Code : 1  
package Topic\_18\_Graphs;

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.util.ArrayList;

public class A\_HasPath {

static class Edge {

int src;

int nbr;

int wt;

Edge(int src, int nbr, int wt) {

this.src = src;

this.nbr = nbr;

this.wt = wt;

}

}

public static void main(String[] args) throws Exception {

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

int vtces = Integer.parseInt(br.readLine());

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

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

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

}

int edges = Integer.parseInt(br.readLine());

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

String[] parts = br.readLine().split(" ");

int v1 = Integer.parseInt(parts[0]);

int v2 = Integer.parseInt(parts[1]);

int wt = Integer.parseInt(parts[2]);

graph[v1].add(new Edge(v1, v2, wt));

graph[v2].add(new Edge(v2, v1, wt));

}

int src = Integer.parseInt(br.readLine());

int dest = Integer.parseInt(br.readLine());

boolean[] visited = new boolean[vtces];

// write your code here

boolean pathExists = hasPath(graph, visited, src, dest);

System.out.println(pathExists);

}

public static boolean hasPath(ArrayList<Edge>[] graph,

boolean[] visited, int src, int dest) {

if (src == dest) {

return true;

}

visited[src] = true;

for (int i = 0; i < graph[src].size(); i++) {

Edge edge = graph[src].get(i);

int nbr = edge.nbr;

if (visited[nbr] == false) {

boolean pathExists = hasPath(graph, visited, nbr, dest);

if (pathExists) {

return true;

}

}

}

return false;

}

}

Code : 2  
package Topic\_18\_Graphs;

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.util.ArrayList;

public class B\_PrintAllPath {

static class Edge {

int src;

int nbr;

int wt;

Edge(int src, int nbr, int wt) {

this.src = src;

this.nbr = nbr;

this.wt = wt;

}

}

public static void main(String[] args) throws Exception {

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

int vtces = Integer.parseInt(br.readLine());

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

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

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

}

int edges = Integer.parseInt(br.readLine());

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

String[] parts = br.readLine().split(" ");

int v1 = Integer.parseInt(parts[0]);

int v2 = Integer.parseInt(parts[1]);

int wt = Integer.parseInt(parts[2]);

graph[v1].add(new Edge(v1, v2, wt));

graph[v2].add(new Edge(v2, v1, wt));

}

int src = Integer.parseInt(br.readLine());

int dest = Integer.parseInt(br.readLine());

// write all your codes here

boolean[] visited = new boolean[vtces];

printAllPaths(graph, visited, src, dest, src + "");

}

public static void printAllPaths(ArrayList<Edge>[] graph, boolean[] visited,

int src, int dest, String psf) {

if (src == dest) {

System.out.println(psf);

return;

}

visited[src] = true;

for (int i = 0; i < graph[src].size(); i++) {

Edge e = graph[src].get(i);

int nbr = e.nbr;

if (visited[nbr] == false) {

printAllPaths(graph, visited, nbr, dest, psf + nbr + "");

}

}

visited[src] = false;

}

}

Code : 3  
package Topic\_18\_Graphs;

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.util.ArrayList;

import java.util.PriorityQueue;

public class C\_MultiSolver {

static class Edge {

int src;

int nbr;

int wt;

Edge(int src, int nbr, int wt) {

this.src = src;

this.nbr = nbr;

this.wt = wt;

}

}

static class Pair implements Comparable<Pair> {

int wsf;

String psf;

Pair(int wsf, String psf) {

this.wsf = wsf;

this.psf = psf;

}

public int compareTo(Pair o) {

return this.wsf - o.wsf;

}

}

public static void main(String[] args) throws Exception {

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

int vtces = Integer.parseInt(br.readLine());

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

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

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

}

int edges = Integer.parseInt(br.readLine());

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

String[] parts = br.readLine().split(" ");

int v1 = Integer.parseInt(parts[0]);

int v2 = Integer.parseInt(parts[1]);

int wt = Integer.parseInt(parts[2]);

graph[v1].add(new Edge(v1, v2, wt));

graph[v2].add(new Edge(v2, v1, wt));

}

int src = Integer.parseInt(br.readLine());

int dest = Integer.parseInt(br.readLine());

int criteria = Integer.parseInt(br.readLine());

int k = Integer.parseInt(br.readLine());

boolean[] visited = new boolean[vtces];

multisolver(graph, src, dest, visited, criteria, k, src + "", 0);

System.out.println("Smallest Path = " + spath + "@" + spathwt);

System.out.println("Largest Path = " + lpath + "@" + lpathwt);

System.out.println("Just Larger Path than " + criteria + " = " + cpath + "@" + cpathwt);

System.out.println("Just Smaller Path than " + criteria + " = " + fpath + "@" + fpathwt);

System.out.println(k + "th largest path = " + pq.peek().psf + "@" + pq.peek().wsf);

}

static String spath;

static Integer spathwt = Integer.MAX\_VALUE;

static String lpath;

static Integer lpathwt = Integer.MIN\_VALUE;

static String cpath;

static Integer cpathwt = Integer.MAX\_VALUE;

static String fpath;

static Integer fpathwt = Integer.MIN\_VALUE;

static PriorityQueue<Pair> pq = new PriorityQueue<>();

public static void multisolver(ArrayList<Edge>[] graph, int src, int dest, boolean[] visited, int criteria, int k, String psf, int wsf) {

if (src == dest) {

if (wsf < spathwt) {

spathwt = wsf;

spath = psf;

}

if (wsf > lpathwt) {

lpathwt = wsf;

lpath = psf;

}

if (wsf > criteria && wsf < cpathwt) {

cpathwt = wsf;

cpath = psf;

}

if (wsf < criteria && wsf > fpathwt) {

fpathwt = wsf;

fpath = psf;

}

if (pq.size() < k) {

pq.add(new Pair(wsf, psf));

} else {

if (wsf > pq.peek().wsf) {

pq.remove();

pq.add(new Pair(wsf, psf));

}

}

return;

}

visited[src] = true;

for (int i = 0; i < graph[src].size(); i++) {

Edge e = graph[src].get(i);

int nbr = e.nbr;

if (visited[nbr] == false) {

multisolver(graph, nbr, dest, visited, criteria, k, psf + nbr, wsf + e.wt);

}

}

visited[src] = false;

}

}

Code : 4  
package Topic\_18\_Graphs;

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.util.ArrayList;

public class D\_GetConnectedComponentsOfGraph {

static class Edge {

int src;

int nbr;

int wt;

Edge(int src, int nbr, int wt) {

this.src = src;

this.nbr = nbr;

this.wt = wt;

}

}

public static void main(String[] args) throws Exception {

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

int vtces = Integer.parseInt(br.readLine());

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

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

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

}

int edges = Integer.parseInt(br.readLine());

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

String[] parts = br.readLine().split(" ");

int v1 = Integer.parseInt(parts[0]);

int v2 = Integer.parseInt(parts[1]);

int wt = Integer.parseInt(parts[2]);

graph[v1].add(new Edge(v1, v2, wt));

graph[v2].add(new Edge(v2, v1, wt));

}

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

boolean[] visited = new boolean[vtces];

for (int v = 0; v < vtces; v++) {

if (visited[v] == false) {

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

drawTreeAndGenerateCompo(graph, v, comp, visited);

comps.add(comp);

}

}

System.out.println(comps); //1

}

public static void drawTreeAndGenerateCompo(ArrayList<Edge>[] graph, int src, ArrayList<Integer> comp, boolean[] visited) {

visited[src] = true;

comp.add(src);

for (Edge e : graph[src]) {

if (visited[e.nbr] == false) {

drawTreeAndGenerateCompo(graph, e.nbr, comp, visited);

}

}

}

}

Code : 5  
package Topic\_18\_Graphs;

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.util.ArrayList;

public class E\_IsGraphConnected {

private static class Edge {

int src;

int nbr;

int wt;

Edge(int src, int nbr, int wt) {

this.src = src;

this.nbr = nbr;

this.wt = wt;

}

}

public static void main(String[] args) throws Exception {

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

int vtces = Integer.parseInt(br.readLine());

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

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

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

}

int edges = Integer.parseInt(br.readLine());

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

String[] parts = br.readLine().split(" ");

int v1 = Integer.parseInt(parts[0]);

int v2 = Integer.parseInt(parts[1]);

int wt = Integer.parseInt(parts[2]);

graph[v1].add(new Edge(v1, v2, wt));

graph[v2].add(new Edge(v2, v1, wt));

}

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

boolean[] visited = new boolean[vtces];

for (int v = 0; v < vtces; v++) {

if (visited[v] == false) {

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

drawTree(graph, v, comp, visited);

comps.add(comp);

}

}

System.out.println(comps.size() == 1); //1

}

public static void drawTree(ArrayList<Edge>[] graph, int src, ArrayList<Integer> comp, boolean[] visited) {

visited[src] = true;

comp.add(src);

for (Edge e : graph[src]) {

if (visited[e.nbr] == false) {

drawTree(graph, e.nbr, comp, visited);

}

}

}

}

Code : 6  
package Topic\_18\_Graphs;

import java.io.BufferedReader;

import java.io.InputStreamReader;

public class F\_NumberOfIslands {

public static void main(String[] args) throws Exception {

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

int m = Integer.parseInt(br.readLine());

int n = Integer.parseInt(br.readLine());

int[][] arr = new int[m][n];

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

String parts = br.readLine();

for (int j = 0; j < arr[0].length; j++) {

arr[i][j] = Integer.parseInt(parts.split(" ")[j]);

}

}

// write your code here

boolean[][] visited = new boolean[m][n];

int count = 0;

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

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

if (visited[i][j] == false && arr[i][j] == 0) {

count++;

fillComp(arr, visited, i, j);

}

}

}

System.out.println(count);

}

static void fillComp(int[][] arr, boolean[][] visited, int i, int j) {

visited[i][j] = true;

if (i + 1 >= 0 && i + 1 < arr.length && visited[i + 1][j] == false && arr[i + 1][j] == 0) {

fillComp(arr, visited, i + 1, j);

}

if (i - 1 >= 0 && i - 1 < arr.length && visited[i - 1][j] == false && arr[i - 1][j] == 0) {

fillComp(arr, visited, i - 1, j);

}

if (j + 1 >= 0 && j + 1 < arr[0].length && visited[i][j + 1] == false && arr[i][j + 1] == 0) {

fillComp(arr, visited, i, j + 1);

}

if (j - 1 >= 0 && j - 1 < arr[0].length && visited[i][j - 1] == false && arr[i][j - 1] == 0) {

fillComp(arr, visited, i, j - 1);

}

}

}

Code : 7  
package Topic\_18\_Graphs;

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.util.ArrayList;

public class G\_PerfectFriends {

static class Edge {

int src;

int nbr;

Edge(int src, int nbr) {

this.src = src;

this.nbr = nbr;

}

}

public static void main(String[] args) throws Exception {

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

int n = Integer.parseInt(br.readLine());

int k = Integer.parseInt(br.readLine());

// write your code here

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

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

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

}

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

String str = br.readLine();

int v1 = Integer.parseInt(str.split(" ")[0]);

int v2 = Integer.parseInt(str.split(" ")[1]);

graph[v1].add(new Edge(v1, v2));

graph[v2].add(new Edge(v2, v1));

}

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

boolean[] visited = new boolean[n];

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

if (visited[i] == false) {

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

fillComp(graph, comp, visited, i);

comps.add(comp);

}

}

int count = 0;

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

for (int j = i + 1; j < comps.size(); j++) {

int si = comps.get(i).size();

int sj = comps.get(j).size();

int pairs = si \* sj;

count += pairs;

}

}

System.out.println(count);

}

static void fillComp(ArrayList<Edge>[] graph, ArrayList<Integer> comp, boolean[] visited, int src) {

visited[src] = true;

comp.add(src);

for (int i = 0; i < graph[src].size(); i++) {

Edge e = graph[src].get(i);

if (visited[e.nbr] == false) {

fillComp(graph, comp, visited, e.nbr);

}

}

}

}

Code : 8  
package Topic\_18\_Graphs;

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.util.ArrayList;

public class H\_HamiltonianPathAndCycle {

static class Edge {

int src;

int nbr;

int wt;

Edge(int src, int nbr, int wt) {

this.src = src;

this.nbr = nbr;

this.wt = wt;

}

}

public static void main(String[] args) throws Exception {

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

int vtces = Integer.parseInt(br.readLine());

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

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

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

}

int edges = Integer.parseInt(br.readLine());

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

String[] parts = br.readLine().split(" ");

int v1 = Integer.parseInt(parts[0]);

int v2 = Integer.parseInt(parts[1]);

int wt = Integer.parseInt(parts[2]);

graph[v1].add(new Edge(v1, v2, wt));

graph[v2].add(new Edge(v2, v1, wt));

}

int src = Integer.parseInt(br.readLine());

boolean[] visited = new boolean[vtces];

getHamilTonian(src, graph, 1, visited, src, src + "", vtces);

// write all your codes here

}

public static void getHamilTonian(int src, ArrayList<Edge> graph[], int vsf, boolean[] visited, int curr, String psf, int vtces) {

if (vsf == vtces) {

boolean hamiltonian = false;

for (Edge e : graph[curr]) {

if (e.nbr == src) {

hamiltonian = true;

System.out.println(psf + "\*");

break;

}

}

if (hamiltonian == false) {

System.out.println(psf + ".");

}

return;

}

visited[curr] = true;

for (Edge e : graph[curr]) {

if (visited[e.nbr] == false) {

getHamilTonian(src, graph, vsf + 1, visited, e.nbr, psf + e.nbr, vtces);

}

}

visited[curr] = false;

}

}

Code : 9  
package Topic\_18\_Graphs;

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.util.ArrayDeque;

import java.util.ArrayList;

public class J\_BFS {

static class Edge {

int src;

int nbr;

Edge(int src, int nbr) {

this.src = src;

this.nbr = nbr;

}

}

static class Pair {

int vtx;

String psf;

Pair(int vtx, String psf) {

this.vtx = vtx;

this.psf = psf;

}

}

public static void main(String[] args) throws Exception {

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

int vtces = Integer.parseInt(br.readLine());

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

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

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

}

int edges = Integer.parseInt(br.readLine());

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

String[] parts = br.readLine().split(" ");

int v1 = Integer.parseInt(parts[0]);

int v2 = Integer.parseInt(parts[1]);

graph[v1].add(new Edge(v1, v2));

graph[v2].add(new Edge(v2, v1));

}

int src = Integer.parseInt(br.readLine());

// write your code here

ArrayDeque<Pair> queue = new ArrayDeque<>();

queue.add(new Pair(src, src + ""));

boolean[] visited = new boolean[vtces];

while (queue.size() > 0) {

// remove, mark\*, work, add\*

Pair rem = queue.remove();

if (visited[rem.vtx] == true) {

continue;

}

visited[rem.vtx] = true;

System.out.println(rem.vtx + "@" + rem.psf);

for (int i = 0; i < graph[rem.vtx].size(); i++) {

Edge e = graph[rem.vtx].get(i);

if (visited[e.nbr] == false) {

queue.add(new Pair(e.nbr, rem.psf + e.nbr));

}

}

}

}

}

Code : 10  
package Topic\_18\_Graphs;

import java.util.Scanner;

public class J\_Knightstour {

public static void main(String[] args) throws Exception {

Scanner s = new Scanner(System.in);

int n = s.nextInt();

int r = s.nextInt();

int c = s.nextInt();

int[][] chess = new int[n][n];

printKnightsTour(chess, r, c, 1);

}

private static void printKnightsTour(int[][] chess, int r, int c, int move) {

// TODO Auto-generated method stub

if (r < 0 || c < 0 || r >= chess.length || c >= chess.length || chess[r][c] > 0) {

return;

} else if (move == chess.length \* chess.length) {

chess[r][c] = move;

displayBoard(chess);

chess[r][c] = 0;

return;

}

chess[r][c] = move;

printKnightsTour(chess, r - 2, c + 1, move + 1);

printKnightsTour(chess, r - 1, c + 2, move + 1);

printKnightsTour(chess, r + 1, c + 2, move + 1);

printKnightsTour(chess, r + 2, c + 1, move + 1);

printKnightsTour(chess, r + 2, c - 1, move + 1);

printKnightsTour(chess, r + 1, c - 2, move + 1);

printKnightsTour(chess, r - 1, c - 2, move + 1);

printKnightsTour(chess, r - 2, c - 1, move + 1);

chess[r][c] = 0;

}

private static void displayBoard(int[][] a) {

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

for (int j = 0; j < a[0].length; j++) {

System.out.print(a[i][j] + " ");

}

System.out.println();

}

System.out.println();

}

}

Code : 11  
package Topic\_18\_Graphs;

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.util.ArrayDeque;

import java.util.ArrayList;

public class K\_IsGraphCyclic {

static class Edge {

int src;

int nbr;

int wt;

Edge(int src, int nbr, int wt) {

this.src = src;

this.nbr = nbr;

this.wt = wt;

}

}

static class Pair {

int vtx;

String psf;

Pair(int vtx, String psf) {

this.vtx = vtx;

this.psf = psf;

}

}

public static void main(String[] args) throws Exception {

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

int vtces = Integer.parseInt(br.readLine());

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

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

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

}

int edges = Integer.parseInt(br.readLine());

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

String[] parts = br.readLine().split(" ");

int v1 = Integer.parseInt(parts[0]);

int v2 = Integer.parseInt(parts[1]);

int wt = Integer.parseInt(parts[2]);

graph[v1].add(new Edge(v1, v2, wt));

graph[v2].add(new Edge(v2, v1, wt));

}

// write your code here

boolean[] visited = new boolean[vtces];

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

if (visited[i] == false) {

boolean isCyclic = isCyclic(graph, visited, i);

if (isCyclic == true) {

System.out.println(true);

return;

}

}

}

System.out.println(false);

}

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

ArrayDeque<Pair> queue = new ArrayDeque<>();

queue.add(new Pair(src, src + ""));

while (queue.size() > 0) {

// remove, mark\*, work, add\*

Pair rem = queue.remove();

if (visited[rem.vtx] == true) {

return true;

}

visited[rem.vtx] = true;

// System.out.println(rem.vtx + "@" + rem.psf);

for (int i = 0; i < graph[rem.vtx].size(); i++) {

Edge e = graph[rem.vtx].get(i);

if (visited[e.nbr] == false) {

queue.add(new Pair(e.nbr, rem.psf + e.nbr));

}

}

}

return false;

}

}

Code : 12  
package Topic\_18\_Graphs;

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.util.ArrayDeque;

import java.util.ArrayList;

public class L\_IsGraphBipartite {

static class Edge {

int src;

int nbr;

Edge(int src, int nbr) {

this.src = src;

this.nbr = nbr;

}

}

public static void main(String[] args) throws Exception {

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

int vtces = Integer.parseInt(br.readLine());

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

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

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

}

int edges = Integer.parseInt(br.readLine());

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

String[] parts = br.readLine().split(" ");

int v1 = Integer.parseInt(parts[0]);

int v2 = Integer.parseInt(parts[1]);

graph[v1].add(new Edge(v1, v2));

graph[v2].add(new Edge(v2, v1));

}

Integer[] visited = new Integer[vtces];

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

if (visited[i] == null) {

boolean isBip = IsCompBipartite(graph, visited, i);

if (isBip == false) {

System.out.println(false);

return;

}

}

}

System.out.println(true);

}

static class Pair {

int vtx;

int level;

Pair(int vtx, int level) {

this.vtx = vtx;

this.level = level;

}

}

public static boolean IsCompBipartite(ArrayList<Edge>[] graph, Integer[] visited, int src) {

ArrayDeque<Pair> queue = new ArrayDeque<>();

queue.add(new Pair(src, 1));

while (queue.size() > 0) {

Pair rem = queue.remove();

if (visited[rem.vtx] != null) {

int originalValue = visited[rem.vtx];

int newValue = rem.level % 2;

if (originalValue != newValue) {

return false;

}

}

visited[rem.vtx] = rem.level % 2; // 0 for even, 1 for odd, null if unvisited

for (Edge e : graph[rem.vtx]) {

if (visited[e.nbr] == null) {

queue.add(new Pair(e.nbr, rem.level + 1));

}

}

}

return true;

}

}

Code : 13  
package Topic\_18\_Graphs;

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.util.ArrayDeque;

import java.util.ArrayList;

public class M\_SpreadOfInfection {

static class Edge {

int src;

int nbr;

Edge(int src, int nbr) {

this.src = src;

this.nbr = nbr;

}

}

public static void main(String[] args) throws Exception {

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

int vtces = Integer.parseInt(br.readLine());

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

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

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

}

int edges = Integer.parseInt(br.readLine());

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

String[] parts = br.readLine().split(" ");

int v1 = Integer.parseInt(parts[0]);

int v2 = Integer.parseInt(parts[1]);

graph[v1].add(new Edge(v1, v2));

graph[v2].add(new Edge(v2, v1));

}

int src = Integer.parseInt(br.readLine());

int t = Integer.parseInt(br.readLine());

int count = 0;

ArrayDeque<Pair> queue = new ArrayDeque<>();

queue.add(new Pair(src, 1));

int[] visited = new int[vtces];

while (queue.size() > 0) {

Pair rem = queue.remove();

if (visited[rem.v] > 0) {

continue;

}

visited[rem.v] = rem.time;

if (rem.time > t) {

break;

} else {

count++;

}

for (Edge e : graph[rem.v]) {

if (visited[e.nbr] == 0) {

queue.add(new Pair(e.nbr, rem.time + 1));

}

}

}

System.out.println(count);

}

static class Pair {

int v;

int time;

Pair(int v, int time) {

this.v = v;

this.time = time;

}

}

}

Code : 14  
package Topic\_18\_Graphs;

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.util.ArrayList;

import java.util.PriorityQueue;

public class N\_shortestpathinweights {

static class Edge {

int src;

int nbr;

int wt;

Edge(int src, int nbr, int wt) {

this.src = src;

this.nbr = nbr;

this.wt = wt;

}

}

static class Pair implements Comparable<Pair> {

int vtx;

String psf;

int wsf;

Pair(int vtx, String psf, int wsf) {

this.vtx = vtx;

this.psf = psf;

this.wsf = wsf;

}

public int compareTo(Pair o) {

return this.wsf - o.wsf;

}

}

public static void main(String[] args) throws Exception {

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

int vtces = Integer.parseInt(br.readLine());

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

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

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

}

int edges = Integer.parseInt(br.readLine());

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

String[] parts = br.readLine().split(" ");

int v1 = Integer.parseInt(parts[0]);

int v2 = Integer.parseInt(parts[1]);

int wt = Integer.parseInt(parts[2]);

graph[v1].add(new Edge(v1, v2, wt));

graph[v2].add(new Edge(v2, v1, wt));

}

int src = Integer.parseInt(br.readLine());

// write your code here

PriorityQueue<Pair> pq = new PriorityQueue<>();

pq.add(new Pair(src, src + "", 0));

boolean[] visited = new boolean[vtces];

while (pq.size() > 0) {

Pair rem = pq.remove();

if (visited[rem.vtx] == true) {

continue;

}

visited[rem.vtx] = true;

System.out.println(rem.vtx + " via " + rem.psf + " @ " + rem.wsf);

for (Edge e : graph[rem.vtx]) {

if (visited[e.nbr] == false) {

pq.add(new Pair(e.nbr, rem.psf + e.nbr, rem.wsf + e.wt));

}

}

}

}

}

Code : 15  
package Topic\_18\_Graphs;

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.util.ArrayList;

import java.util.PriorityQueue;

public class O\_minimumwirerequiredtoconnectallpcs {

static class Edge {

int src;

int nbr;

int wt;

Edge(int src, int nbr, int wt) {

this.src = src;

this.nbr = nbr;

this.wt = wt;

}

}

static class Pair implements Comparable<Pair> {

int vtx;

int par;

int cost;

Pair(int vtx, int par, int cost) {

this.vtx = vtx;

this.par = par;

this.cost = cost;

}

public int compareTo(Pair o) {

return this.cost - o.cost;

}

}

public static void main(String[] args) throws Exception {

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

int vtces = Integer.parseInt(br.readLine());

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

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

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

}

int edges = Integer.parseInt(br.readLine());

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

String[] parts = br.readLine().split(" ");

int v1 = Integer.parseInt(parts[0]);

int v2 = Integer.parseInt(parts[1]);

int wt = Integer.parseInt(parts[2]);

graph[v1].add(new Edge(v1, v2, wt));

graph[v2].add(new Edge(v2, v1, wt));

}

// write your code here

PriorityQueue<Pair> pq = new PriorityQueue<>();

pq.add(new Pair(0, -1, 0));

boolean[] visited = new boolean[vtces];

while (pq.size() > 0) {

Pair rem = pq.remove();

if (visited[rem.vtx] == true) {

continue;

}

visited[rem.vtx] = true;

if (rem.par != -1) {

System.out.println("[" + rem.vtx + "-" + rem.par + "@" + rem.cost + "]");

}

for (Edge e : graph[rem.vtx]) {

if (visited[e.nbr] == false) {

pq.add(new Pair(e.nbr, rem.vtx, e.wt));

}

}

}

}

}

Code : 16  
package Topic\_18\_Graphs;

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.util.ArrayList;

import java.util.Stack;

public class P\_orderofcompilation {

static class Edge {

int src;

int nbr;

Edge(int src, int nbr) {

this.src = src;

this.nbr = nbr;

}

}

public static void main(String[] args) throws Exception {

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

int vtces = Integer.parseInt(br.readLine());

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

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

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

}

int edges = Integer.parseInt(br.readLine());

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

String[] parts = br.readLine().split(" ");

int v1 = Integer.parseInt(parts[0]);

int v2 = Integer.parseInt(parts[1]);

graph[v1].add(new Edge(v1, v2));

}

boolean[] visited = new boolean[vtces];

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

// write your code here

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

if (visited[i] == false) {

topologicalsort(graph, visited, i, st);

}

}

while (st.size() > 0) {

System.out.println(st.pop());

}

}

static void topologicalsort(ArrayList<Edge>[] graph, boolean[] visited, int src, Stack<Integer> st) {

visited[src] = true;

for (Edge e : graph[src]) {

if (visited[e.nbr] == false) {

topologicalsort(graph, visited, e.nbr, st);

}

}

st.push(src);

}

}

Code : 17  
package Topic\_18\_Graphs;

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.util.ArrayList;

import java.util.Stack;

public class Q\_iterativedepthfirsttraversal {

static class Edge {

int src;

int nbr;

Edge(int src, int nbr) {

this.src = src;

this.nbr = nbr;

}

}

static class Pair {

int vtx;

String psf;

Pair(int vtx, String psf) {

this.vtx = vtx;

this.psf = psf;

}

}

public static void main(String[] args) throws Exception {

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

int vtces = Integer.parseInt(br.readLine());

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

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

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

}

int edges = Integer.parseInt(br.readLine());

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

String[] parts = br.readLine().split(" ");

int v1 = Integer.parseInt(parts[0]);

int v2 = Integer.parseInt(parts[1]);

graph[v1].add(new Edge(v1, v2));

graph[v2].add(new Edge(v2, v1));

}

int src = Integer.parseInt(br.readLine());

boolean[] visited = new boolean[vtces];

// write your code here

Stack<Pair> st = new Stack<>();

st.push(new Pair(src, src + ""));

while (st.size() > 0) {

Pair rem = st.pop();

if (visited[rem.vtx] == true) {

continue;

}

visited[rem.vtx] = true;

System.out.println(rem.vtx + "@" + rem.psf);

for (Edge e : graph[rem.vtx]) {

if (visited[e.nbr] == false) {

st.push(new Pair(e.nbr, rem.psf + e.nbr));

}

}

}

}

}