

**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**

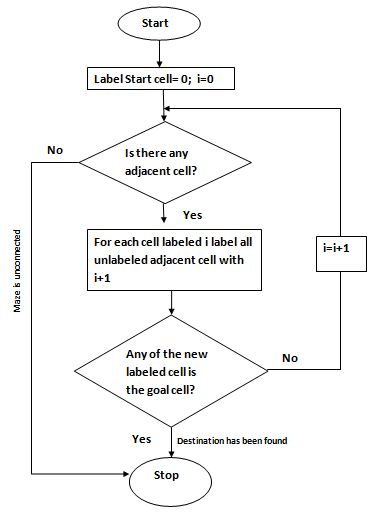
|  |
| --- |
| **Lab Report Status**  **Marks: ………………………………… Signature:.....................**  **Comments:.............................................. Date:..............................** |

1. **TITLE OF THE LAB REPOT EXPERIMENT**

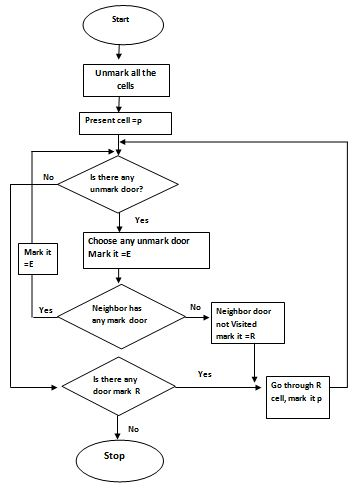
### **OBJECTIVE :**

# ****3.**** PROCEDURE / ANALYSIS / DESIGN:

**Flow chart: BFS**



DFS



# 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 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);

}

}

}

}

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);

}

}

**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);

}

}

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;

}

}

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**

**DFS**

### 

# ANALYSIS AND DISCUSSION:

* **BFS stands for Breath First Search.**
* **DFS stands for Depth First Search. This search algorithm works in depth of graph's nodes.**