/\* Amir Yamini

\* Professor Schwartz

\* 12-07-2017

\* Project 3 - Graph

\*/

import javax.xml.transform.Source;

import java.util.\*;

import java.io.File;

import java.io.FileReader;

import java.io.IOException;

import java.io.FileNotFoundException;

public class Graph{

//------------------------------------------------------

private ArrayList<EdgeNode>[] adjList;

private int nVertices;

private int nEdges;

private String fileName;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Constructor\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public Graph ( String inputFileName) {

// TODO Auto-generated method stub

try {

// Open the file to read

Scanner in = new Scanner(new File(inputFileName));

List<String> fields = new ArrayList<String>();

//int thePackage = (fields.size() - 1)/3;

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

int vicCount = 0;

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

int vecCount = 0;

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

int jenCount = 0;

int counter = 1;

int veryImportantNumber = 0;

int fieldSpell = 1;

while (in.hasNext()) {

fields.add(in.nextInt() +"");

}

//System.out.print(fields.toString());

while (fieldSpell < fields.size()) {

veryImportantNumber = Integer.parseInt(fields.get(0));

if (counter == 1) {

victor.add(Integer.parseInt(fields.get(fieldSpell)));

vicCount++;

}

if (counter == 2) {

vector.add(Integer.parseInt(fields.get(fieldSpell)));

vecCount++;

}

if (counter == 3) {

jenny.add(Integer.parseInt(fields.get(fieldSpell)));

jenCount++;

counter = 0;

}

counter++;

fieldSpell++;

}

System.out.println();

// for(int v = 0; v < victor.size(); v++){

// System.out.print(" " + victor.get(v));

// }

// System.out.println();

// for(int r = 0; r < victor.size(); r++){

// System.out.print(" " + vector.get(r));

// }

// System.out.println();

// for(int j = 0; j < jenny.size(); j++){

// System.out.print(" " + jenny.get(j));

// }

//System.out.println(veryImportantNumber);

int variable = 0;

adjList = (ArrayList<EdgeNode>[]) new ArrayList[veryImportantNumber];

nVertices = veryImportantNumber;

for(int a = 0; a < nVertices; a++) {

adjList[a] = new ArrayList();

}

for(int m = 0; m < jenny.size(); m++) {

variable = victor.get(m);

nEdges++;

adjList[variable].add(new EdgeNode(variable, vector.get(m), jenny.get(m)));

}

System.out.println();

// for (int y = 0; y < adjList.length ; y++) {

// System.out.print("v= " + y + " ");

// System.out.println(adjList[y].toString());

// }

}

catch(FileNotFoundException ex){

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Print graph method\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public void printGraph()

{

System.out.println("Graph: nVertices: " + nVertices + " nEdges: " + nEdges);

for(int p = 0; p < adjList.length; p++)

{

System.out.println("v= " + p + " " + adjList[p].toString());

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* BFS Shortest paths \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public SPPacket bfsShortestPaths (int start) {

int [] distance = new int[nVertices];

int [] daddy = new int [nVertices];

Queue<Integer> mainQueue = new PriorityQueue<>(nVertices);

boolean[] detective = new boolean[nVertices];

for(int d = 0; d < nVertices; d++){

distance[d] = Integer.MAX\_VALUE;

daddy[d] = -1;

}

int current = start;

mainQueue.add(start);

//daddy[start] = -1;

detective[start] = true;

distance[start] = 0;

while(!mainQueue.isEmpty()){

if(detective[current]){

current = mainQueue.remove();

}

detective[current] = true;

for(int s = 0; s < adjList[current].size(); s++){

if(!detective[adjList[current].get(s).vertex2]){

mainQueue.add(adjList[current].get(s).vertex2);

distance[adjList[current].get(s).vertex2] = distance[adjList[current].get(s).vertex1] + 1;

daddy[adjList[current].get(s).vertex2] = adjList[current].get(s).vertex1;

}

}

}

// for (int p = 0; p < daddy.length; p++){

// System.out.println(daddy[p]);

// }

return new SPPacket(start, distance, daddy);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Dijkstra's Shortest Path Algorithm\*\*\* \*/

public SPPacket dijkstraShortestPaths (int start ) {

int [] distance = new int[nVertices];

int [] daddy = new int [nVertices];

Queue<Integer> mainQueue = new PriorityQueue<>(nVertices);

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

distance[v] = Integer.MAX\_VALUE;

daddy[v] = -1;

mainQueue.add(0);

}

distance[start] = 0;

int count = start;

mainQueue.add(start);

while(!mainQueue.isEmpty()){

mainQueue.remove();

for(int s = 0; s < adjList[count].size(); s++){

if(distance[adjList[count].get(s).vertex1] + adjList[count].get(s).weight < distance[adjList[count].get(s).vertex2]){

mainQueue.add(adjList[count].get(s).vertex2);

distance[adjList[count].get(s).vertex2] = distance[adjList[count].get(s).vertex1] + adjList[count].get(s).weight;

daddy[adjList[count].get(s).vertex2] = adjList[count].get(s).vertex1;

count = adjList[count].get(s).vertex1;

}else {

count = adjList[count].get(s).vertex2;

}

}

}

// for (int p = 0; p < daddy.length; p++){

// System.out.println(daddy[p]);

// }

// for (int p = 0; p < daddy.length; p++){

// System.out.println("distance " + distance[p]);

// }

return new SPPacket(start, distance, daddy);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Bellman Ford Shortest Paths \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public SPPacket bellmanFordShortestPaths(int start) {

int [] distance = new int[nVertices];

int [] distance2 = new int[nVertices];

int [] daddy = new int [nVertices];

ArrayList<EdgeNode> maEdges = new ArrayList<>();

//initialize

for(int d = 0; d < nVertices; d++){

distance[d] = Integer.MAX\_VALUE;

distance2[d] = Integer.MAX\_VALUE;

daddy[d] = -1;

}

//initialize

for(int x = 0; x < nVertices; x++){

for(int y = 0; y < adjList[x].size(); y++){

maEdges.add(adjList[x].get(y));

}

}

//System.out.println(maEdges.toString());

int current = start;

distance[start] = 0;

distance2[start] = 0;

for(int v = 0; v < nVertices-1; v++) {

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

if (distance[maEdges.get(e).vertex1] + maEdges.get(e).weight < distance[maEdges.get(e).vertex2]) {

if (distance[maEdges.get(e).vertex1] != Integer.MAX\_VALUE) {

distance[maEdges.get(e).vertex2] = distance[maEdges.get(e).vertex1] + maEdges.get(e).weight;

//dummy array filler for comparison later

distance2[maEdges.get(e).vertex2] = distance2[maEdges.get(e).vertex1] + maEdges.get(e).weight;

daddy[maEdges.get(e).vertex2] = maEdges.get(e).vertex1;

}

}

}

}

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

if (distance[maEdges.get(e).vertex1] + maEdges.get(e).weight < distance[maEdges.get(e).vertex2]){

if (distance[maEdges.get(e).vertex1] != Integer.MAX\_VALUE) {

distance[maEdges.get(e).vertex2] = distance[maEdges.get(e).vertex1] + maEdges.get(e).weight;

daddy[maEdges.get(e).vertex2] = maEdges.get(e).vertex1;

}

}

}

//comparison for negative cycle

int p = 0;

boolean cycle = true;

while(p < distance.length && cycle == true){

if(distance[p] != distance2[p]){

cycle = false;

}

p++;

}

if(cycle == false){

return null;

}

return new SPPacket(start, distance, daddy);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Prints shortest paths\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public void printShortestPaths( SPPacket spp) {

//Find the path of parents

System.out.println(spp);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*isStronglyConnected\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public boolean isStronglyConnected()

{

SPPacket[] whereIsDaddy = new SPPacket[nVertices];

for(int a = 0; a < nVertices; a++) {

whereIsDaddy[a] = bfsShortestPaths(a);

for(int y = 0; y < whereIsDaddy[a].parent.length; y++) {

if(whereIsDaddy[a].parent[y] == -1 && a != y) {

return false;

}

}

}

return true;

}

}//end Graph class

//place the EdgeNode class and the SPPacket class inside the Graph.java file

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class EdgeNode

{

int vertex1;

int vertex2;

int weight;

public EdgeNode ( int v1, int v2, int w)

{

vertex1 = v1;

vertex2 = v2;

weight = w;

}

public String toString() {

return "(" + vertex1 + "," + vertex2 + "," + weight + ")";

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class SPPacket

{

int[] d; //distance array

int[] parent; //parent path array

int source; //source vertex

public SPPacket( int start, int[] dist, int[] pp) {

source = start;

d = dist;

parent = pp;

}

public int[] getDistance() {

return d;

}

public int[] getParent() {

return parent;

}

public int getSource() {

return source;

}

public String toString() {

String whereIsDaddy = "";

whereIsDaddy += ("Shortest Paths from vertex " + source + " to vertex\n");

for(int x = 0; x < parent.length; x++) {

whereIsDaddy += (x + ": [");

String clues = "";

int childServices = x;

if(childServices != source) {

while(parent[childServices] != -1) {

clues = parent[childServices] + "," + clues;

childServices = parent[childServices];

}

}

whereIsDaddy = whereIsDaddy + clues;

whereIsDaddy = whereIsDaddy + x;

whereIsDaddy += ("] Path Weight = " + d[x] + "\n");

}

return whereIsDaddy;

}

}