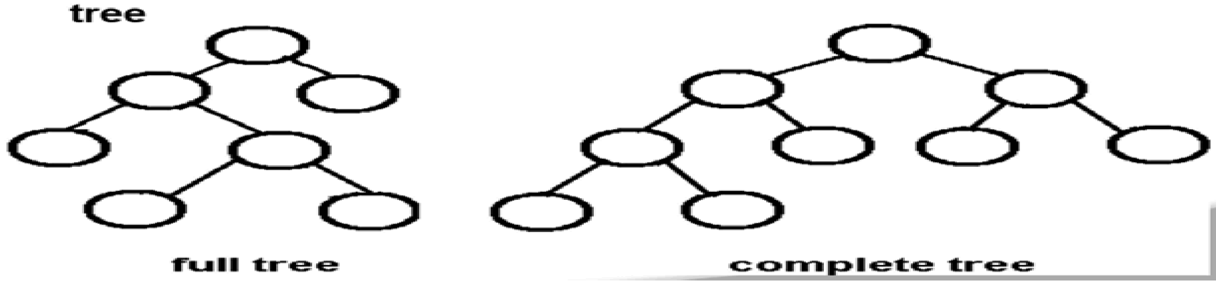
Q14) Explain the following with an example: i. Full binary tree ii. Strictly binary tree iii. Complete binary tree
A.



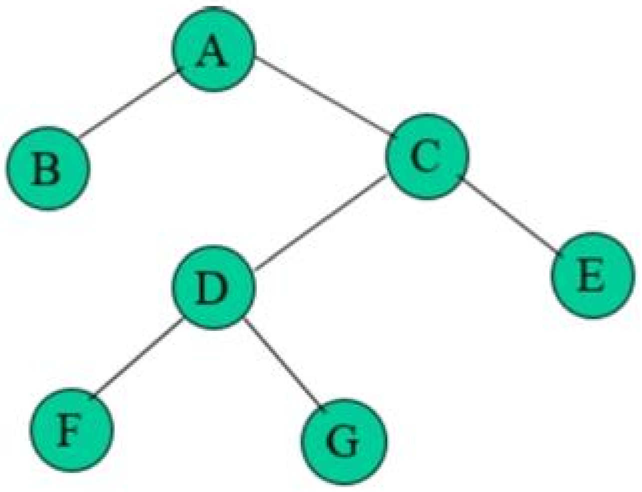
Binary Tree

- A binary tree is a hierarchy of nodes, where every parent node has at most two child nodes. There is a unique node, called the root, that does not have a parent.
- A binary tree can be defined recursively as
- Root node
- Left subtree: left child and all its descendants
- Right subtree: right child and all its descendants

- A full tree is a binary tree in which
  - Number of nodes at level *l is 2l–1*
  - Total nodes in a full tree of height *n is*
- A complete tree of height *n is a binary tree*
  - Number of nodes at level 1 *l n–1 is 2l–1*
  - Leaf nodes at level *n occupy the* leftmost positions in the tree



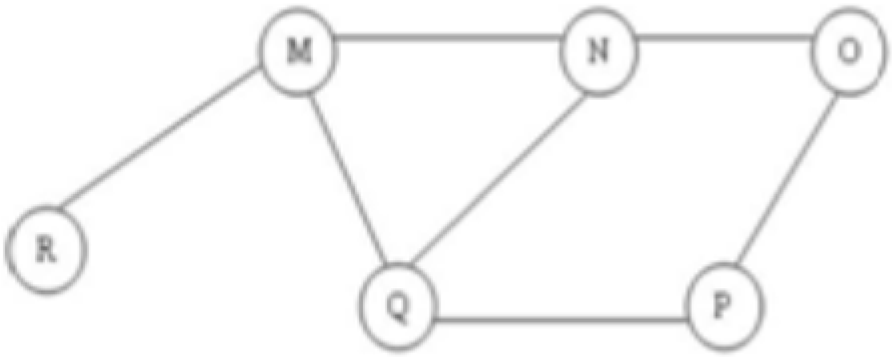
● If every non leaf node in a binary tree has nonempty left and right subtrees, the tree is called a strictly binary tree.
● A strictly binary tree with n leaves always contains 2n 1 nodes.



Q15) Write the applications of trees and graphs?
A. Graphs
● In Data Structures, graphs are used to represent the flow of computation.
● Google maps uses graphs for building navigation systems, where intersection of two(or more) roads are considered to be a vertex and the road connecting two vertices is considered to be an edge, thus their navigation system is based on the algorithm to calculate the shortest path between two vertices.
Trees
● Storing naturally hierarchical data: Trees are used to store the data in the hierarchical structure. For example, the file system. The file system stored on the disc drive, the file and folder are in the form of the naturally hierarchical data and stored in the form of trees.
● Organize data: It is used to organize data for efficient insertion, deletion and searching. For example, a binary tree has a logN time for searching an element.

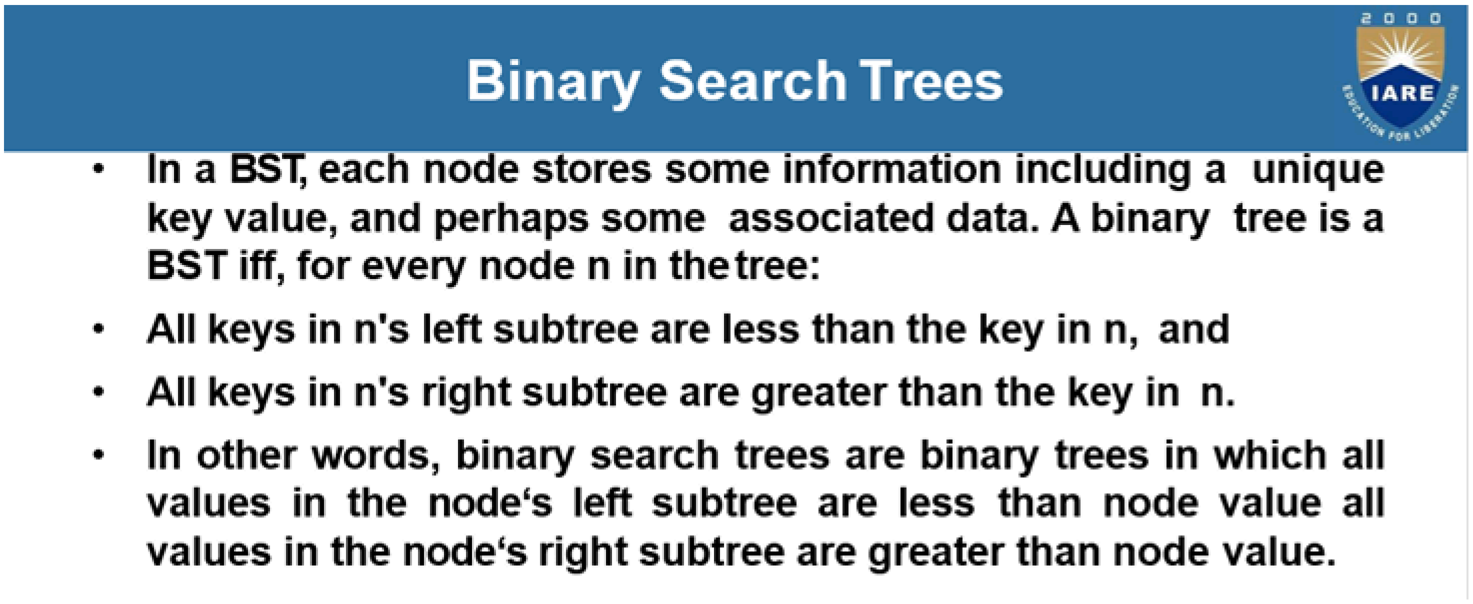
● Heap: It is also a tree data structure implemented using arrays. It is used to implement priority queues.

Q16) The Breadth First Search algorithm has been implemented using the queue data structure. Discover breadth first search for the graph shown in Figure with starting node M.



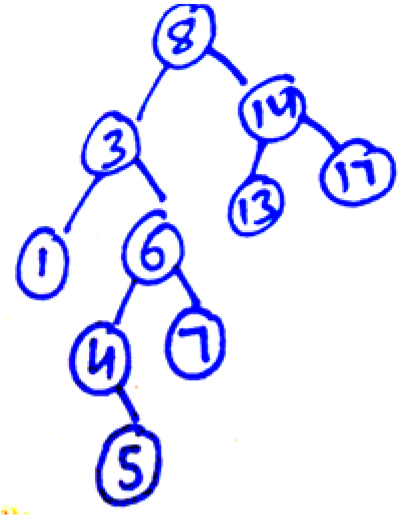
A.
BFS: M R Q N P O

Q17) Define a binary search tree and write the properties of a binary search tree? Construct a binary search with the following keys: 8, 3, 1, 6, 14, 4, 7, 13, 17, 5

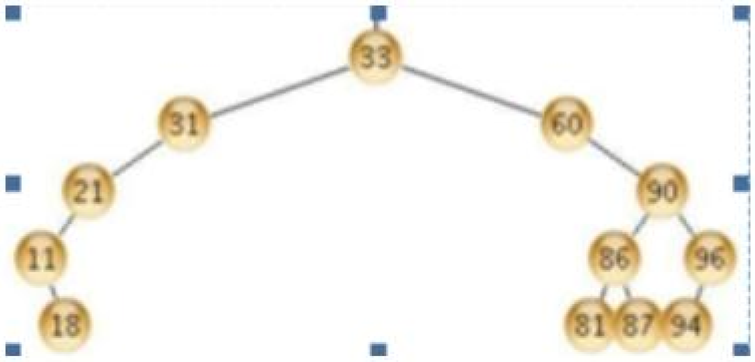


Binary Search Trees

- In a BST, each node stores some information including a unique key value, and perhaps some associated data. A binary tree is a BST iff, for every node n in the tree:
- All keys in n's left subtree are less than the key in n, and
- All keys in n's right subtree are greater than the key in n.
- In other words, binary search trees are binary trees in which all values in the node‘s left subtree are less than node value all values in the node‘s right subtree are greater than node value.



Q18) Write the procedure for finding an element 85in a given binary search tree?



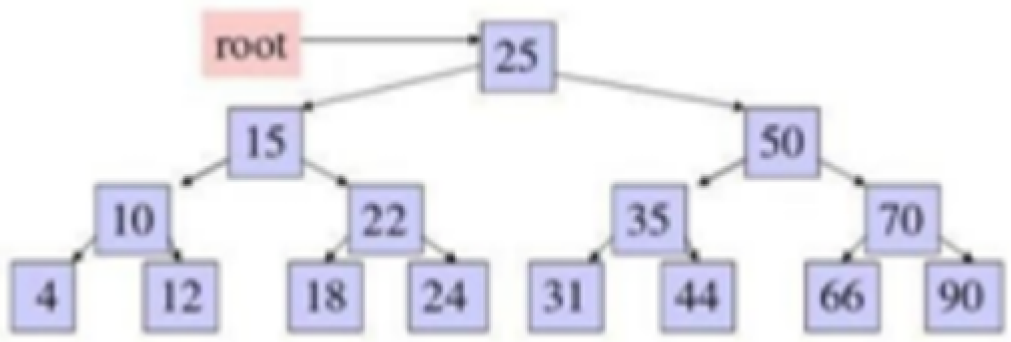
A.

- The element to be searched is 85.
- Compare 85 with the root node.
- Since 85 33 we can move to the right subtree.
- Compare 85 with 60. From 85-60 we can move to the right subtree. Since 85 to 90 we can move to the left subtree.
- Since 85 86 we can move to the left subtree.
- 85 81 but 81 has no children hence 85 is not present in the binary tree.

Q19) Write a program for Breadth First Search traversal of a graph?
A.

```
  import java.util.*;

class Graph {
    private int vertices; // Number of vertices
    private LinkedList<Integer> adjList[]; // Adjacency List

    // Constructor
    Graph(int v) {
        vertices = v;
        adjList = new LinkedList[v];
        for (int i = 0; i < v; ++i) {
            adjList[i] = new LinkedList<>();
        }
    }

    // Method to add an edge to the graph
    void addEdge(int v, int w) {
        adjList[v].add(w);
    }

    // Method to perform BFS traversal from a given source
    void BFS(int startVertex) {
        boolean visited[] = new boolean[vertices];
        LinkedList<Integer> queue = new LinkedList<>();

        visited[startVertex] = true;
        queue.add(startVertex);

        while (queue.size() != 0) {
            startVertex = queue.poll();
            System.out.print(startVertex + " ");

            Iterator<Integer> i = adjList[startVertex].listIterator();
            while (i.hasNext()) {
                int n = i.next();
                if (!visited[n]) {
                    visited[n] = true;
                    queue.add(n);
                }
            }
        }
    }

    public static void main(String args[]) {
        Graph g = new Graph(4);

        g.addEdge(0, 1);
        g.addEdge(0, 2);
        g.addEdge(1, 2);
        g.addEdge(2, 0);
        g.addEdge(2, 3);
        g.addEdge(3, 3);

        System.out.println("Breadth First Traversal starting from vertex
2:");

        g.BFS(2);
    }
}
```

Q20) Write the in-order, pre-order and post-order traversal of a given tree?



A.

In = LVR

4 10 12 15 18 22 24 25 31 35 44 50 66 70 90

Pre = VLR

25 15 10 4 12 22 18 24 50 35 31 44 70 66 90

Post = LRV

4 12 10 18 24 22 15 31 44 35 66 90 70 50 25