Assignment 3

Name:
Roll No:

Title

Implementing Depth First Search and Breadth First Search Algorithms

Problem Statement: Implement Depth First Search (DFS) and Breadth First Search (BFS) algorithms for traversing an undirected graph or tree data structure. Compare and analyze their performance and traversal orders.

Objective: The objective of this lab assignment is to gain a practical understanding of Depth First Search and Breadth First Search algorithms, their implementation, and their differences in terms of traversal order and performance.

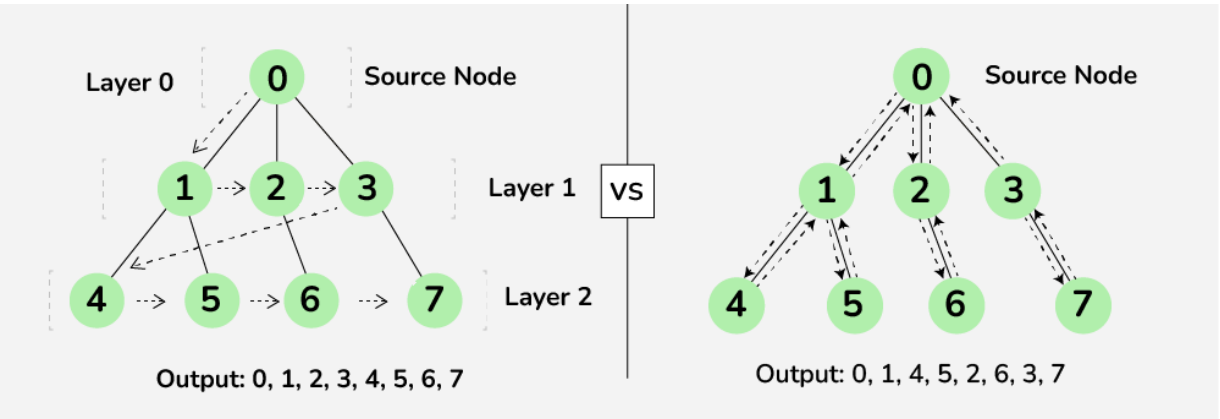
Outcome: By completing this assignment, students will:

- Understand the concepts of DFS and BFS algorithms.
- Gain hands-on experience in implementing DFS and BFS algorithms.
- Compare and contrast the traversal orders produced by DFS and BFS.
- Analyze the time and space complexities of both algorithms.

Input: An undirected graph or tree represented using adjacency list or matrix.

Output: The traversal orders and paths obtained using Depth First Search and Breadth First Search algorithms.

BFS **DFS**



Theory and Algorithms:

1. Depth First Search (DFS) Algorithm: DFS is a graph traversal algorithm that explores as far as possible along each branch before backtracking. It can be implemented using recursion or an explicit stack.

2. Breadth First Search (BFS) Algorithm: BFS is a graph traversal algorithm that explores the neighbour vertices at the present depth before moving on to vertices at the next depth level. It can be implemented using a queue.

Lab Procedure:

1. Prepare the graph or tree data structure for testing the algorithms.
2. Implement the DFS algorithm
3. Implement the BFS algorithm
4. Apply both algorithms to the prepared graph or tree data structure.
5. Record the traversal orders and paths obtained from both algorithms.
6. Analize and compare the traversal orders.
7. Document your observations and analysis in the lab report.

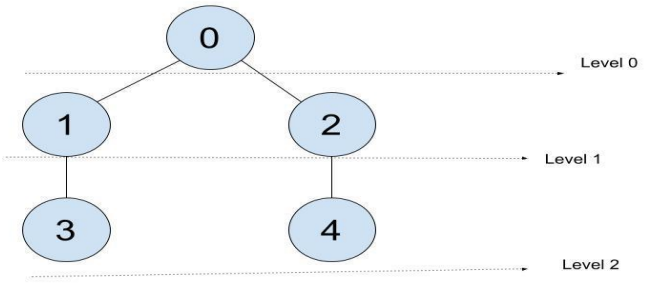
Conclusion: In this lab assignment, we successfully implemented the Depth First Search (DFS) and Breadth First Search (BFS) algorithms for traversing undirected graphs or tree data structures. We observed that DFS explores as far as possible before backtracking, while BFS explores neighbour vertices at each depth level. The choice between these algorithms depends on the application's requirements.

SET A

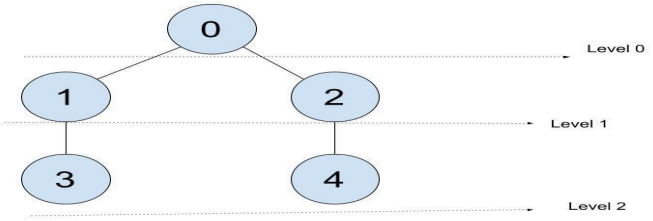
1. Find the level of given node in an undirected Graph.(use BFS)

→Given an undirected graph with **V** vertices and **E** edges and a node **X**, The task is to find the level of node **X** in the undirected graph. If X does not exist in the graph then return -1.

Input: V = 5, Edges = {{0, 1}, {0, 2}, {1, 3}, {2, 4}}, X = 3
Output: 2
Explanation: Starting from vertex 0, at level 0 we have node 0, at level 1 we have nodes 1 and 2 and at level 2 we have nodes 3 and 4. So the answer is 2



The example graph
Input: *V = 5, Edges = {{0, 1}, {0, 2}, {1, 3}, {2, 4}}, X = 5*
Output: *-1*
Explanation: *Vertex 5 is not present in the given graph so answer is -1*

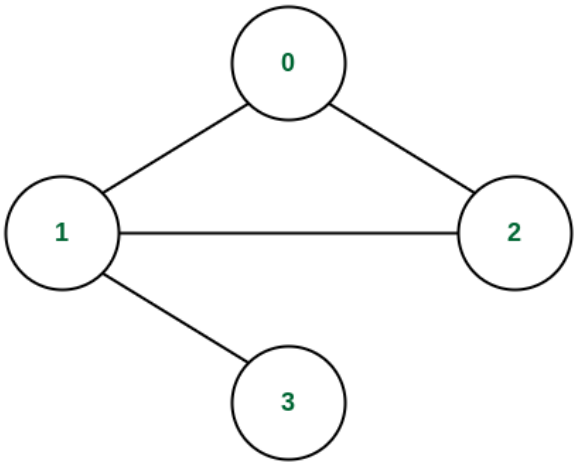


- Find the maximum vertex of the graph and store it in a variable (say **maxVertex**).
- create adjacency list **adj[]** of size **maxVertex+1**.
- Check if **X** is present or not, if not then return **-1**.
- To traverse the graph, create a queue for level order traversal.
- Push vertex 0 in a queue, and set a counter level to 0.
- Create a visited array of size **maxVertex+1** to mark the visited nodes.
- Start BFS traversal if the value of **X** is found in front of the queue then return the **level**.
    - Keep popping nodes from the queue till it becomes empty and increment the counter level

- In one iteration, push all the unvisited nodes in the queue connected with the current node
- Increment the level variable to jump to the next level.

2.Check if all nodes of Undirected Graph can be visited from given Node

Given an undirected graph consisting of N nodes labelled from 0 to N – 1, which are represented by a 2D array array[][], where array[i] represents all the nodes that are connected to the ith node, the task is to find whether we can visit all the nodes from node X.



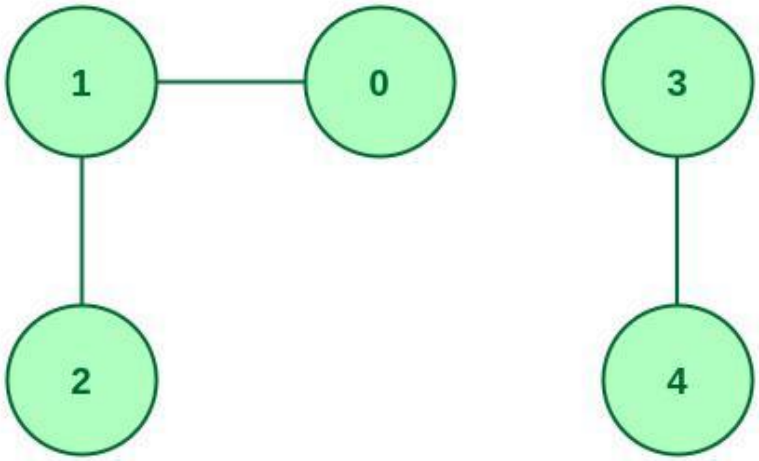
Examples:
Input: array = { { 1, 2 }, { 0, 3, 2 }, {0, 1}, {1} }, N = 4, X = 0
Output: YES
Explanation: As can be seen from the below graph, we can reach to any node from node 0

Input: array= { { 1, 2 }, { 0, 3, 2 }, {0, 1}, {1} }, N = 5, X = 0
Output: NO
Explanation: No node is connected to the node 4.

3.Given an undirected graph, the task is to print all the connected components line by line(use DFS)

Input: Consider the following graph

- Initialize all vertices as not visited.
- Do the following for every vertex **v**:
    - If **v** is not visited before, call the DFS. and print the newline character to print each component in a new line
        - Mark **v** as visited and print **v**.
        - For every adjacent **u** of **v**, If **u** is not visited, then recursively call the DFS



Assignment Evaluation :

0: Not Done [ ] 1: Incomplete [ ] 2: Late Complete [ ]

3: Needs Improvement [ ] 4: Complete [ ] 5: Well done [ ]

Signature of the Instructor: ---------- Date ----------