****

**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 1**

**Course Title: Algorithm lab**

**Course Code: CSE 206 Section:D9**

**Lab Experiment Name:** Detecting Cycles in a Graph using BFS and Performing Topological Sorting using DFS

**Student Details**

|  |  |  |
| --- | --- | --- |
| **Name** | | **ID** |
| **1.** | Rukonuzzaman Topu | 232002280 |

**Lab Date :**

**Submission Date : 05/03/2025**

**Course Teacher’s Name : Farjana Akter Jui**

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

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

This lab experiment involves implementing two graph traversal techniques:

1.Detecting Cycles in a Graph using BFS.

2.Performing Topological Sorting using DFS.

Both tasks focus on understanding the applications of graph algorithms and their respective implementations in Java.

**2. OBJECTIVES/AIM**

1. Understand how to traverse graphs using BFS and DFS.
2. Implement BFS to detect cycles in an undirected graph.
3. Implement DFS to perform a topological sort of a directed acyclic graph.
4. Gain hands-on experience with graph representations using adjacency lists.
5. Analyze the behavior of different traversal techniques and their applications.

**3. PROCEDURE**

**Procedure for Cycle Detection using BFS:**

1. Represent the graph using an adjacency list.
2. Use a queue to explore the graph breadth-first while tracking visited nodes.
3. If a node is revisited that is not the direct parent, a cycle is detected.
4. Output the detected cycle if found.

**Procedure for Topological Sort using DFS:**

1. Represent the graph using an adjacency list.
2. Use a stack to store nodes in topological order.
3. Apply DFS recursively to visit all nodes and push them onto the stack after visiting.
4. Print the stack content as the topological order.

**4. IMPLEMENTATION**

1. **Cycle Detection in a Graph using BFS**

import java.util.\*;

public class CycleDetectionBFS {

    static class Edge {

        int src, dest;

        public Edge(int s, int d) {

            this.src = s;

            this.dest = d;

        }

    }

    static void createGraph(ArrayList<Edge>[] graph) {

        for (int i = 0; i < graph.length; i++) {

            graph[i] = new ArrayList<>();

        }

        graph[0].add(new Edge(0, 1));

        graph[1].add(new Edge(1, 0));

        graph[1].add(new Edge(1, 2));

        graph[2].add(new Edge(2, 1));

        graph[2].add(new Edge(2, 3));

        graph[3].add(new Edge(3, 2));

        graph[3].add(new Edge(3, 4));

        graph[4].add(new Edge(4, 3));

        graph[4].add(new Edge(4, 1));

        graph[1].add(new Edge(1, 4));

    }

    static boolean isCyclicBFS(ArrayList<Edge>[] graph, int V) {

        boolean[] visited = new boolean[V];

        int[] parent = new int[V];

        Arrays.fill(parent, -1);

        for (int i = 0; i < V; i++) {

            if (!visited[i]) {

                if (bfsDetectCycle(graph, visited, parent, i)) {

                    return true;

                }

            }

        }

        return false;

    }

    static boolean bfsDetectCycle(ArrayList<Edge>[] graph, boolean[] visited, int[] parent, int src) {

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

        queue.add(src);

        visited[src] = true;

        while (!queue.isEmpty()) {

            int node = queue.poll();

            for (Edge edge : graph[node]) {

                int neighbor = edge.dest;

                if (!visited[neighbor]) {

                    visited[neighbor] = true;

                    parent[neighbor] = node;

                    queue.add(neighbor);

                } else if (neighbor != parent[node]) {

                    System.out.print("Cycle detected: ");

                    printCyclePath(parent, node, neighbor);

                    return true;

                }

            }

        }

        return false;

    }

    static void printCyclePath(int[] parent, int start, int end) {

        List<Integer> cyclePath = new ArrayList<>();

        int node = start;

        while (node != -1) {

            cyclePath.add(node);

            if (node == end) break;

            node = parent[node];

        }

        Collections.reverse(cyclePath);

        cyclePath.add(end);

        for (int i = 0; i < cyclePath.size(); i++) {

            System.out.print(cyclePath.get(i));

            if (i < cyclePath.size() - 1) System.out.print(" -> ");

        }

        System.out.println();

    }

    public static void main(String[] args) {

        int V = 5;

        ArrayList<Edge>[] graph = new ArrayList[V];

        createGraph(graph);

        if (!isCyclicBFS(graph, V)) {

            System.out.println("No cycle found in the graph.");

        }

    }

}

2. **Topological Sorting using DFS**

import java.util.\*;

public class TopologicalSortDFS {

    static class Graph {

        private int V;

        private ArrayList<Integer>[] adj;

        public Graph(int V) {

            this.V = V;

            adj = new ArrayList[V];

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

                adj[i] = new ArrayList<>();

            }

        }

        public void addEdge(int src, int dest) {

            adj[src].add(dest);

        }

        private void topologicalSortUtil(int v, boolean visited[], Stack<Integer> stack) {

            visited[v] = true;

            for (int neighbor : adj[v]) {

                if (!visited[neighbor]) {

                    topologicalSortUtil(neighbor, visited, stack);

                }

            }

            stack.push(v);

        }

        public void topologicalSort() {

            Stack<Integer> stack = new Stack<>();

            boolean visited[] = new boolean[V];

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

                if (!visited[i]) {

                    topologicalSortUtil(i, visited, stack);

                }

            }

            System.out.print("Topological Order: ");

            while (!stack.isEmpty()) {

                System.out.print(stack.pop() + " ");

            }

            System.out.println();

        }

    }

    public static void main(String[] args) {

        Graph g = new Graph(6);

        g.addEdge(5, 2);

        g.addEdge(5, 0);

        g.addEdge(4, 0);

        g.addEdge(4, 1);

        g.addEdge(2, 3);

        g.addEdge(3, 1);

        System.out.println("Performing Topological Sort using DFS:");

        g.topologicalSort();

    }

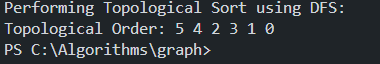
}

**5. TEST RESULT / OUTPUT**

**Cycle Detection using BFS Output:**



**Topological Sort using DFS Output:**



**6. DISCUSSION**

**Cycle Detection using BFS:**

* The algorithm successfully detects cycles in an undirected graph.
* It keeps track of visited nodes and parent nodes to identify back edges.
* Complexity: O(V + E)

**Topological Sort using DFS:**

* This implementation works only on Directed Acyclic Graphs (DAGs).
* DFS explores deeply before backtracking, storing nodes in stack.
* Complexity: O(V + E)

**Challenges Faced:**

* Understanding parent tracking for cycle detection.
* Ensuring proper stack operations in DFS-based sorting.