TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO MÔN TOÁN TỔ HỢP VÀ ĐỒ THỊ**

**BÀI TẬP LỚN**

*Người thực hiện*: **HUỲNH NGỌC TIẾN – 51702194**

**HUỲNH VĂN HOÀI – 51702105**

**ĐOÀN TUẤN KIỆT - 51702125**

Lớp **: 17050201**

Khoá  **: 21**

**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**



**BÁO CÁO MÔN TOÁN TỔ HỢP VÀ ĐỒ THỊ**

**BÀI TẬP LỚN**

*Người thực hiện*: **HUỲNH NGỌC TIẾN – 51702194**

**HUỲNH VĂN HOÀI – 51702105**

**ĐOÀN TUẤN KIỆT - 51702125**

Lớp **: 17050201**

Khoá  **: 21**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2019**

LỜI CẢM ƠN

Xin cảm ơn giảng viên đã cung cấp kiến thức và tạo điều kiện để em có thể hoàn thành bài báo cáo cũng như củng cố kiến thức cho bản thân để có thể áp dụng

# ĐỒ ÁN ĐƯỢC HOÀN THÀNH

**TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

Tôi xin cam đoan đây là sản phẩm đồ án của riêng tôi; 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.

**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.

*TP. Hồ Chí Minh, ngày 22 tháng 02 năm 2019*

*Tác giả*

*(ký tên và ghi rõ họ tên)*

*Huỳnh Ngọc Tiến*

*Huỳnh Văn Hoài*

*Đoàn Tuấn Kiệt*

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)

DANH MỤC CÁC HÌNH ẢNH

[***Hình 4. 1 Sơ đồ ví dụ Maximum Network Flow* 9**](#_Toc7813645)

[***Hình 4. 2 Kết quả Maximum Network Flow* 14**](#_Toc7813646)

[***Hình 4. 3 Sơ đồ ví dụ Shortest Path* 15**](#_Toc7813647)

[***Hình 4. 4 Kết quả Shortest Path* 18**](#_Toc7813648)

[***Hình 4. 5 Kết quả Minimum Spanning Tree* 22**](#_Toc7813649)

**MỤC LỤC**

[**ĐỒ ÁN ĐƯỢC HOÀN THÀNH ii**](#_Toc7814463)

[**PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN iii**](#_Toc7814464)

[**MỤC LỤC 1**](#_Toc7814465)

[**CHƯƠNG I: INTRODUCTION 3**](#_Toc7814466)

[**1. MAXIMUM NETWORK FLOW 3**](#_Toc7814467)

[**2. SHORTEST PATH 4**](#_Toc7814468)

[**3. MINIMUM SPANNING TREE 4**](#_Toc7814469)

[**CHƯƠNG II: STATE OF THE ART 6**](#_Toc7814470)

[**1. Ford – Fulkerson algorithm 6**](#_Toc7814471)

[**2. Dijkstra algorithm 6**](#_Toc7814472)

[**3. Prim algorithm 6**](#_Toc7814473)

[**CHƯƠNG III: APPROACH 7**](#_Toc7814474)

[**1. Maximum Network Flow 7**](#_Toc7814475)

[**2. Shotest Path 7**](#_Toc7814476)

[**3. Minimum Spanning Tree 7**](#_Toc7814477)

[**CHƯƠNG IV: EXPERIMENT AND RESULT 8**](#_Toc7814478)

[**1. Maximum Network Flow 8**](#_Toc7814479)

[**1.1 Code: 8**](#_Toc7814480)

[**1.2 Kết quả Maximum Network Flow 13**](#_Toc7814481)

[**2. Shortest Path 14**](#_Toc7814482)

[**2.1 Code: 14**](#_Toc7814483)

[**2.2 Kết quả ShortestPath 16**](#_Toc7814484)

[**3. Minimum Spanning Tree 18**](#_Toc7814485)

[**3.1 Code 18**](#_Toc7814486)

[**3.2 Kết quả Minimum Spanning Tree 22**](#_Toc7814487)

[**CHƯƠNG V: CONCLUSION 23**](#_Toc7814488)

[**1. Maximum Network Flow (Ford & Fullkerson) 23**](#_Toc7814489)

[**2. Shortest Path (Dijkstra) 23**](#_Toc7814490)

[**3. Minimum Spanning Tree (Prim) 23**](#_Toc7814491)

[**TÀI LIỆU THAM KHẢO 24**](#_Toc7814492)

# CHƯƠNG I: INTRODUCTION

## 1. MAXIMUM NETWORK FLOW

Là một trong những bài toán tối ưu trên đồ thị tìm được những ứng dụng rất rộng rãi trong cả thực tế cũng như trong lý thuyết tổ hợp. Bài toán được đề xuất vào đầu những năm 1950 và gắn liền với tên tuổi của 2 nhà toán học Mỹ [Lester Randolph Ford](https://vi.wikipedia.org/w/index.php?title=Lester_Randolph_Ford&action=edit&redlink=1) và [Delbert Ray Fulkerson](https://vi.wikipedia.org/wiki/Delbert_Ray_Fulkerson).

Trong lý thuyết đồ thị, một mạng lưu lượng được định nghĩa là một đồ thị có hướng liên quan đến một nguồn (S) và chìm (T) và một số nút khác được kết nối với các cạnh. Mỗi cạnh có một công suất riêng là giới hạn lưu lượng tối đa mà cạnh đó có thể cho phép. Lưu lượng trong mạng phải tuân theo các điều kiện sau:

* Đối với bất kỳ nút không nguồn và không chìm, luồng đầu vào bằng với luồng đầu ra.
* Đối với bất kỳ cạnh (E) trong mạng , 0 flow(Ei) Capacity(Ei).
* Tổng lưu lượng ra khỏi nút nguồn bằng tổng lưu lượng đến nút chìm.
* Luồng ròng ở các cạnh theo đối xứng nghiêng, tức là dòng chảy từ nút u đến nút v. Điều này dẫn đến một kết luận trong đó bạn phải tổng hợp tất cả các luồng giữa hai nút (một trong hai hướng) để tìm luồng ròng giữa các nút ban đầu.

Lưu lượng cực đại:

Nó được định nghĩa là lưu lượng tối đa mà mạng sẽ cho phép truyền từ nguồn sang chìm. Nhiều thuật toán tồn tại trong việc giải quyết vấn đề dòng chảy tối đa. Hai thuật toán chính để giải quyết các loại vấn đề này là thuật toán Ford-Fulkerson và Thuật toán Dinic.

## 2. SHORTEST PATH

Bài toán đường đi ngắn nhất nguồn đơn là bài toán tìm một [đường đi](https://vi.wikipedia.org/w/index.php?title=%C4%90%C6%B0%E1%BB%9Dng_%C4%91i_(l%C3%BD_thuy%E1%BA%BFt_%C4%91%E1%BB%93_th%E1%BB%8B&action=edit&redlink=1) giữa hai đỉnh sao cho tổng các trọng số của các cạnh tạo nên đường đi đó là nhỏ nhất. Một mạng, được xác định bởi hai bộ ký hiệu: nút và cung. Một vòng cung bao gồm một cặp đỉnh được sắp xếp và biểu thị một hướng chuyển động có thể xảy ra giữa các đỉnh. Đường dẫn như vậy là một chuỗi trong đó nút đầu cuối của mỗi cung được xác định cho nút ban đầu của cung tiếp theo. Vấn đề tìm đường đi ngắn nhất (đường dẫn có độ dài tối thiểu) từ nút 1 đến bất kỳ nút nào khác trong mạng được gọi là Vấn đề đường dẫn ngắn nhất.

Do đó, vấn đề đường đi ngắn nhất là vấn đề tìm đường đi giữa hai đỉnh (hoặc nút) trong biểu đồ sao cho tổng trọng số của các cạnh cấu thành của nó được giảm thiểu.

## 3. MINIMUM SPANNING TREE

Với một [đồ thị liên thông](https://vi.wikipedia.org/wiki/%C4%90%E1%BB%93_th%E1%BB%8B_li%C3%AAn_th%C3%B4ng), [vô hướng](https://vi.wikipedia.org/wiki/%C4%90%E1%BB%93_th%E1%BB%8B_v%C3%B4_h%C6%B0%E1%BB%9Bng) cho trước, [cây bao trùm](https://vi.wikipedia.org/wiki/C%C3%A2y_bao_tr%C3%B9m) của nó là một [đồ thị con](https://vi.wikipedia.org/wiki/Thu%E1%BA%ADt_ng%E1%BB%AF_l%C3%BD_thuy%E1%BA%BFt_%C4%91%E1%BB%93_th%E1%BB%8B#%C4%90%E1%BB%93_th%E1%BB%8B_con) có dạng [cây](https://vi.wikipedia.org/w/index.php?title=%C4%90%E1%BB%93_th%E1%BB%8B_c%C3%A2y&action=edit&redlink=1) và có tất cả các [đỉnh](https://vi.wikipedia.org/wiki/%C4%90%E1%BB%93_th%E1%BB%8B_(l%C3%BD_thuy%E1%BA%BFt_%C4%91%E1%BB%93_th%E1%BB%8B)) liên thông với nhau. Một đồ thị có thể có nhiều cây bao phủ khác nhau. Chúng ta cũng có thể gán một trọng số cho mỗi cạnh, là con số biểu thị sự "không ưa thích" và dùng nó để tính toán trọng số của một cây bao trùm bằng cách cộng tất cả trọng số của cạnh trong cây bao trùm đó. Khi đó, một cây bao trùm nhỏ nhất là một cây bao trùm có trọng số bé hơn bằng trọng số của tất cả các cây bao trùm khác. Tổng quát hơn, bất kỳ một đồ thị vô hướng (không nhất thiết liên thông) đều có một rừng bao phủ nhỏ nhất, là hội của các cây bao trùm nhỏ nhất của các [thành phần liên thông](https://vi.wikipedia.org/wiki/Th%C3%A0nh_ph%E1%BA%A7n_li%C3%AAn_th%C3%B4ng) của nó.

Về cơ bản cây khung được sử dụng để tìm các đường ngắn nhất để kết nối tất cả các nút trong một Graph. Các ứng dụng phổ biến của cây khung là: Lập kế hoạch mạng dân sự Giao thức định tuyến mạng máy tính Cluster Analysis Chúng ta tìm hiểu ví dụ đơn giản sau để hiểu các ứng dụng này. Bạn thử tưởng tượng một mạng internet trong thành phố là một hình Graph lớn và bây giờ kế hoạch đặt ra là triển khai các đường dây mạng sao cho với độ dài dây là ngắn nhất mà vẫn có thể kết nối được tất cả các nút trong thành phố. Đó là một ví dụ giải thích cho ứng dụng của cây khung.

Ví dụ như một hãng TV truyền hình cáp muốn nối cáp đến một khu dân cư mới. Nếu bị ràng buộc chỉ được chôn cáp ở một số tuyến đường nhất định, ta sẽ có thể hình thành được một đồ thị biểu diễn các điểm kết nối với nhau theo các tuyến đường đó. Một số tuyến có chi phí cao hơn, vì chúng dài hơn, hoặc cáp phải được chôn sâu hơn; những con đường này sẽ được thể hiện bằng những cạnh có trọng số lớn hơn. Một cây bao trùm của đồ thị sẽ là một tập con các con đường như vậy sao cho nó không được tạo thành vòng (chu trình) mà vẫn phải nối được đến tất cả các nhà. Sẽ có thể có vài cây bao trùm như vậy. Một cây bao trùm nhỏ nhất sẽ là cây bao trùm có tổng chi phí thấp nhất.

# CHƯƠNG II: STATE OF THE ART

## 1. Ford – Fulkerson algorithm

Thuật toán Ford- Fulkerson dùng để tính toán [luồng cực đại](https://vi.wikipedia.org/wiki/B%C3%A0i_to%C3%A1n_lu%E1%BB%93ng_c%E1%BB%B1c_%C4%91%E1%BA%A1i) trong một [mạng vận tải](https://vi.wikipedia.org/w/index.php?title=M%E1%BA%A1ng_v%E1%BA%ADn_t%E1%BA%A3i&action=edit&redlink=1). Miễn là tồn tại một đường đi từ nguồn (nút bắt đầu) đến điểm xả (nút cuối), với điều kiện tất cả các cung trên đường đi đó vẫn còn khả năng thông qua, thì ta sẽ gửi đi một luồng dọc theo đường đi đó. Sau đó chúng ta tìm một đường đi khác, và tiếp tục như vậy. Một đường đi còn khả năng thông qua là một đường đi có khả năng mở rộng thêm hay một đường đi mà luồng qua đó còn khả năng tăng thêm

## 2. Dijkstra algorithm

**Thuật toán Dijkstra** là một thuật toán giải quyết bài toán đường đi ngắn nhất trong một đồ thị có hướng **không có cạnh trọng số âm.**

Cho 1 đồ thị có hướng **G = (V, E)** với các cạnh có trọng số không âm, có dữ liệu nhập vào là ma trận trọng số **L** và 2 đỉnh **x, y** cho trước. Việc ta cần làm là tìm đường đi ngắn nhất từ **x** đến **y** trong đồ thị **G**.

Việc chúng ta cần làm là chỉ ra đỉnh **v** bất kì sao cho **x -> v** là đường đi ngắn nhất. Ta gọi **length[v]** là giá trị đường đi ngắn nhất từ **x -> v**, có thể hiểu **length[v]** là giá trị đường đi ngắn nhất trong các đường đi từ đỉnh **x**qua các đỉnh trong tập hợp S (nếu có) rồi đến **v**.

## 3. Prim algorithm

Thuật toán Prim là một [thuật toán tham lam](https://vi.wikipedia.org/wiki/Gi%E1%BA%A3i_thu%E1%BA%ADt_tham_lam) để tìm [cây bao trùm nhỏ nhất](https://vi.wikipedia.org/wiki/C%C3%A2y_bao_tr%C3%B9m_nh%E1%BB%8F_nh%E1%BA%A5t) của một [đồ thị vô hướng](https://vi.wikipedia.org/wiki/%C4%90%E1%BB%93_th%E1%BB%8B_v%C3%B4_h%C6%B0%E1%BB%9Bng) [có trọng số](https://vi.wikipedia.org/w/index.php?title=%C4%90%E1%BB%93_th%E1%BB%8B_c%C3%B3_tr%E1%BB%8Dng_s%E1%BB%91&action=edit&redlink=1) [liên thông](https://vi.wikipedia.org/wiki/%C4%90%E1%BB%93_th%E1%BB%8B_li%C3%AAn_th%C3%B4ng). Nghĩa là nó tìm một tập hợp các [cạnh](https://vi.wikipedia.org/wiki/%C4%90%E1%BB%93_th%E1%BB%8B_(l%C3%BD_thuy%E1%BA%BFt_%C4%91%E1%BB%93_th%E1%BB%8B)) của đồ thị tạo thành một [cây](https://vi.wikipedia.org/wiki/C%C3%A2y_(l%C3%BD_thuy%E1%BA%BFt_%C4%91%E1%BB%93_th%E1%BB%8B)) chứa tất cả các [đỉnh](https://vi.wikipedia.org/wiki/%C4%90%E1%BB%93_th%E1%BB%8B_(l%C3%BD_thuy%E1%BA%BFt_%C4%91%E1%BB%93_th%E1%BB%8B)), sao cho tổng trọng số các cạnh của cây là nhỏ nhất.

# CHƯƠNG III: APPROACH

## 1. Maximum Network Flow

Cách tiếp Maximum Network Flow là bắt đầu với luồng bằng 0 và tạo ra các luồng có giá trị cao hơn. Khi đi từ s đến t sẽ thêm các đường dẫn để đến được vị trí t sao cho có giá trị cao nhất

## 2. Shotest Path

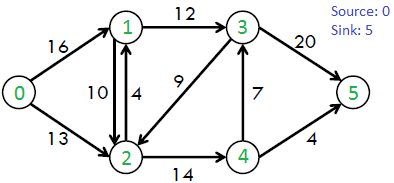
Cách tiếp cận hiệu quả là các thuật toán đường dẫn ngắn nhất được giới thiệu cho các mạng xác định có thể dễ dàng được sử dụng để xác định đường dẫn với thời gian di chuyển tối thiểu dự kiến trong mạng ngẫu nhiên. Tuy nhiên, đường dẫn tối ưu được xác định theo phương pháp này có thể không đáng tin cậy, vì cách tiếp cận này không giải quyết được sự thay đổi thời gian di chuyển. Để giải quyết vấn đề này, một số nhà nghiên cứu sử dụng phân phối thời gian di chuyển thay vì giá trị dự kiến của nó để họ tìm phân phối xác suất của tổng thời gian di chuyển bằng các phương pháp tối ưu hóa khác nhau như lập trình động và thuật toán của Dijkstra.

## 3. Minimum Spanning Tree

Vấn đề này cũng có thể được phân phối theo cách phân tán. Nếu mỗi nút được coi là một máy tính và không có nút nào biết bất cứ điều gì ngoại trừ các liên kết được liên kết của chính nó, vẫn có thể tính toán cây bao trùm tối thiểu.

# CHƯƠNG IV: EXPERIMENT AND RESULT

1. Maximum Network Flow



*Hình 4. 1 Sơ đồ ví dụ Maximum Network Flow*

**1.1 Code:**

import java.util.\*;

import java.lang.\*;

import java.io.\*;

public class MaximumFlow

{

public static final int S = 6; //S la so luong dinh

//tra ve gia tri la true neu di tu s den t co trong do thi

boolean BFS(int Graph[][],int s,int t,int parent[])

{

boolean visited[] = new boolean[S];

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

visited[i]=false;

LinkedList<Integer> queue = new LinkedList<Integer>();

queue.add(s);

visited[s] = true;

parent[s]=-1;

while (queue.size()!=0)

{

int u = queue.poll();

for (int v=0; v<S; v++)

{

if (visited[v]==false && Graph[u][v] > 0)

{

queue.add(v);

parent[v] = u;

visited[v] = true;

}

}

}

return (visited[t] == true);

}

public int maxFlow(int graph[][],int s,int t)

{

int u, v;

int rGraph[][] = new int[S][S];

for (u = 0; u < S; u++)

for (v = 0; v < S; v++)

rGraph[u][v] = graph[u][v];

int parent[] = new int[S];

int maxFlow = 0;

while (BFS(rGraph, s, t, parent))

{

int pathFlow = Integer.MAX\_VALUE;

for (v=t; v!=s; v=parent[v])

{

u = parent[v];

pathFlow = Math.min(pathFlow, rGraph[u][v]);

}

for (v=t; v != s; v=parent[v])

{

u = parent[v];

rGraph[u][v] -= pathFlow;

rGraph[v][u] += pathFlow;

}

maxFlow += pathFlow;

}

return maxFlow;

}

private static void DFS(int[][]rGraph,int s,boolean[] visited) {

visited[s] = true;

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

if (rGraph[s][i] > 0 && !visited[i]) {

DFS(rGraph, i, visited);

}

}

}

public void minCut(int[][] graph, int s, int t) {

int u,v;

int[][] rGraph = new int[S][S];

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

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

rGraph[i][j] = graph[i][j];

}

}

int[] parent = new int[S];

while (BFS(rGraph, s, t, parent)) {

int pathFlow = Integer.MAX\_VALUE;

for (v = t; v != s; v = parent[v]) {

u = parent[v];

pathFlow = Math.min(pathFlow, rGraph[u][v]);

}

for (v = t; v != s; v = parent[v]) {

u = parent[v];

rGraph[u][v] = rGraph[u][v] - pathFlow;

rGraph[v][u] = rGraph[v][u] + pathFlow;

}

}

boolean[] isVisited = new boolean[S];

DFS(rGraph, s, isVisited);

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

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

if (graph[i][j] > 0 && isVisited[i] && !isVisited[j]) {

System.out.println(i + " - " + j);

}

}

}

}

public static void main (String[] args)

{

// Tạo biểu đồ

int graph[][] = { {0, 16, 13, 0, 0, 0},

{0, 0, 10, 12, 0, 0},

{0, 4, 0, 0, 14, 0},

{0, 0, 9, 0, 0, 20},

{0, 0, 0, 7, 0, 4},

{0, 0, 0, 0, 0, 0}};

MaximumFlow n = new MaximumFlow();

System.out.println("Maximum Flow: " + n.maxFlow(graph, 0, 5)); //giả sử đi từ đỉnh 0 đến đỉnh 5

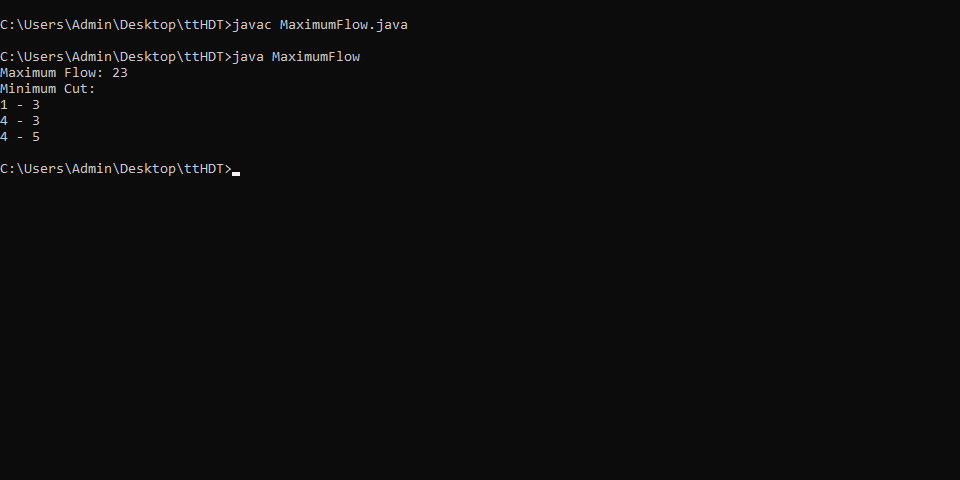
System.out.println("Minimum Cut:");

n.minCut(graph,0,5);

}

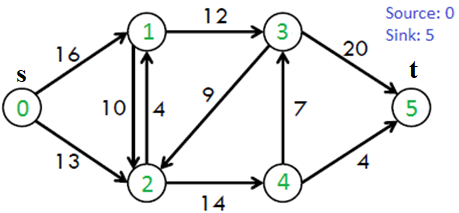
}

**1.2** **Kết quả** Maximum Network Flow



*Hình 4. 2 Kết quả Maximum Network Flow*

2. Shortest Path



*Hình 4. 3 Sơ đồ ví dụ Shortest Path*

2.1 Code:

import java.util.\*;

class ShortestPath {

static final int V = 6;

int minDistance(int dist[], Boolean set[])

{

//Tao gia tri nho nhat

int min = Integer.MAX\_VALUE, minIndex = -1;

for (int v = 0; v < V; v++)

if (set[v] == false && dist[v] <= min) {

min = dist[v];

minIndex = v;

}

return minIndex;

}

void print(int dist[])

{

System.out.println("Khoang cach ngan nhat tu s toi t la: "+dist[V-1] );

}

void solution(int graph[][], int src)

{

int dist[] = new int[V];

Boolean set[] = new Boolean[V];

//Khoi tao khoang cach la INFI va set[i] la false

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

dist[i] = Integer.MAX\_VALUE;

set[i] = false;

}

//Khoang cach tu source toi source = 0

dist[src] = 0;

//Tim duong di ngan nhat cho cac dinh

for (int count = 0; count < V - 1; count++) {

int u = minDistance(dist, set);

set[u] = true;

for (int v = 0; v < V; v++)

if (!set[v] && graph[u][v] != 0 &&

dist[u] != Integer.MAX\_VALUE && dist[u] + graph[u][v] < dist[v])

dist[v] = dist[u] + graph[u][v];

}

print(dist);

}

public static void main(String[] args)

{

int graph[][] = { {0, 16, 13, 0, 0, 0},

{0, 0, 10, 12, 0, 0},

{0, 4, 0, 0, 14, 0},

{0, 0, 9, 0, 0, 20},

{0, 0, 0, 7, 0, 4},

{0, 0, 0, 0, 0, 0}};

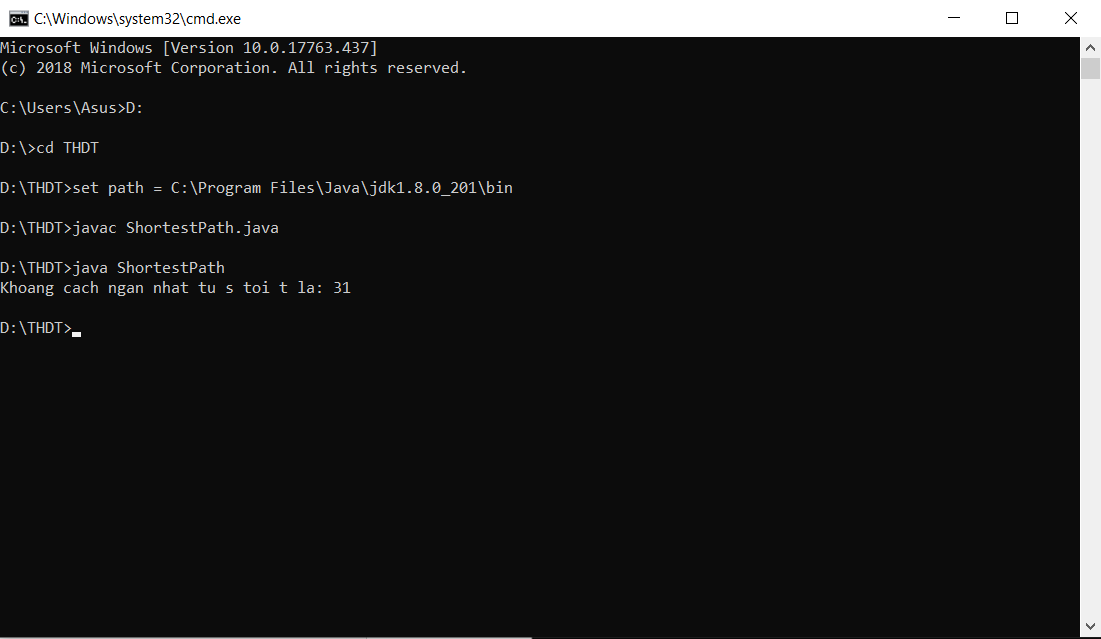
ShortestPath t = new ShortestPath();

t.solution(graph, 0);

}

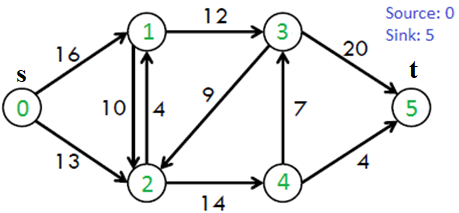
}

2.2 Kết quả ShortestPath



*Hình 4. 4 Kết quả Shortest Path*

3. Minimum Spanning Tree



*Hình 4. 5 Sơ đồ ví dụ Minimum Spanning Tree*

3.1 Code

import java.util.ArrayList;

import java.util.Comparator;

import java.util.PriorityQueue;

public class MinimumSpanningTree {

static class Edge {

int source;

int destination;

int weight;

public Edge(int source, int destination, int weight) {

this.source = source;

this.destination = destination;

this.weight = weight;

}

}

static class Graph {

int vertices;

ArrayList<Edge> allEdges = new ArrayList<>();

Graph(int vertices) {

this.vertices = vertices;

}

public void addEgde(int source, int destination, int weight) {

Edge edge = new Edge(source, destination, weight);

allEdges.add(edge);

}

public void kruskalMST(){

PriorityQueue<Edge> pq = new PriorityQueue<>(allEdges.size(), Comparator.comparingInt(o -> o.weight));

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

pq.add(allEdges.get(i));

}

int [] parent = new int[vertices];

makeSet(parent);

ArrayList<Edge> mst = new ArrayList<>();

int index = 0;

while(index<vertices-1){

Edge edge = pq.remove();

int x\_set = find(parent, edge.source);

int y\_set = find(parent, edge.destination);

if(x\_set==y\_set){

}else {

mst.add(edge);

index++;

union(parent,x\_set,y\_set);

}

}

System.out.println("Minimum Spanning Tree: ");

printGraph(mst);

}

public void makeSet(int [] parent){

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

parent[i] = i;

}

}

public int find(int [] parent, int vertex){

if(parent[vertex]!=vertex)

return find(parent, parent[vertex]);;

return vertex;

}

public void union(int [] parent, int x, int y){

int x\_set\_parent = find(parent, x);

int y\_set\_parent = find(parent, y);

parent[y\_set\_parent] = x\_set\_parent;

}

public void printGraph(ArrayList<Edge> edgeList){

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

Edge edge = edgeList.get(i);

System.out.println("Edge-" + i + " source: " + edge.source +

" destination: " + edge.destination +

" weight: " + edge.weight);

}

}

}

public static void main(String[] args) {

int vertices = 6;

Graph graph = new Graph(vertices);

graph.addEgde(0, 1, 4);

graph.addEgde(0, 2, 3);

graph.addEgde(1, 2, 1);

graph.addEgde(1, 3, 2);

graph.addEgde(2, 3, 4);

graph.addEgde(3, 4, 2);

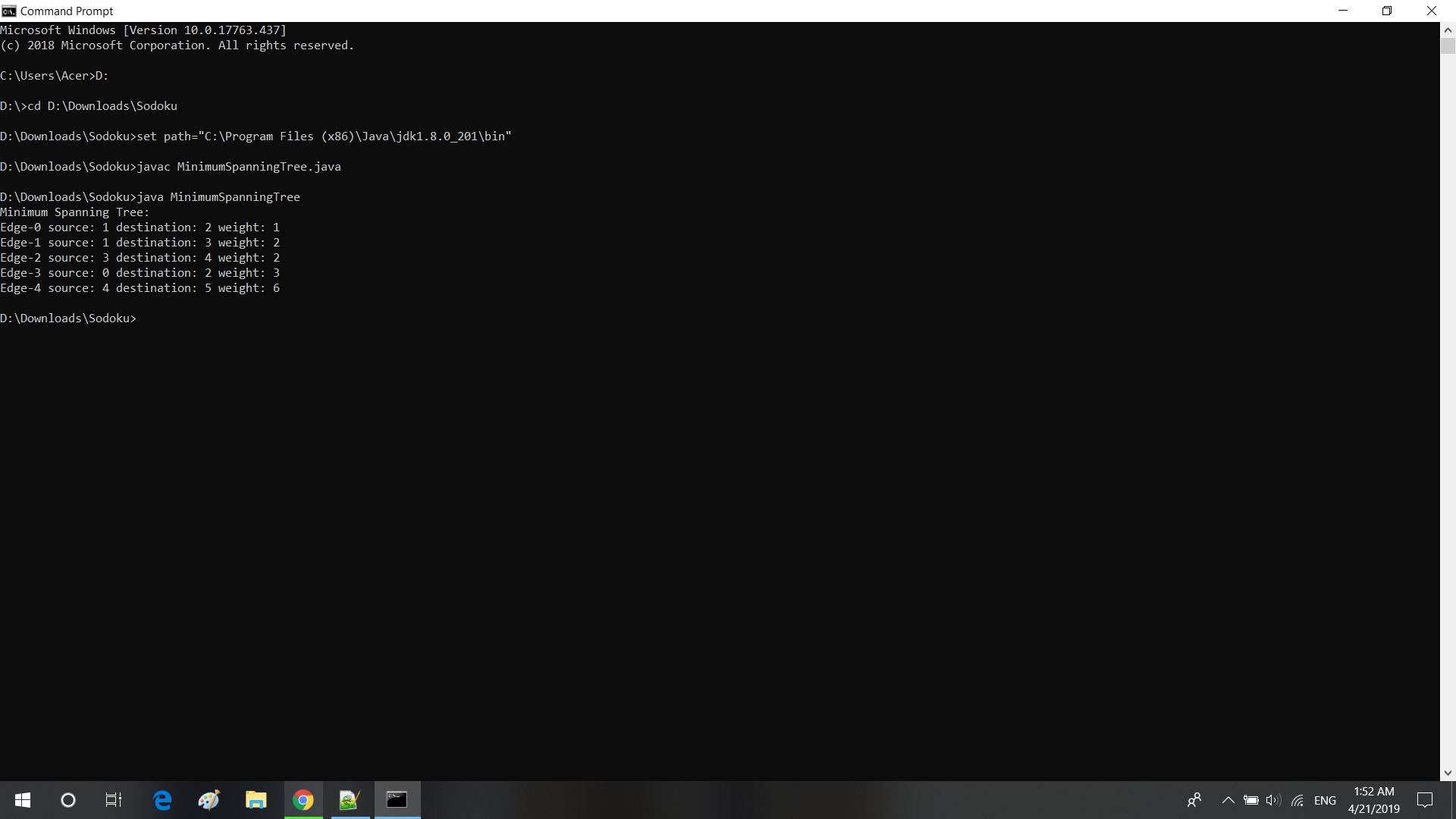
graph.addEgde(4, 5, 6);

graph.kruskalMST();

}

}

3.2 Kết quả Minimum Spanning Tree



*Hình 4. 6 Kết quả Minimum Spanning Tree*

# CHƯƠNG V: CONCLUSION

1. Maximum Network Flow (Ford & Fullkerson)

Bài toán tìm luồng cực đại (trên mạng) là bài toán cực đại hóa | f | với các ràng buộc trên. Giải thuật đầu tiên được sử dụng để tìm luồng cực đại là giải thuật đường tăng luồng (augmenting path algorithm) của Ford & Fullkerson đã độc lập cải tiến giải thuật đường tăng luồng để có được giải thuật có độ phức tạp thời gian đa thức. Kể từ đó, đã có rất nhiều giải thuật hiệu quả hơn được phát triển có thể kể đến như phương pháp đẩy/gán nhãn lại.

2. Shortest Path (Dijkstra)

Tóm lại, Dijkstra là thuật toán tìm đường ngắn nhất từ một đỉnh tới tất cả các đỉnh còn lại trong đồ thị.

Thuật toán có thể áp dụng cho cả đồ thị vô hướng và có hướng.

Dijkstra là một thuật toán kinh điển được áp dụng rất nhiều trong thực tế, ví dụ như tìm đường ngắn nhất trên ứng dụng bản đồ, tìm đường ngắn nhất để truyền dữ liệu trong mạng viễn thông, mạng máy tính với nhiều hub và router.

3. Minimum Spanning Tree (Prim)

Độ phức tạp thời gian là O (VlogV + ElogV) = O (ElogV), làm cho nó giống như thuật toán của Kruskal. Tuy nhiên, thuật toán của Prim có thể được cải thiện bằng cách sử dụng Fibros Heaps thành O (E + logV).

Thuật toán của Prim nhanh hơn đáng kể trong giới hạn khi bạn có một biểu đồ thực sự dày đặc với nhiều cạnh hơn các đỉnh.

Sử dụng thuật toán của Prim khi bạn có một biểu đồ có nhiều cạnh

# TÀI LIỆU THAM KHẢO

<https://vi.wikipedia.org/wiki/Thu%E1%BA%ADt_to%C3%A1n_Ford-Fulkerson>

<https://vi.wikipedia.org/wiki/Lu%E1%BB%93ng_c%E1%BB%B1c_%C4%91%E1%BA%A1i>

<https://vi.wikipedia.org/wiki/Thu%E1%BA%ADt_to%C3%A1n_Dijkstra>

<https://vi.wikipedia.org/wiki/B%C3%A0i_to%C3%A1n_%C4%91%C6%B0%E1%BB%9Dng_%C4%91i_ng%E1%BA%AFn_nh%E1%BA%A5t>

<https://vi.wikipedia.org/wiki/Thu%E1%BA%ADt_to%C3%A1n_Prim>

<https://vi.wikipedia.org/wiki/C%C3%A2y_bao_tr%C3%B9m_nh%E1%BB%8F_nh%E1%BA%A5t>