GRAPHS – I

Code: Has Path

import java.util.\*;

public class Solution {

public static void main(String[] args) throws NumberFormatException, IOException {

int n , e;

Scanner scan = new Scanner(System.in);

n =scan.nextInt();

e = scan.nextInt();

int edges[][] = new int[n][n];

int fv1=0 , sv1=0;

for(int i=0;i<e;i++)

{

fv1 = scan.nextInt();

sv1 = scan.nextInt();

edges[fv1][sv1] = 1;

edges[sv1][fv1] = 1;

}

int sv = scan.nextInt();

int ev = scan.nextInt();

System.out.println(BFS(edges , sv , ev));

}

public static boolean BFS(int edges[][],int sv, int ev)

{

boolean visited[] = new boolean[edges.length];

return printBFSHelper(edges , sv , ev , visited);

}

public static boolean printBFSHelper(int edges[][] , int sv, int ev , boolean visited[])

{

if(sv > (edges.length-1) || ev >(edges.length-1))

return false;

if(edges[sv][ev]==1)return true;

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

visited[sv] = true;

q.add(sv);

while(!q.isEmpty())

{

int front = q.remove();

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

{

if(edges[front][i] == 1 && !visited[i])

{

if(i==ev)return true;

else{

q.add(i);

visited[i]=true;

}

}

}

}

return false;

}

}

Code : Get Path – DFS

import java.util.\*;

public class Solution {

public static void main(String[] args) throws NumberFormatException, IOException {

int n , e;

Scanner scan = new Scanner(System.in);

n =scan.nextInt();

e = scan.nextInt();

int edges[][] = new int[n][n];

int fv1=0 , sv1=0;

for(int i=0;i<e;i++)

{

fv1 = scan.nextInt();

sv1 = scan.nextInt();

edges[fv1][sv1] = 1;

edges[sv1][fv1] = 1;

}

int sv = scan.nextInt();

int ev = scan.nextInt();

ArrayList<Integer> ans = (DFS(edges , sv , ev));

if(ans!=null)

for(int i:ans)

{

System.out.print(i+" ");

}

}

public static ArrayList<Integer> DFS(int edges[][],int sv, int ev)

{

boolean visited[] = new boolean[edges.length];

return printDFSHelper(edges , sv , ev , visited);

}

public static ArrayList<Integer> printDFSHelper(int[][] edges ,int sv ,int ev ,boolean[] visited)

{

if(sv==ev)

{

ArrayList<Integer> ans = new ArrayList<>();

ans.add(sv);

return ans;

}

visited[sv] = true;

int n= edges.length;

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

if(edges[sv][i]==1 && !visited[i]){

ArrayList<Integer> smallAns = printDFSHelper(edges,i,ev,visited);

if(smallAns!=null)

{

smallAns.add(sv);

return smallAns;

}

}

}

return null;

}

}

Code : Is Connected ?

public class Solution {

static BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

public static void dfs(int[][] edges, int sv, boolean[] visited) {

visited[sv] = true;

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

if(edges[sv][i] == 1 && !visited[i]) {

dfs(edges, i, visited);

visited[i] = true;

}

}

}

public static boolean isConnected(int[][] edges) {

boolean[] visited = new boolean[edges.length];

dfs(edges, 0, visited);

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

if(!visited[i]) {

return false;

}

}

return true;

}

public static void main(String[] args) throws NumberFormatException, IOException {

String[] strNums;

strNums = br.readLine().split("\\s");

int n = Integer.parseInt(strNums[0]);

int e = Integer.parseInt(strNums[1]);

if(n == 0) {

System.out.println("true");

return;

}

int edges[][] = new int[n][n];

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

String[] strNums1;

strNums1 = br.readLine().split("\\s");

int fv = Integer.parseInt(strNums1[0]);

int sv = Integer.parseInt(strNums1[1]);

edges[fv][sv] = 1;

edges[sv][fv] = 1;

}

System.out.println(isConnected(edges));

}

}

Code : All Connected Components

class VertexOutOfRangeException extends Exception {

public String toString() {

return "Valid input for the vertex in specified range is expected!";

}

}

public class Solution {

static BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

public static void connectedComponents(Map<Integer, ArrayList<Integer>> adjacencyList, Map<Integer, Boolean> visited, ArrayList<Integer> smallOutput, int vertex) {

smallOutput.add(vertex);

visited.put(vertex, true);

ArrayList<Integer> adjVerticies = adjacencyList.get(vertex);

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

if(!visited.containsKey(adjVerticies.get(i))) {

connectedComponents(adjacencyList, visited, smallOutput, adjVerticies.get(i));

}

}

}

public static ArrayList<ArrayList<Integer>> allConnectedComponents(Map<Integer,

ArrayList<Integer>> adjacencyList) {

Map<Integer, Boolean> visited = new HashMap<>();

ArrayList<ArrayList<Integer>> output = new ArrayList<>();

int i = 0;

while(visited.size() != adjacencyList.size()) {

while(i < adjacencyList.size()) {

if(!visited.containsKey(i)) {

ArrayList<Integer> smallOutput = new ArrayList<>();

connectedComponents(adjacencyList, visited, smallOutput, i);

output.add(smallOutput);

}

i++;

} }

return output; }

public static void main(String[] args) throws VertexOutOfRangeException, IOException{

String[] strNums;

strNums = br.readLine().split("\\s");

int noOfVertices = Integer.parseInt(strNums[0]);

int noOfEdges = Integer.parseInt(strNums[1]);

Map<Integer, ArrayList<Integer>> adjacencyList = new HashMap<>();

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

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

}

int currentEntry = 1;

while(currentEntry <= noOfEdges) {

String[] strNums1;

strNums1 = br.readLine().split("\\s");

int source = Integer.parseInt(strNums1[0]);

int destination = Integer.parseInt(strNums1[1]);

ArrayList<Integer> edgeListForDestination = adjacencyList.get(source);

ArrayList<Integer> edgeListForSource = adjacencyList.get(destination);

if(edgeListForDestination != null && edgeListForSource != null) {

edgeListForDestination.add(destination);

edgeListForSource.add(source);

} else {

throw new VertexOutOfRangeException();

}

currentEntry++;

}

ArrayList<ArrayList<Integer>> allConnectedComponents = allConnectedComponents(adjacencyList);

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

ArrayList<Integer> components = allConnectedComponents.get(i);

Collections.sort(components);

for(int k = 0; k < components.size(); k++) {

System.out.print(components.get(k) + " ");

}

System.out.println();

} } }