TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**Bài Tập Lớn Môn Toán Tổ Hợp và Đồ Thị**

**Combinatorial Optimization**

*Người hướng dẫn*: **LÊ ĐÌNH THẬN**

*Người thực hiện*: **NGUYỄN LÊ THIỆN ĐĂNG – 51503045**

**ĐINH CAO KHẢI – 51503318**

**NGÔ MẠNH HOÀNG– 51503356**

Lớp **: 15050303**

Khoá  **: 19**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2019**

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**Bài Tập Lớn Môn Toán Tổ Hợp và Đồ Thị**

**Combinatorial Optimization**

*Người hướng dẫn*: **LÊ ĐÌNH THẬN**

*Người thực hiện*: **NGUYỄN LÊ THIỆN ĐĂNG – 51503045**

**ĐINH CAO KHẢI – 51503318**

**NGÔ MẠNH HOÀNG– 51503356**

Lớp **: 15050303**

Khoá  **: 19**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2019**

LỜI CẢM ƠN

Được sự phân công của khoa Công nghệ thông tin Trường Đại học Tôn Đức Thắng và sự hướng dẫn của thầy LÊ ĐÌNH THẬN, chúng tôi đã thực hiện thiết kế trang web nghe nhạc.

Để hoàn thành báo cáo này, chúng tôi xin chân thành cảm ơn Khoa đã tạo cơ hội. Xin chân thành cảm ơn các thầy cô đã tận tình hướng dẫn, giảng dạy trong suốt quá trình học tập và rèn luyện. Mặc dù đã có nhiều cố gắng và nỗ lực để hoàn thành, nhưng chúng tôi vẫn có những hạn chế về kinh nghiệm và thiếu sót mà bản thân nhận ra được. Chúng tôi rất mong sự góp ý chân thành của thầy cô để có thể rút kinh nghiệm cũng như hoàn thiện báo cáo một cách tốt nhất.

Chúng tôi xin chân thành cảm ơn!.

**ĐỒ ÁN ĐƯỢC HOÀN THÀNH**

**TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

Chúng tôi xin cam đoan đây là sản phẩm đồ án của riêng chúng tôi và được sự hướng dẫn của Lê Đình Thận. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong đồ án còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

**Nếu phát hiện có bất kỳ sự gian lận nào tôi xin hoàn toàn chịu trách nhiệm về nội dung đồ án của mình.** Trường đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do tôi gây ra trong quá trình thực hiện (nếu có).

*TP. Hồ Chí Minh, ngày tháng năm*

*Tác giả*

*(ký tên và ghi rõ họ tên)*

NGUYỄN LÊ THIỆN ĐĂNG

ĐINH CAO KHẢI

NGÔ MẠNH HOÀNG

PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN

**Phần xác nhận của GV hướng dẫn**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Tp. Hồ Chí Minh, ngày tháng năm

(kí và ghi họ tên)

**Phần đánh giá của GV chấm bài**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Tp. Hồ Chí Minh, ngày tháng năm

(kí và ghi họ tên)

TÓM TẮT

MỤC LỤC

[LỜI CẢM ƠN i](#_Toc8077593)

[PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN iii](#_Toc8077594)

[TÓM TẮT iv](#_Toc8077595)

[MỤC LỤC 1](#_Toc8077596)

[DANH MỤC KÍ HIỆU VÀ CHỮ VIẾT TẮT 2](#_Toc8077597)

[DANH MỤC CÁC BẢNG BIỂU, HÌNH VẼ, ĐỒ THỊ 3](#_Toc8077598)

[I. Introduction 4](#_Toc8077599)

[1. Maximum Network Flow 4](#_Toc8077600)

[2. Shorted Path 4](#_Toc8077601)

[3. Minimum Spanning Tree 4](#_Toc8077602)

[II. State of the-art 5](#_Toc8077603)

[1. Maximum Network Flow 5](#_Toc8077604)

[2. Shorted Path 6](#_Toc8077605)

[3. Minimum Spanning Tree 6](#_Toc8077606)

[III. Approaches 7](#_Toc8077607)

[1. Maximum Network Flow 7](#_Toc8077608)

[2. Shorted Path 8](#_Toc8077609)

[3. Minimum Spanning Tree 17](#_Toc8077610)

[IV. Implementation and Results 18](#_Toc8077611)

[1 Maximum Network Flow 18](#_Toc8077612)

[2 Shorted Path 20](#_Toc8077613)

[3. Minimum Spanning Tree 20](#_Toc8077614)

[TÀI LIỆU THAM KHẢO 22](#_Toc8077615)

DANH MỤC KÍ HIỆU VÀ CHỮ VIẾT TẮT

**CÁC KÝ HIỆU**

**CÁC CHỮ VIẾT TẮT**

DANH MỤC CÁC BẢNG BIỂU, HÌNH VẼ, ĐỒ THỊ

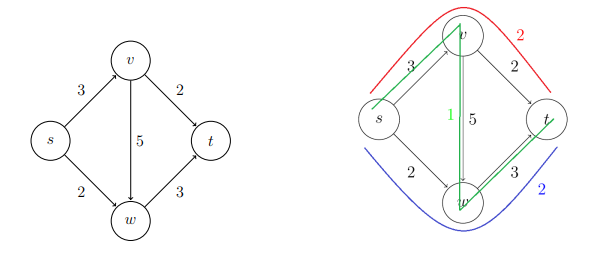
**DANH MỤC HÌNH**

**DANH MỤC BẢNG**

I. Introduction

1. Maximum Network Flow

The Ford–Fulkerson method or Ford–Fulkerson algorithm (FFA) is a [greedy algorithm](https://en.wikipedia.org/wiki/Greedy_algorithm) that computes the [maximum flow](https://en.wikipedia.org/wiki/Maximum_flow_problem) in a [flow network](https://en.wikipedia.org/wiki/Flow_network). It is sometimes called a "method" instead of an "algorithm" as the approach to finding augmenting paths in a residual graph is not fully specified or it is specified in several implementations with different running times. It was published in 1956 by [L. R. Ford, Jr.](https://en.wikipedia.org/wiki/L._R._Ford,_Jr.) and [D. R. Fulkerson](https://en.wikipedia.org/wiki/D._R._Fulkerson). The name "Ford–Fulkerson" is often also used for the [Edmonds–Karp algorithm](https://en.wikipedia.org/wiki/Edmonds%E2%80%93Karp_algorithm), which is a fully defined implementation of the Ford–Fulkerson method



2. Shorted Path

The Dijkstra Algorithm finds the shortest path from a source to all destinations in a directed graph (single source shortest path problem). During this process it will also determine a spanning tree for the graph.

**3. Minimum Spanning Tree**

Given a connected and undirected graph, a spanning tree of that graph is a subgraph that is a tree and connects all the vertices together. A single graph can have many different spanning trees. A minimum spanning tree (MST) or minimum weight spanning tree for a weighted, connected and undirected graph is a spanning tree with weight less than or equal to the weight of every other spanning tree. The weight of a spanning tree is the sum of weights given to each edge of the spanning tree.

II. State of the-art

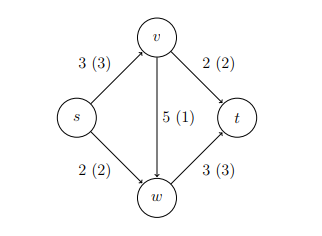
1. Maximum Network Flow

The maximum flow problem is a stone-cold classic in the design and analysis of algorithms. It’s easy to understand intuitively, so let’s do an informal example before giving the formal 3 definition.

The picture in Figure 1(a) resembles the ones you saw when studying shortest paths, but the semantics are different. Each edge is labeled with a capacity, the maximum amount of stuff that it can carry. The goal is to figure out how much stuff can be pushed from the vertex s to the vertex t..

For example, Figure 1(b) exhibits a method of pushing five units of flow from s to t, while respecting all edges’ capacities. Can we do better? Certainly not, since at most 5 units of flow can escape s on its two outgoing edges.

Formally, an instance of the maximum flow problem is specified by the following ingredients:

* A directed graph G, with vertices V and directed edges E;
* A source vertex s ∈
* A sink vertex t ∈ V ; A nonnegative and integral capacity ue for each edge e ∈ E

Denoting a flow by keeping track of the amount of flow on each edge. Flow amount is given in brackets.

2. Shorted Path

The idea of Dijkstra is simple.

Dijkstra partitions all nodes into two distinct sets: unsettled and settled. Initially all nodes are in the unsettled sets, e.g. they must be still evaluated. A node is moved to the settled set if a shortest path from the source to this node has been found.

Initially the distance of each node to the source is set to a very high value.

First only the source is in the set of unsettledNodes. The algorithms runs until the unsettledNodes are empty. In each iteration it selects the node with the lowest distance from the source out of the unsettled nodes. It reads all edges which are outgoing from the source and evaluates for each destination node, in the edges which are not yet settled, if the known distance from the source to this node can be reduced while using the selected edge. If this can be done then the distance is updated and the node is added to the nodes which need evaluation.

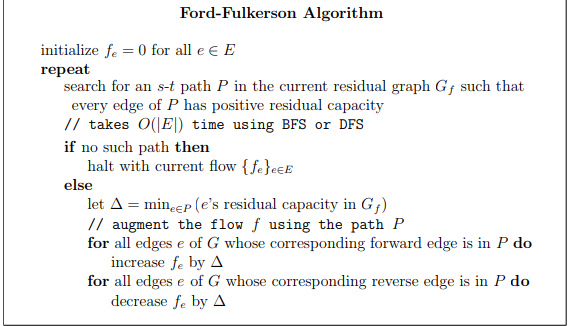
3. Minimum Spanning Tree

The classic "easy" optimization problem is to find the minimum spanning tree (MST) of a connected, undirected graph. Good polynomial-time algorithms have been known since 1930. Over the last 10 years, however, the standard O(m log n) results of Kruskal and Prim have been improved to linear or near-linear time. The new methods use several tricks of general interest in order to reduce the number of edge weight comparisons and the amount of other work. This tutorial reviews those methods, building up strategies step by step so as to expose the insights behind the algorithms. Implementation details are clarified, and some generalizations are given. Specifically, the paper attempts to shed light on the classical algorithms of Kruskal, of Prim, and of Bor uvka; the improved approach of Gabow, Galil, and Spencer, which takes time only O(m log(log\* n \Gamma log\* m n )); and the randomized O(m) algorithm of Karger, Klein, and Tarjan, which relies on an O(m) MST verification algorithm by King...

**III. Approaches**

1. Maximum Network Flow

Why should we care about the maximum flow problem? Like all central algorithmic problems, the maximum flow problem is useful in its own right, plus many different problems are really just thinly disguised version of maximum flow. For some relatively obvious and literal applications, the maximum flow problem can model the routing of traffic through a transportation network, packets through a data network, or oil through a distribution network

The idea behind the algorithm is as follows: as long as there is a path from the source (start node) to the sink (end node), with available capacity on all edges in the path, we send flow along one of the paths. Then we find another path, and so on. A path with available capacity is called an [augmenting path](https://en.wikipedia.org/wiki/Augmenting_path).

2. Shorted Path

import java.io.BufferedReader;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileNotFoundException;

import java.io.FileWriter;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.Scanner;

import java.util.logging.Level;

import java.util.logging.Logger;

public class Dijkstra

{

int soDinh ; //so dinh cua do thi

int G[][]; //ma tran trong so

int diemDau; //diem dau

int diemDich;

int doDai[] ; //do dai toi dinh i

int daXet[]; // danh dau la dinh co duong di la ngan nhat

int dinhTruoc[]; //luu vet dinh truoc

int O = 0; //gia tri vo cung

public Dijkstra()

{

soDinh = 0;

G = null;

doDai = null;

daXet=null;

dinhTruoc = null;

diemDau = 0;

O = 0;

}

//Doc du lieu

public boolean getDuLieuTuFile(String tenfile)

{

String chuoifile[]=tenfile.split(".");

String path=tenfile;

try

{

File file=new File(path);

if(!file.exists())

{

return false;

}

FileInputStream input = new FileInputStream(path);

InputStreamReader istream=new InputStreamReader(input);

BufferedReader reader=new BufferedReader(istream);

String sc;

//doc so dinh cua do thi , dinh bat dau , dinh dich

sc = reader.readLine();

String temp[] = sc.split(" ");

soDinh = Integer.parseInt(temp[0]);

diemDau = Integer.parseInt(temp[1]);

diemDich = Integer.parseInt(temp[2]);

G=new int[soDinh][soDinh];

int row=0;

//doc matran do thi

while ((sc = reader.readLine()) != null)

{

temp = sc.split(" ");

for (int col = 0; col < soDinh; col++)

{

G[row][col] = Integer.parseInt(temp[col]);

}

row++;

}

//dong file

reader.close();

istream.close();

input.close();

}

catch(FileNotFoundException ex)

{

System.err.println("LOi file");

}

catch (IOException ex)

{

System.err.println("Ngoai le xay ra.!");

}

return true;

}

public void fileWrite(String output) {

try {

FileWriter fw = new FileWriter(output);

if(diemDich!=diemDau)

{

//in duong di

if(doDai[diemDich] < O)

{

fw.write("Độ dài đường đi từ đỉnh " + diemDau + " đến đỉnh " + diemDich + " là: " + doDai[diemDich]+"\t----");

int mang[] = new int[soDinh];

int dem = 0;

int i=diemDich;

while (i != diemDau)

{

mang[dem++] = i;

i = dinhTruoc[i];

}

fw.write("\tChi tiết: " + diemDau);

for (int k = dem - 1; k >= 0; k--) {

fw.write("-->" + (mang[k]));

}

}

else

{

fw.write("\t\tKhông có đường đi từ đỉnh "+( ((diemDau)))+" đến đỉnh " + ( ((diemDich))));

}

fw.write("\n");

}

fw.close();

} catch (Exception e) {

System.out.println(e);

}

System.out.println("Success...");

}

public void thuatToan\_Dijkstra()

{

O=999999999;

//Gan trong so cho cac canh khong co duong di la vong cung

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

{

for (int j = 0; j < soDinh; j++)

{

if (i != j && G[i][j] == 0)

{

G[i][j] = O;

}

}

}

diemDau--;

doDai= new int[soDinh];

daXet=new int[soDinh];

dinhTruoc=new int[soDinh];

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

{

doDai[i] = O; // khoi tao do dai tu a toi moi dinh la vo cung

daXet[i] = 0; // danh sach cac diem da xet

dinhTruoc[i] = diemDau; // dat diem bat dau cua moi diem la a

}

int i=0;

//Khoi tao d(diemDau,diemDau)=0;

doDai[diemDau] = 0;

for(int dinh=0;dinh<soDinh;dinh++)

{

for (i = 0; i < soDinh; i++)// tim mot diem chua xet ma co duong di < vo cung

{

if (daXet[i] != 1 && doDai[i] < O)

{

break;

}

}

if(i==soDinh)//khong co dinh nao thoa man

{

break;

}

for (int j = 0; j < soDinh; j++)// tim dinh ma co do dai la nho nhat

{

if (daXet[j]!=1 && doDai[i] > doDai[j])

{

i = j;

}

}

daXet[i] = 1; // cho i vao danh sach xet roi

for (int j = 0; j < soDinh; j++) // tinh lai do dai cua cac diem chua xet

{

if (daXet[j]!=1 && doDai[i] + G[i][j] < doDai[j])

{

doDai[j] = doDai[i] + G[i][j]; // thay doi do dai cua d[i,j]

dinhTruoc[j] = i; // danh dau diem truoc j la i

}

}

}

}

public void inDuongDi()

{

for(int dinh=0;dinh<soDinh;dinh++)

if(dinh!=diemDau)

{

//in duong di

if(doDai[dinh] < O)

{

//System.out.print("\t\tĐộ dài đường đi từ đỉnh " + ((char)(diemDau+65)) + " đến đỉnh " + ((char)(dinh+65)) + " là: " + doDai[dinh]+"\t----");

int mang[] = new int[soDinh];

int dem = 0;

int i=dinh;

while (i != diemDau)

{

mang[dem++] = i;

i = dinhTruoc[i];

}

// System.out.print("\tChi tiết: " + ((char)(diemDau+65)));

for (int k = dem - 1; k >= 0; k--) {

//System.out.print("-->" + (char)(mang[k]+65));

}

}

else

{

//System.out.println("\t\tKhông có đường đi từ đỉnh "+( ((char)(diemDau+65)))+" đến đỉnh " + ( ((char)(dinh+65))));

}

// System.out.println("\n");

}

}

public void inDuong2Di()

{

System.out.println("Nhap dinh dich: ");

Scanner sc=new Scanner(System.in);

int diemCuoi=sc.nextInt();

//in duong di

if(doDai[diemCuoi] < O)

{

//System.out.print("\t\tĐộ dài đường đi từ đỉnh " + ((char)(diemDau+65)) + " đến đỉnh " + ((char)(diemCuoi+65)) + " là: " + doDai[diemCuoi]+"\t----");

int mang[] = new int[soDinh];

int dem = 0;

int i=diemCuoi;

while (i != diemDau)

{

mang[dem++] = i;

i = dinhTruoc[i];

}

System.out.print("\tChi tiết: " + ((char)(diemDau+65)));

for (int k = dem - 1; k >= 0; k--) {

//System.out.print("-->" + (char)(mang[k]+65));

}

}

else

{

//System.out.println("\t\tKhông có đường đi từ đỉnh "+( ((char)(diemDau+65)))+" đến đỉnh " + ( ((char)(diemCuoi+65))));

}

}

public void inMatran()

{

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

{

for(int j=0;j<soDinh;j++)

{

if(G[i][j]==0)

{

}

else

{

}

}

}

}

}

3. Minimum Spanning Tree

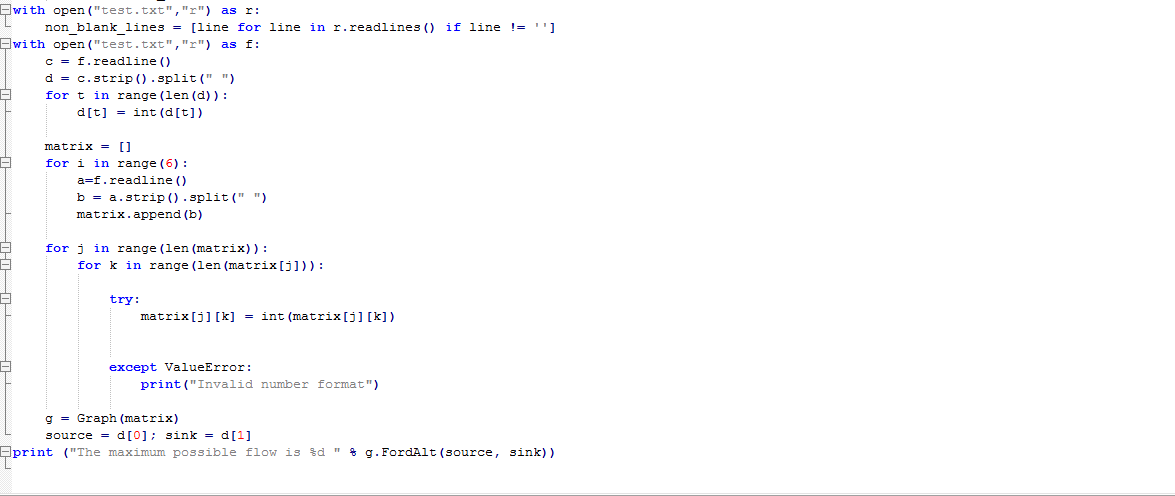
The given problem can be solved using Kruskal’s algorithm to find the Minimum Spanning tree. We find the vertex which has maximum degree in the graph. At first we will perform the union of all the edges which are incident to this vertex and than carry out normal Kruskal’s algorithm. This gives us optimal spanning tree.

IV. Implementation and Results

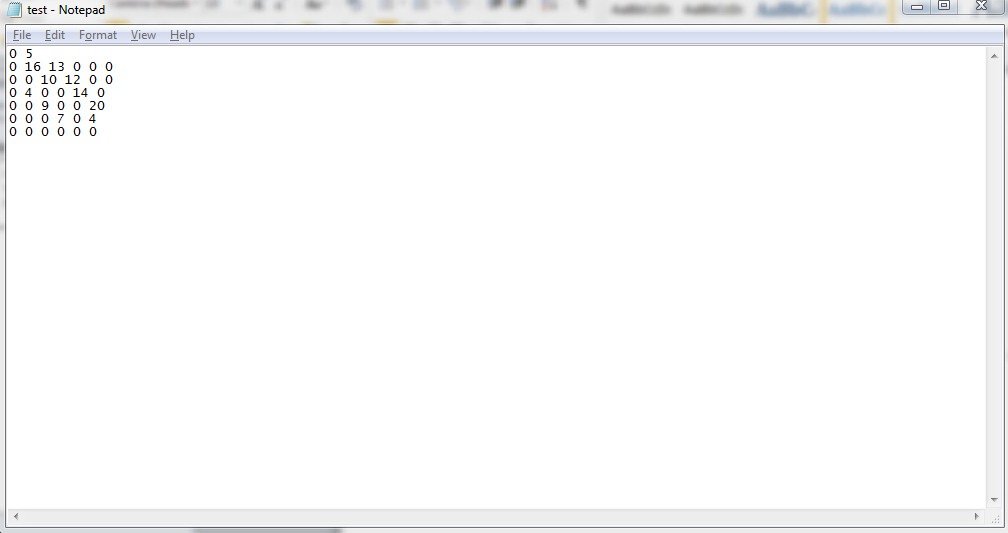
1 Maximum Network Flow

File maximum.py

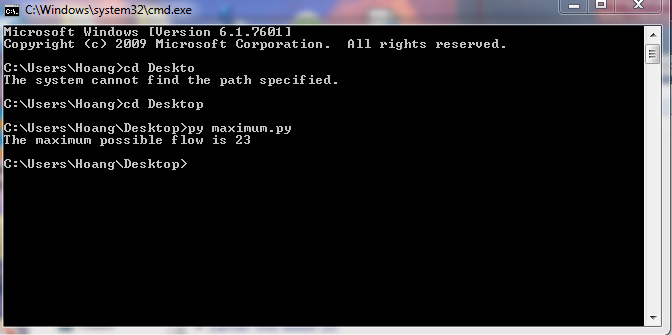




File test.txt

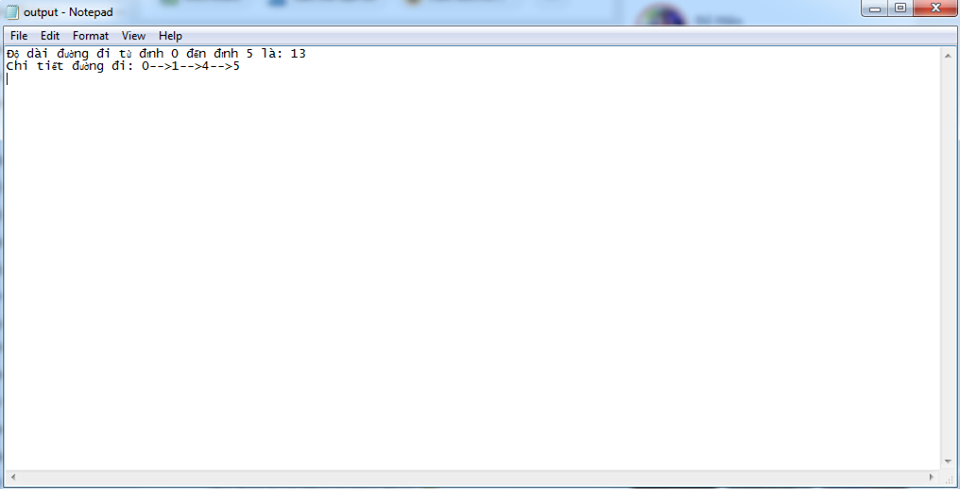


Result



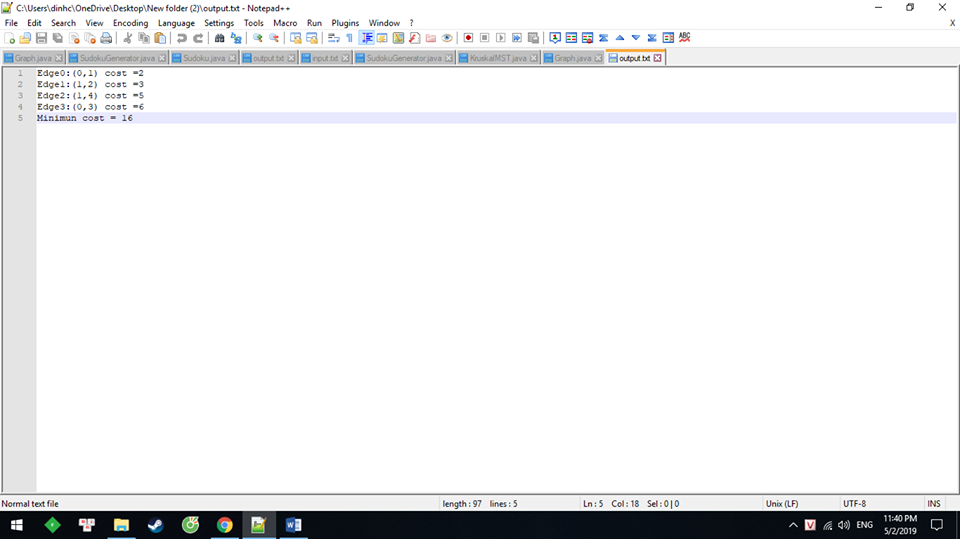
2 Shorted Path

Result



3. Minimum Spanning Tree

Result



TÀI LIỆU THAM KHẢO

[1]Minimum spanning tree:

https://en.wikipedia.org/wiki/Minimum\_spanning\_tree

[2]Maximum flow problem:

https://en.wikipedia.org/wiki/Maximum\_flow\_problem

[3] Shortest path problem:

https://en.wikipedia.org/wiki/Shortest\_path\_problem

[4]https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-in-java-using-priorityqueue/

[5]https://www.geeksforgeeks.org/kruskals-minimum-spanning-tree-algorithm-greedy-algo-2/