Assignment 03

Name: MD. Shahriar Mim. ID: 17101463

*My Code automatically detects the source nodes and sink nodes from the given input file on the basis of incoming and outgoing edges of a node/vertex. Also takes inputs from a .txt file. The values for inputs are given too.*

Java Code:

import java.io.\*;  
import java.util.\*;  
public class Assignment03{  
 static ArrayList<Integer> *source* = new ArrayList<Integer>();  
   
 static ArrayList<Integer> *sink* = new ArrayList<Integer>();  
   
 static LinkedList< LinkedList<Integer> > *mainList*;  
 static LinkedList< LinkedList<Double> > *disList*;  
 static int[][] *graph*;  
   
 public static void main (String[] args) throws java.lang.Exception  
 {  
 File f = new File("input1.txt");  
 Scanner sc = new Scanner (f);  
 int n = sc.nextInt();  
 n++ ;  
 int m = sc.nextInt();  
 int[] incm = new int[n];  
 int[] outcm = new int[n];  
 *mainList* = new LinkedList< LinkedList<Integer> >();  
 *disList* = new LinkedList< LinkedList<Double> >();  
 for (int i =0; i<n; i++){  
 LinkedList<Integer> dummy = new LinkedList<Integer>();  
 LinkedList<Double> dummy1 = new LinkedList<Double>();  
 *mainList*.add(dummy);  
 *disList*.add(dummy1);  
   
 }  
   
 for (int i =0 ; i<m; i++){  
   
 int v = sc.nextInt();  
 int q = sc.nextInt();  
 double w = sc.nextDouble();  
 *mainList*.get(v).add(q);  
 outcm[v]=outcm[v]+1;  
 incm[q]=incm[q]+1;  
 *disList*.get(v).add(w);  
 }  
 System.*out*.println(*mainList*);  
 System.*out*.println(*disList*);  
   
 for (int i =1; i<n;i++){  
 if (outcm[i]==0){  
 *sink*.add(i);  
   
 }  
 }  
   
 for (int i =1; i<n;i++){  
 if (incm[i]==0){  
 *source*.add(i);  
   
 }  
 }  
 System.*err*.println("source = "+*source*+" "+"Sink = "+*sink*);  
   
 *task01*();  
 System.*err*.println("++++++++++++TASK01 CHECKED++++++++++ \n");  
 *task02*();  
 System.*err*.println("++++++++++++TASK02 CHECKED++++++++++ \n");  
 // fordfulkerson  
 n = *mainList*.size();  
 *graph* = new int[n][n];  
 for(int i =0; i<*mainList*.size();i++){  
 for(int j = 0 ; j<*mainList*.get(i).size();j++){  
 int x = *mainList*.get(i).get(j);  
 double val = *disList*.get(i).get(j);  
 int v = (int) val;  
 *graph*[i][x]=v;  
 }  
 }  
// for(int i =0; i<graph.length;i++){  
// for(int j = 0 ; j<graph[i].length;j++){  
// System.out.print(graph[i][j]+" ");  
// }  
// System.out.println();  
// }  
   
 int s = *source*.get(0);  
 int t = *sink*.get(0);  
 System.*out*.println("The max flow is " +*fordFulkerson*());  
 //min Cut  
   
 *minCut*();  
 }  
 public static void task01(){  
 // System.err.println(mainList.size());  
 if(*source*.size()>1){  
   
 int superSource = *mainList*.size();  
 LinkedList<Integer> dummy = new LinkedList<Integer>();  
 LinkedList<Double> dummy1 = new LinkedList<Double>();  
 *mainList*.add(dummy);  
 *disList*.add(dummy1);  
 // System.err.println(mainList.size());   
 for (int i = 0; i<*source*.size(); i++){  
 *mainList*.get(superSource).add(*source*.get(i));  
   
 *disList*.get(superSource).add(Double.*MAX\_VALUE*);  
 }  
 *source*.clear();  
 *source*.add(superSource);  
 }  
 if(*sink*.size()>1){  
 int superSink = *mainList*.size();  
 LinkedList<Integer> dummy = new LinkedList<Integer>();  
 *mainList*.add(dummy);  
 LinkedList<Double> dummy1 = new LinkedList<Double>();  
 *disList*.add(dummy1);  
 for (int i = 0; i<*sink*.size(); i++){  
 *mainList*.get(*sink*.get(i)).add(superSink);  
 *disList*.get(*sink*.get(i)).add(Double.*MAX\_VALUE*);  
 }  
 *sink*.clear();  
 *sink*.add(superSink);  
 }  
 System.*out*.println("New Graph is : ");  
 System.*out*.println(*mainList*);  
 System.*out*.println(*disList*);  
 System.*err*.println("New source = "+*source*+" "+" New Sink = "+*sink*);  
 }  
 public static void task02(){  
 for (int i = 0; i<*mainList*.size();i++){  
 LinkedList<Integer> li = *mainList*.get(i);  
 for (int j = 0; j<li.size();j++){  
 int x = li.get(j);  
 LinkedList<Integer> li2 = *mainList*.get(x);  
 if(li2.contains(i)){  
 int indx = li2.indexOf(i);  
 LinkedList<Integer> dummyVert = new LinkedList<Integer>();  
 *mainList*.add(dummyVert);  
 *mainList*.get(*mainList*.size()-1).add(i);  
 li2.remove(indx);  
 LinkedList<Double> dummyWeight = new LinkedList<Double>();  
 *disList*.add(dummyWeight);  
 double val = *disList*.get(x).get(indx);  
 *disList*.get(x).remove(indx);  
 *disList*.get(*disList*.size()-1).add(val);  
   
 }  
 }  
 }  
 System.*out*.println("New Graph is : ");  
 System.*out*.println(*mainList*);  
 System.*out*.println(*disList*);  
 }//task02  
   
 static int fordFulkerson()  
 {  
 int x;  
 int y;  
 int s = *source*.get(0);  
 int t = *sink*.get(0);  
 int residualGraph[][] = new int[*mainList*.size()][*mainList*.size()];  
   
 for (x = 0; x < *mainList*.size(); x++)  
 for (y = 0; y < *mainList*.size(); y++)  
 residualGraph[x][y] = *graph*[x][y];  
   
   
 int parent[] = new int[*mainList*.size()];  
   
 int max\_flow = 0;  
   
   
 while (*bfs*(residualGraph,parent)) {  
   
   
 int path\_flow = Integer.*MAX\_VALUE*;  
 for (y = t; y != s; y = parent[y]) {  
 x = parent[y];  
 path\_flow = Math.*min*(path\_flow, residualGraph[x][y]);  
 }  
   
 for (y = t; y != s; y = parent[y]) {  
 x = parent[y];  
 residualGraph[x][y] -= path\_flow;  
 residualGraph[y][x] += path\_flow;  
 }  
   
   
 max\_flow += path\_flow;  
 }  
   
   
 return max\_flow;  
 }  
 static boolean bfs(int residualGraph[][], int parent[])  
 {  
 int s = *source*.get(0);  
 int t = *sink*.get(0);  
   
 boolean visited[] = new boolean[*mainList*.size()];  
 for (int i = 0; i < *mainList*.size(); ++i)  
 visited[i] = false;  
   
   
 LinkedList<Integer> queue  
 = new LinkedList<Integer>();  
 queue.add(s);  
 visited[s] = true;  
 parent[s] = -1;  
   
   
 while (queue.size() != 0) {  
 int x = queue.poll();  
   
 for (int y = 0; y < *mainList*.size(); y++) {  
 if (visited[y] == false  
 && residualGraph[x][y] > 0) {  
 if (y == t) {  
 parent[y] = x;  
 return true;  
 }  
 queue.add(y);  
 parent[y] = x;  
 visited[y] = true;  
 }  
 }  
 }  
   
   
 return false;  
 }  
 private static void dfs(int[][] residualGraph, int s,  
 boolean[] visited) {  
 visited[s] = true;  
 for (int i = 0; i < residualGraph.length; i++) {  
 if (residualGraph[s][i] > 0 && !visited[i]) {  
 *dfs*(residualGraph, i, visited);  
 }  
 }  
 }  
 private static void minCut() {  
 int x,y;  
 int s = *source*.get(0);  
 int t = *sink*.get(0);  
 int[][] residualGraph = new int[*mainList*.size()][*mainList*.size()];  
 for (int i = 0; i < *graph*.length; i++) {  
 for (int j = 0; j < *graph*.length; j++) {  
 residualGraph[i][j] = *graph*[i][j];  
 }  
 }  
   
   
 int[] parent = new int[*graph*.length];  
   
   
 while (*bfs*(residualGraph,parent)) {  
   
   
 int pathFlow = Integer.*MAX\_VALUE*;  
 for (y = t; y != s; y = parent[y]) {  
 x = parent[y];  
 pathFlow = Math.*min*(pathFlow, residualGraph[x][y]);  
 }  
   
   
 for (y = t; y != s; y = parent[y]) {  
 x = parent[y];  
 residualGraph[x][y] = residualGraph[x][y] - pathFlow;  
 residualGraph[y][x] = residualGraph[y][x] + pathFlow;  
 }  
 }  
   
   
 boolean[] isVisited = new boolean[*graph*.length];  
 *dfs*(residualGraph, s, isVisited);  
   
 System.*out*.println("The min cut edges are ");  
 for (int i = 0; i < *graph*.length; i++) {  
 for (int j = 0; j < *graph*.length; j++) {  
 if (*graph*[i][j] > 0 && isVisited[i] && !isVisited[j]) {  
 System.*out*.println(i + " - " + j);  
 }  
 }  
 }  
 }  
   
}

Input File Values:

10 16  
1 2 6  
1 3 6  
2 4 4  
2 5 2   
3 2 5  
2 3 10  
3 5 9  
4 6 4  
4 7 7  
5 4 8  
5 7 7  
6 8 7  
7 8 4  
7 6 11  
7 9 3  
10 2 3

*Please go to next page*

OUTPUT:

