Green University of Bangladesh

Department of Computer Science and Engineering (CSE)
Faculty of Sciences and Engineering
Semester: (Spring,Year:2025),B.Sc.in CSE (Day)

LAB REPORT NO - 01
Course Title: Algorithms
Course Code: CSE206 Section:232-D5

Lab Experiment Name : Graph Algorithms - Cycle Detection and Topological Sorting

Student Details

	Name	ID
1.	MD.SHAJALAL	223002088

Lab Date : 18 - 02 - 2025
Submission Date : 25 – 02 - 2025
Course Teacher's Name : Md.Abu Rumman Refat

Lab Report Status
Marks: ………………………………… **Signature:.....................**
Comments:.. **Date:.............................**

1. TITLE OF THE LAB REPOT EXPERIMENT

Graph Algorithms - Cycle Detection and Topological Sorting

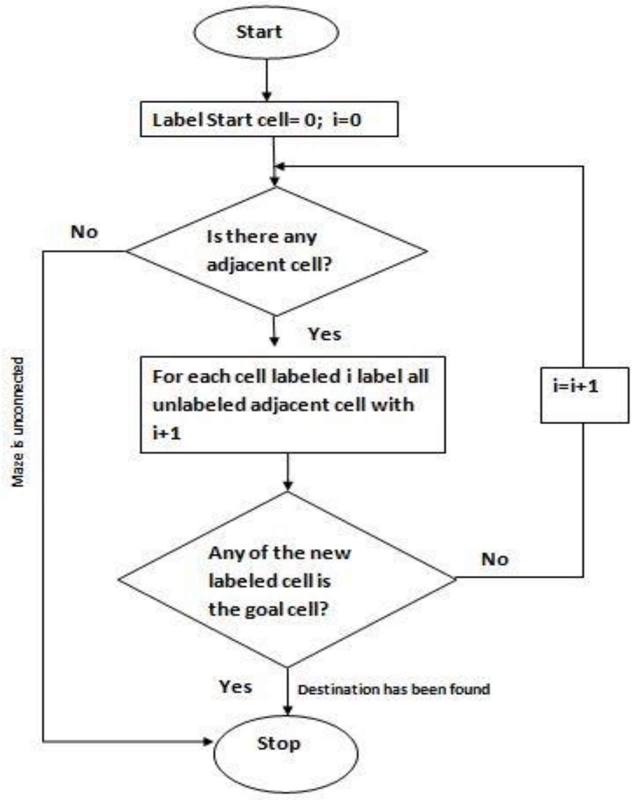
2. OBJECTIVE :

The objective of this lab is to implement graph algorithms in Java to:

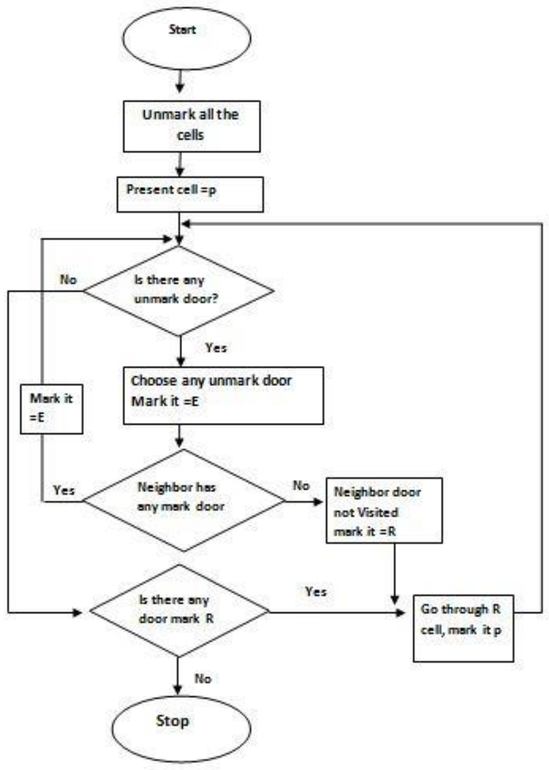
1. Detect cycles in a directed graph using the Breadth-First Search (BFS) algorithm.
2. Perform topological sorting on a directed graph using the Depth-First Search (DFS) algorithm.

3. PROCEDURE / ANALYSIS / DESIGN:

Flow chart: BFS


Start
Label Start cell= 0; i=0
Is there any adjacent cell?
No
Yes
For each cell labeled i label all unlabeled adjacent cell with i+1
i=i+1
Maze is unconnected
Any of the new labeled cell is the goal cell?
No
Yes
Destination has been found
Stop

DFS



Start
Unmark all the cells
Present cell =p
Is there any unmark door?
No
Yes
Choose any unmark door Mark it =E
Mark it =E
Neighbor has any mark door
Yes
No
Neighbor door not Visited mark it =R
Is there any door mark R
Yes
Go through R cell, mark it p
No
Stop

IMPLEMENTATION:

Code:

BFS

```
import java.util.*;

public class BFSAdjacencyMatrix {

    private int[][] adjacencyMatrix;

    private int numVertices;

    public BFSAdjacencyMatrix(int numVertices) {

        this.numVertices = numVertices;

        adjacencyMatrix = new int[numVertices][numVertices];

    }
```

```
// consider this as an undirected graph
public void addEdge(int source, int destination) {
    adjacencyMatrix[source][destination] = 1;
    adjacencyMatrix[destination][source] = 1;
}
public void bfs(int startVertex, int endVertex) {
    boolean[] visited = new boolean[numVertices];
    int[] parent = new int[numVertices];
    Queue<Integer> queue = new LinkedList<>();
    visited[startVertex] = true;
    queue.add(startVertex);
    while (!queue.isEmpty()) {
        int currentVertex = queue.poll();
        for (int i = 0; i < numVertices; i++) {
            if (adjacencyMatrix[currentVertex][i] == 1 && !visited[i]) {
                visited[i] = true;
                parent[i] = currentVertex;
                queue.add(i);
            }
        }
    }
    if (!visited[endVertex]) {
        System.out.println("No path exists between " + startVertex + " and " + endVertex);
    } else {
        System.out.println("Path from " + startVertex + " to " + endVertex + ":");
```

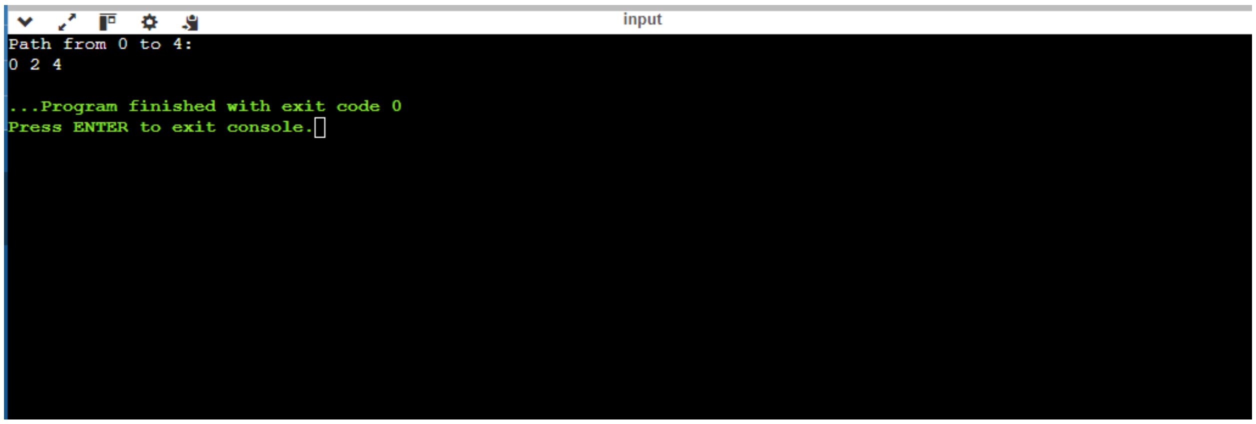
```
            printPath(parent, startVertex, endVertex);
        }
    }
    private void printPath(int[] parent, int startVertex, int endVertex) {
        if (endVertex == startVertex || parent[endVertex] == -1) {
            System.out.print(endVertex + " ");
            return;
        }
        printPath(parent, startVertex, parent[endVertex]);
        System.out.print(endVertex + " ");
    }
    public static void main(String[] args) {
        int numVertices = 5;
        BFSAdjacencyMatrix graph = new BFSAdjacencyMatrix(numVertices);
        graph.addEdge(0, 1);
        graph.addEdge(0, 2);
        graph.addEdge(1, 3);
        graph.addEdge(2, 4);
        int startVertex = 0;
        int endVertex = 4;
        graph.bfs(startVertex, endVertex);
    }
}
```

DFS:

```
import java.util.*;

public class DFSAdjacencyList {

    private int numVertices;

    private LinkedList<Integer>[] adjacencyList;

    public DFSAdjacencyList(int numVertices) {

        this.numVertices = numVertices;

        adjacencyList = new LinkedList[numVertices];

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

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

        }

    }

    public void addEdge(int source, int destination) {

        adjacencyList[source].add(destination);

    }

    private boolean isCyclicUtil(int vertex, boolean[] visited, boolean[] recStack) {

        if (recStack[vertex]) {

            return true; // Found a cycle

        }

        visited[vertex] = true;

        recStack[vertex] = true;

        for (Integer neighbor : adjacencyList[vertex]) {

            if (!visited[neighbor] && isCyclicUtil(neighbor, visited, recStack)) {

                return true;

            }
        }
        recStack[vertex] = false;
        return false;
    }
    public boolean isCyclic() {
        boolean[] visited = new boolean[numVertices];
        boolean[] recStack = new boolean[numVertices];
        for (int i = 0; i < numVertices; i++) {
            if (!visited[i] && isCyclicUtil(i, visited, recStack)) {
                return true;
            }
        }
        return false;
    }
    public static void main(String[] args) {
        int numVertices = 4;
        DFSAdjacencyList graph = new DFSAdjacencyList(numVertices);
        graph.addEdge(0, 1);
        graph.addEdge(1, 2);
        graph.addEdge(2, 3);
        graph.addEdge(3, 0);
        if (graph.isCyclic()) {
            System.out.println("Graph contains a cycle");
        } else {
            System.out.println("Graph does not contain a cycle");
        }
    }
}
```

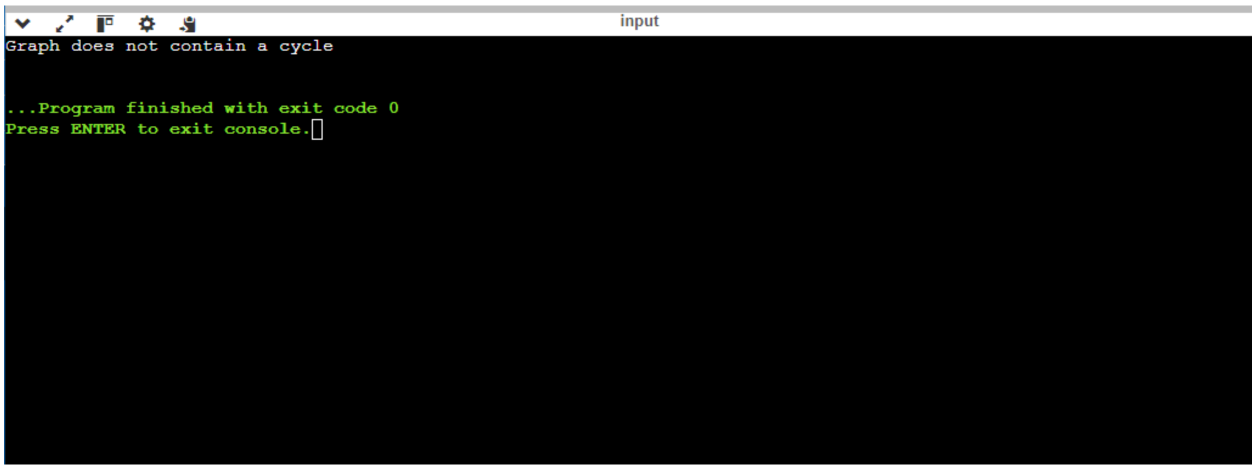
Output:-

BFS



```
Path from 0 to 4:
0 2 4

...Program finished with exit code 0
Press ENTER to exit console.
```

DFS



```
Graph does not contain a cycle


...Program finished with exit code 0
Press ENTER to exit console.
```

ANALYSIS AND DISCUSSION:

- BFS stands for Breath First Search. This search algorithm works level wise. Here my task was implementation of BFS by using adjacency matrix and find the path between two nodes. First, I input the number of node and edges then build a graph using node and edges source, destination. Then apply BFS algorithm on this graph and find the path between two nodes.

- DFS stands for Depth First Search. This search algorithm works in depth of graph's nodes. Here my task was implementation of DFS using adjacency list and check the given graph has any cycle or not. Same as BFS i take input and build a graph. Then apply DFS algorithm and find the path. If i visited any node in the previous, then at other when I find the same node from different path. It is called cycle graph. Here i found that cycle. If it available, then we can say it has a cycle else it has not any cycle