DataStructures and algorithms

Lab 11

Name:Pranjal bajaj

Cms:023-21-0022

Task 1

import java.util.\*;

public class graph {

    public boolean validPath(int n, int[][] edges, int source, int destination) {

        Map<Integer, List<Integer>> graph = new HashMap<>();

        for (int i = 0; i < n; i++) {

            graph.put(i, new ArrayList<>());

        }

        for (int[] edge : edges) {

            graph.get(edge[0]).add(edge[1]);

            graph.get(edge[1]).add(edge[0]);

        }

        System.out.println("Graph Representation:");

        for (int node : graph.keySet()) {

            System.out.println(node + " -> " + graph.get(node));

        }

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

        boolean[] visited = new boolean[n];

        queue.add(source);

        visited[source] = true;

        while (!queue.isEmpty()) {

            int current = queue.poll();

            if (current == destination) {

                return true;

            }

            for (int neighbor : graph.get(current)) {

                if (!visited[neighbor]) {

                    visited[neighbor] = true;

                    queue.add(neighbor);

                }

            }

        }

        return false;

    }

    public static void main(String[] args) {

        graph checker = new graph();

        int n1 = 6;

        int[][] edges1 = {{0, 1}, {0, 2}, {3, 5}, {5, 4}, {4, 3}};

        int source1 = 0;

        int destination1 = 5;

        System.out.println("Valid Path: " + checker.validPath(n1, edges1, source1, destination1));

        int n2 = 3;

        int[][] edges2 = {{0, 1}, {1, 2}, {2, 0}};

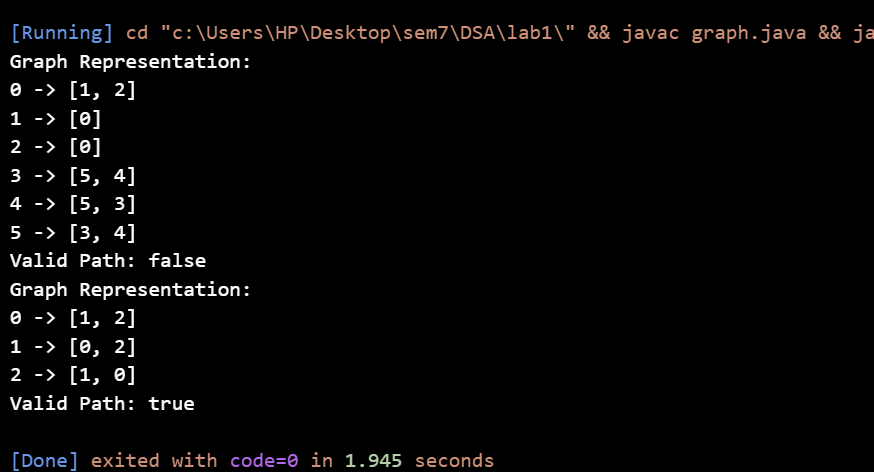
        int source2 = 0;

        int destination2 = 2;

        System.out.println("Valid Path: " + checker.validPath(n2, edges2, source2, destination2));

    }

}



Task 2

import java.util.\*;

public class graph {

    private Map<Integer, List<Integer>> graph;

    public graph(int n, int[][] edges) {

        graph = new HashMap<>();

        for (int i = 0; i < n; i++) {

            graph.put(i, new ArrayList<>());

        }

        for (int[] edge : edges) {

            graph.get(edge[0]).add(edge[1]);

            graph.get(edge[1]).add(edge[0]);

        }

    }

    public int getTotalNodes() {

        return graph.size();

    }

    public int getTotalEdges() {

        int edges = 0;

        for (int node : graph.keySet()) {

            edges += graph.get(node).size();

        }

        return edges / 2; // Divide by 2 to avoid double counting in an undirected graph

    }

    public void printGraph() {

        System.out.println("Graph Representation:");

        for (int node : graph.keySet()) {

            System.out.println(node + " -> " + graph.get(node));

        }

    }

    public static void main(String[] args) {

        int n = 6;

        int[][] edges = {{0, 1}, {0, 2}, {3, 5}, {5, 4}, {4, 3}};

        graph graph = new graph(n, edges);

        graph.printGraph();

        System.out.println("Total Nodes: " + graph.getTotalNodes());

        System.out.println("Total Edges: " + graph.getTotalEdges());

    }

}



Task 3

import java.util.\*;

public class graph {

    private Map<Integer, List<Integer>> graph;

    public graph(int n, int[][] edges) {

        graph = new HashMap<>();

        for (int i = 0; i < n; i++) {

            graph.put(i, new ArrayList<>());

        }

        for (int[] edge : edges) {

            graph.get(edge[0]).add(edge[1]);

            graph.get(edge[1]).add(edge[0]);

        }

    }

    public boolean hasCycle() {

        boolean[] visited = new boolean[graph.size()];

        for (int node : graph.keySet()) {

            if (!visited[node]) {

                if (dfs(node, -1, visited)) {

                    return true;

                }

            }

        }

        return false;

    }

    private boolean dfs(int current, int parent, boolean[] visited) {

        visited[current] = true;

        for (int neighbor : graph.get(current)) {

            if (!visited[neighbor]) {

                if (dfs(neighbor, current, visited)) {

                    return true;

                }

            } else if (neighbor != parent) {

                return true;

            }

        }

        return false;

    }

    public static void main(String[] args) {

        int n1 = 5;

        int[][] edges1 = {{0, 1}, {1, 2}, {2, 3}, {3, 4}};

        graph graph1 = new graph(n1, edges1);

        System.out.println("Graph 1 has cycle: " + graph1.hasCycle());

        int n2 = 5;

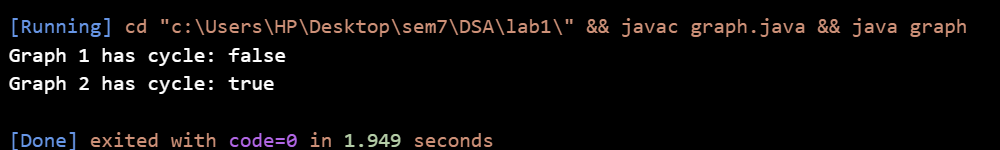
        int[][] edges2 = {{0, 1}, {1, 2}, {2, 0}, {3, 4}};

        graph graph2 = new graph(n2, edges2);

        System.out.println("Graph 2 has cycle: " + graph2.hasCycle());

    }

}



Task 4

import java.util.\*;

public class graph {

    private Map<Integer, List<Integer>> graph;

    public graph(int n, int[][] edges) {

        graph = new HashMap<>();

        for (int i = 0; i < n; i++) {

            graph.put(i, new ArrayList<>());

        }

        for (int[] edge : edges) {

            graph.get(edge[0]).add(edge[1]);

            graph.get(edge[1]).add(edge[0]);

        }

    }

    public List<Integer> findVerticesAtDistanceK(int start, int k) {

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

        boolean[] visited = new boolean[graph.size()];

        queue.add(start);

        visited[start] = true;

        int distance = 0;

        while (!queue.isEmpty() && distance < k) {

            int size = queue.size();

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

                int current = queue.poll();

                for (int neighbor : graph.get(current)) {

                    if (!visited[neighbor]) {

                        visited[neighbor] = true;

                        queue.add(neighbor);

                    }

                }

            }

            distance++;

        }

        return distance == k ? new ArrayList<>(queue) : new ArrayList<>();

    }

    public static void main(String[] args) {

        int n = 6;

        int[][] edges = {{0, 1}, {0, 2}, {1, 3}, {1, 4}, {2, 5}};

        graph graph = new graph(n, edges);

        int startVertex = 0;

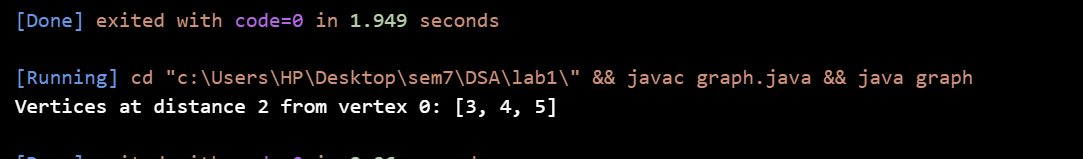
        int k = 2;

        System.out.println("Vertices at distance " + k + " from vertex " + startVertex + ": " +

                graph.findVerticesAtDistanceK(startVertex, k));

    }

}



Task 5

import java.util.\*;

public class graph {

    private Map<Integer, List<Integer>> graph;

    public graph(int n, int[][] edges) {

        graph = new HashMap<>();

        for (int i = 0; i < n; i++) {

            graph.put(i, new ArrayList<>());

        }

        for (int[] edge : edges) {

            graph.get(edge[0]).add(edge[1]);

            graph.get(edge[1]).add(edge[0]);

        }

    }

    public void printAllPaths(int source, int destination) {

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

        boolean[] visited = new boolean[graph.size()];

        System.out.println("All possible paths from " + source + " to " + destination + ":");

        findPaths(source, destination, visited, path);

    }

    private void findPaths(int current, int destination, boolean[] visited, List<Integer> path) {

        visited[current] = true;

        path.add(current);

        if (current == destination) {

            System.out.println(path);

        } else {

            for (int neighbor : graph.get(current)) {

                if (!visited[neighbor]) {

                    findPaths(neighbor, destination, visited, path);

                }

            }

        }

        path.remove(path.size() - 1);

        visited[current] = false;

    }

    public static void main(String[] args) {

        int n = 5;

        int[][] edges = {

            {0, 1}, {0, 2}, {1, 2}, {1, 3}, {2, 3}, {3, 4}

        };

        graph graph = new graph(n, edges);

        graph.printAllPaths(0, 4);

    }

}

