TRƯỜNG ĐẠI HỌC NHA TRANG

**KHOA CÔNG NGHỆ THÔNG TIN**

-----🙞🙜🕮🙞🙜-----

****

**BÁO CÁO THỰC TẬP CƠ SỞ**

**ĐỀ TÀI: “Cài đặt và ứng dụng giải thuật tìm đường đi ngắn nhất trong định tuyến và các ứng dụng liên quan”**

**Tên sinh viên: Ngô Trung Tín**

**Lớp học phần: 63.CNTT-5**

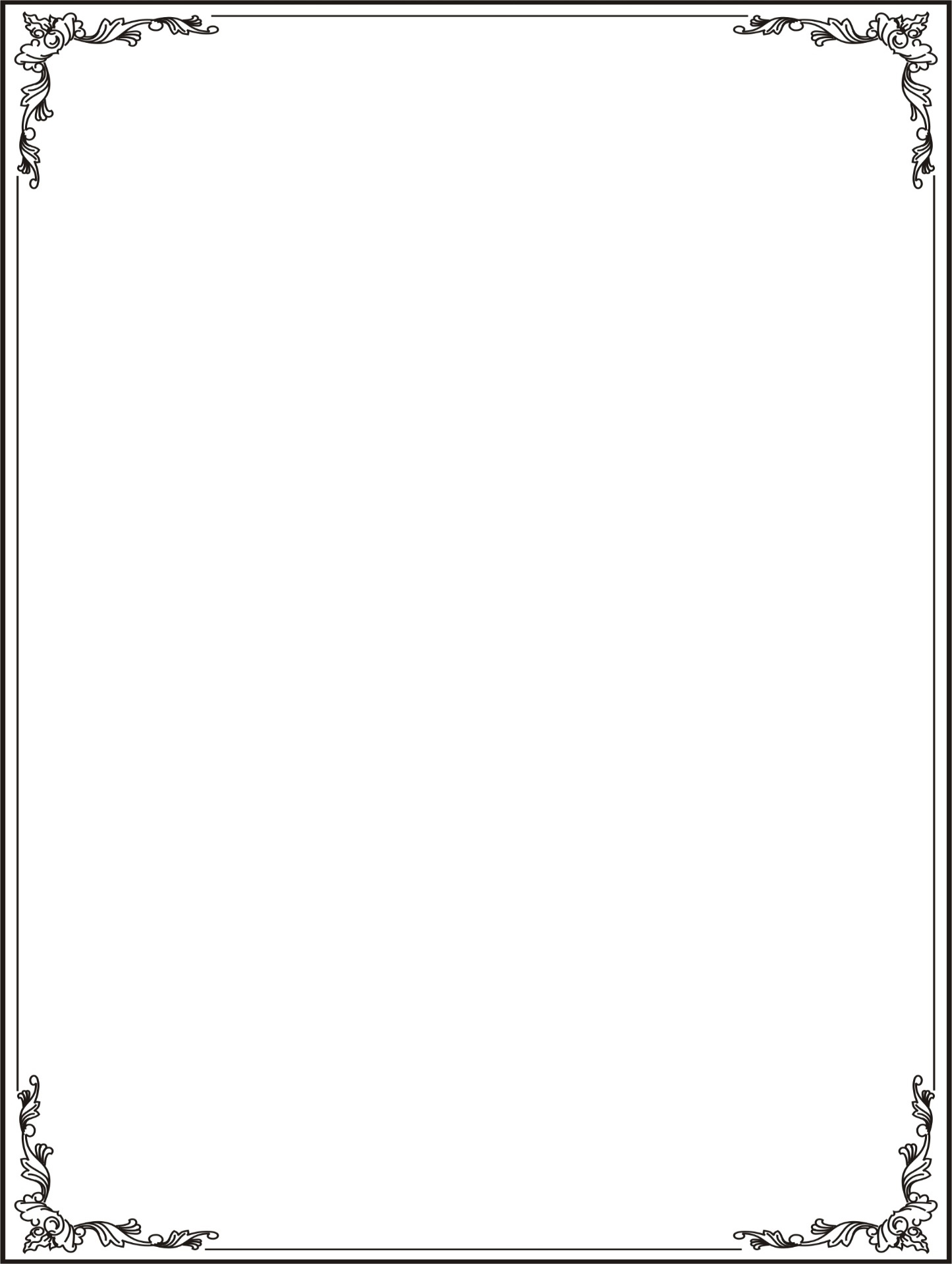
**MSSV: 63135722**

**SĐT: 0589473569**

**Email:** [**tin.ntru.63cntt@ntu.edu.vn**](mailto:tin.ntru.63cntt@ntu.edu.vn)

**Giảng viên hướng dẫn: Cẩn Thị Phượng**

**Khánh Hòa – 2023**

TRƯỜNG ĐẠI HỌC NHA TRANG

**KHOA CÔNG NGHỆ THÔNG TIN**

-----🙞🙜🕮🙞🙜-----

****

**Khánh Hòa – 2023**

# MỤC LỤC

[LỜI MỞ ĐẦU 7](#_Toc155297404)

[CHƯƠNG 1. TỔNG QUAN ĐỀ TÀI 8](#_Toc155297405)

[CHƯƠNG 2. CƠ SỞ LÝ THUYẾT 9](#_Toc155297406)

[2.1 ĐỒ THỊ, ĐỊNH TUYẾN VÀ ĐƯỜNG ĐI NGẮN NHẤT(theo Wikipedia) 9](#_Toc155297407)

[2.1.1 Đồ thị 9](#_Toc155297408)

[2.1.1.1 Khái niệm đồ thị 9](#_Toc155297409)

[2.1.1.2 Phân loại đồ thị 9](#_Toc155297410)

[2.1.1.3 Biểu diễn đồ thị 10](#_Toc155297411)

[2.1.2 Định tuyến trong đồ thị và đường đi ngắn nhất 11](#_Toc155297412)

[2.1.2.1 Định tuyến 11](#_Toc155297413)

[2.1.2.2 Đường đi ngắn nhất 11](#_Toc155297414)

[2.2 Thuật toán Dijkstra 11](#_Toc155297415)

[2.2.1 Lịch sử 11](#_Toc155297416)

[2.2.2 Độ phức tạp 11](#_Toc155297417)

[2.2.3 Nguyên lí giải thuật 11](#_Toc155297418)

[2.2.4 Minh họa 12](#_Toc155297419)

[CHƯƠNG 3. KẾT QUẢ NGHIÊN CỨU VÀ THẢO LUẬN 15](#_Toc155297420)

[3.1 Giới thiệu 15](#_Toc155297421)

[3.1.1 Tổng quan về Code 15](#_Toc155297422)

[3.1.1.1 Cấu trúc dữ liệu Canh và Dinh 15](#_Toc155297423)

[3.1.1.2 Lớp nhập dữ liệu ban đầu 16](#_Toc155297424)

[3.1.1.3 Lớp Graph 17](#_Toc155297425)

[3.1.1.4 Lớp Dijkstra 18](#_Toc155297426)

[3.1.1.5 Hàm main 19](#_Toc155297427)

[3.1.2 Thông tin trong file đầu vào và kết quả đầu ra 19](#_Toc155297428)

[3.1.2.1 File chứa dữ liệu đầu vào 19](#_Toc155297429)

[3.1.2.2 Kết quả đầu ra 20](#_Toc155297430)

[3.2 Chi tiết thuật toán 20](#_Toc155297431)

[3.3 So sánh với thuật toán A\* 24](#_Toc155297432)

[3.3.1 Giới thiệu A\* 24](#_Toc155297433)

[3.3.2 Code thuật toán A\* 24](#_Toc155297434)

[3.3.3 So sánh 25](#_Toc155297435)

[CHƯƠNG 4. KẾT LUẬN 27](#_Toc155297436)

[TÀI LIỆU THAM KHẢO 28](#_Toc155297437)

[References 28](#_Toc155297438)

**MỤC LỤC HÌNH ẢNH**

[Hình 1. Đồ thị 7 đỉnh và 12 cạnh 8](file:///D:\Tài%20Liệu%20Học%20Tập\HK2_2023_2024\TTCS\BAOCAO.docx#_Toc155298766)

[Hình 2. Đồ thị có hướng 8](file:///D:\Tài%20Liệu%20Học%20Tập\HK2_2023_2024\TTCS\BAOCAO.docx#_Toc155298767)

[Hình 3. Đồ thị trọng số vô hướng 9](file:///D:\Tài%20Liệu%20Học%20Tập\HK2_2023_2024\TTCS\BAOCAO.docx#_Toc155298768)

[Hình 4. Ma *trận* đồ thị trọng số vô hướng từ Hình 3 9](file:///D:\Tài%20Liệu%20Học%20Tập\HK2_2023_2024\TTCS\BAOCAO.docx#_Toc155298769)

[Hình 5. Đồ thị trọng số 11](file:///D:\Tài%20Liệu%20Học%20Tập\HK2_2023_2024\TTCS\BAOCAO.docx#_Toc155298770)

[Hình 6. Vòng lặp đầu tiên 11](#_Toc155298771)

[Hình 7. Vòng lặp thứ hai 12](#_Toc155298772)

[Hình 8. Vòng lặp thứ ba 12](#_Toc155298773)

[Hình 9. Vòng lặp cuối 12](file:///D:\Tài%20Liệu%20Học%20Tập\HK2_2023_2024\TTCS\BAOCAO.docx#_Toc155298774)

[Hình 10. Kết quả minh họa 13](file:///D:\Tài%20Liệu%20Học%20Tập\HK2_2023_2024\TTCS\BAOCAO.docx#_Toc155298775)

[Hình 11. Cấu trúc cạnh và đỉnh 14](file:///D:\Tài%20Liệu%20Học%20Tập\HK2_2023_2024\TTCS\BAOCAO.docx#_Toc155298776)

[Hình 12. Lớp GraphNhapXuat 15](#_Toc155298777)

[Hình 13. Lớp Graph 16](#_Toc155298778)

[Hình 14. Lớp Dijkstra 17](#_Toc155298779)

[Hình 15. Hàm main 18](file:///D:\Tài%20Liệu%20Học%20Tập\HK2_2023_2024\TTCS\BAOCAO.docx#_Toc155298780)

[Hình 16. File .txt chứa các dòng dữ liệu 18](file:///D:\Tài%20Liệu%20Học%20Tập\HK2_2023_2024\TTCS\BAOCAO.docx#_Toc155298781)

[Hình 17. Kết quả khi thực hiện thuật toán 19](file:///D:\Tài%20Liệu%20Học%20Tập\HK2_2023_2024\TTCS\BAOCAO.docx#_Toc155298782)

# LỜI MỞ ĐẦU

Trong một thế giới liên kết hiện đại, vai trò của việc tìm kiếm đường đi ngắn nhất đóng vai trò quan trọng trong nhiều lĩnh vực, từ giao thông đến viễn thông và mạng máy tính. Việc xác định đường đi ngắn nhất giúp tiết kiệm thời gian, di chuyển dễ dàng và tối ưu hóa giao thông. Trong viễn thông vì xác định được đường đi tối ưu nên dữ liệu, thông tin được truyền đi chính xác và ít mất mát. Trong mạng máy tính thì các thuật toán xác định được đường dẫn tối ưu giữa các thiết bị. Tìm kiếm đường đi ngắn nhất là một yếu tố then chốt quyết định hiệu suất và hiệu quả của hệ thống và ứng dụng.

Theo bài báo của Cục thông tin khoa học và Công nghệ Quốc gia đăng trên trang web vista.gov.vn vào năm 2022 có viết như sau: “Theo Ericsson (2020), lưu lượng dữ liệu mạng di động tăng 50% giữa quý 3 năm 2019 và quý 3 năm 2020. Lưu lượng dữ liệu toàn cầu lần lượt đạt 180 và 230 exabyte  (1018 byte: 1 tỷ tỷ) mỗi tháng vào năm 2019 và 2020. Đến năm 2026, khối lượng này được dự báo sẽ tăng hơn gấp ba lần, đạt 780 exabyte mỗi tháng” . Qua đó, ta có thể thấy được khi dữ liệu càng tăng thì càng phải có các phương pháp truyền đi thích hợp. Điều này dẫn tới việc phát triển các thuật toán tìm đường trở nên quan trọng.

Vậy nên trước hết ta cần là nắm vững các kiến thức về tìm kiếm đường đi và các thuật toán phổ biến. Cần phải hiểu rõ các thuật toán tìm đường đi cơ bản, từ đó mới có thể đủ nền tảng để tạo ra, xử lí các vấn đề về định tuyến.

Mục tiêu đề tài:

* Trình bày được các lí thuyết đồ thị liên quan và thuật toán Dijkstra
* Xây dựng được thuật toán Dijkstra bằng cách tiếp cận khác(không dùng hàng đợi ưu tiên)
* So sánh được thuật toán Dijkstra với thuật toán A\*

Giới hạn và phạm vi đề tài:

* Sử dụng các kiến thức tích lũy để xây dựng thuật toán
* Đối tượng nghiên cứu: thuật toán Dijkstra và A\*
* Được triển khai trên đồ thị tự xây dựng

# CHƯƠNG 1. TỔNG QUAN ĐỀ TÀI

Theo Báo cáo đề tài: “Cài đặt thuật toán tìm đường đi ngắn nhất trong đồ thị” của Đỗ Thị Huyền, niên khóa 2010-2014. Bài báo cáo nghiên cứu về cấu trúc của ngôn ngữ xây dựng thuật toán, khái niệm cơ bản của đồ thị và cách cài đặt thuật toán Dijkstra. Em tham khảo về “Giải thuật Dijkstra” từ bài báo cáo trên.

Ưu điểm của nghiên cứu là trình bày được các vấn đề liên quan đến thuật toán, xác định được vấn đề và mô hình hóa bài toán cần giải quyết. Nhưng vẫn còn hạn chế như xây dựng lại những cái có sẵn, chưa làm rõ độ phức tạp và không có sự so sánh với các thuật toán khác.

Phương pháp nghiên cứu bao gồm tìm hiểu và phân tích kĩ lưỡng các thuật toán có sẵn, thử nghiệm các cách tiếp cận thuật toán dựa trên những nghiên cứu trước.

Chương 2: giới thiệu các lí thuyết liên quan đến nghiên cứu: đồ thị, nguyên lí thuật toán và minh họa được thuật toán.

Chương 3: Hiểu được những code có sẵn. Xây dựng code dùng cấu trúc dữ liệu khác. Đi sâu chi tiết từng bước thuật toán hoạt động và so sánh được các thuật toán.

Kết luận và hướng phát triển được trình bày ở cuối chương.

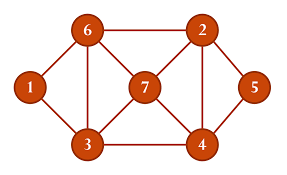
# CHƯƠNG 2. CƠ SỞ LÝ THUYẾT

## 2.1 ĐỒ THỊ, ĐỊNH TUYẾN VÀ ĐƯỜNG ĐI NGẮN NHẤT(theo Wikipedia)

### 2.1.1 Đồ thị

#### 2.1.1.1 Khái niệm đồ thị

Đồ thị là một bộ hai tập hợp hữu hạn: tập hợp đỉnh và tập hợp cạnh nối các đỉnh với nhau.

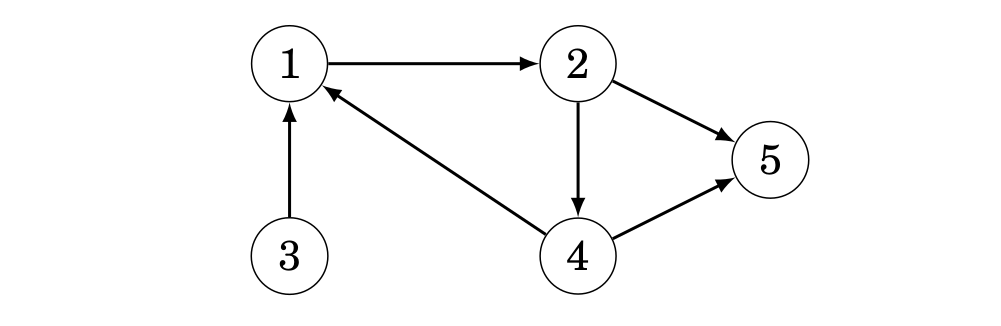
 Kí hiệu: G = (V, E). Trong đó: V: tập hợp đỉnh, E: tập hợp cạnh

Hình 1. Đồ thị 7 đỉnh và 12 cạnh

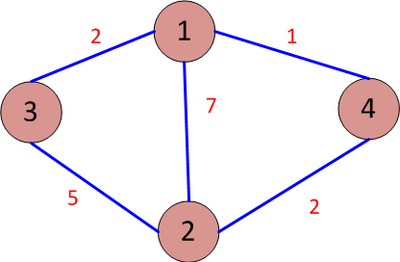
#### 2.1.1.2 Phân loại đồ thị

Đồ thị vô hướng G = (V, E): tập cạnh E không phân biệt cạnh nối từ đỉnh X đến Y với cạnh nối từ Y đến X. (Hình 1.)

Đồ thị có hướng G = (V, A): tập cạnh A phân biệt cạnh nối từ đỉnh X đến Y với cạnh nối từ Y đến X.

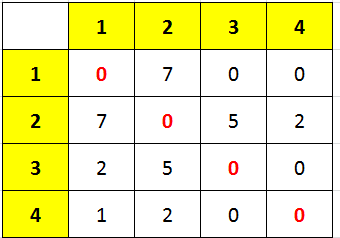


Hình 2. Đồ thị có hướng

 Đồ thị trọng số G = (V, E/A, w): tập cạnh E/A có tập trọng số w tương ứng với mỗi cạnh .

Hình 3. Đồ thị trọng số vô hướng

#### 2.1.1.3 Biểu diễn đồ thị

 Ma trận kề: mảnh hai chiều trong đó các hàng và cột tương ứng với các đỉnh của đồ thị. Mỗi phần tử trong ma trận thể hiện trọng số của cạnh giữa hai đỉnh, nếu không có cạnh nối thì sử dụng giá trị “inf”(vô cùng).

Hình 4. Ma *trận* đồ thị trọng số vô hướng từ Hình 3

Danh sách kề: Danh sách kề lưu trữ thông tin về mỗi đỉnh và các đỉnh kề với nó cùng với trọng số của cạnh. Đối với mỗi đỉnh, ta liệt kê danh sách các đỉnh kề và trọng số tương ứng.

Ví dụ: từ hình 3, đỉnh 1 có 3 cạnh kề(3, 2, 4) nên sẽ tạo một cấu trúc dữ liệu phù hợp đề nhập các giá trị lần lượt: (1, 3, 2), (1, 2, 7), (1, 4, 1).

## 2.1.2 Định tuyến trong đồ thị và đường đi ngắn nhất

#### 2.1.2.1 Định tuyến

Định tuyến trong đồ thị: là quá trình xác định và tính toán các đường đi từ một điểm xuất phát đến một điểm đích trong một mạng lưới hoặc đồ thị. Có thể dùng các thuật toán như: A\*, Dijkstra, Bellman-Ford.

#### 2.1.2.2 Đường đi ngắn nhất

Đường đi ngắn nhất: đường đi từ một đỉnh đến một đỉnh khác với tổng trọng số của các cạnh trên đường đi đó là nhỏ nhất so với tất cả các đường đi khác giữa hai đỉnh. Các thuật toán thường được sử dụng: Dijkstra, Bellman-Ford(có thể sử dụng A\* nhưng không phải lúc nào cũng trả về kết quả tốt nhất).

## 2.2 Thuật toán Dijkstra

### 2.2.1 Lịch sử

Thuật toán Dijkstra là một thuật toán quan trọng trong lĩnh vực đồ thị, được sử dụng để tìm đường đi ngắn nhất từ một đỉnh xuất phát đến tất cả các đỉnh còn lại trong đồ thị có trọng số không âm. Thuật toán này được phát minh bởi nhà toán học Edsger W. Dijkstra vào năm 1956 và vẫn là một trong những thuật toán phổ biến nhất trong lý thuyết đồ thị và ứng dụng thực tế.

### 2.2.2 Độ phức tạp

Thuật toán có độ phức tạp về thời gian khi áp dụng cách giải trình từ là O(V^2) với V là số lượng đỉnh trong đồ thị. Khi sử dụng **priority queue**(hàng đợi ưu tiên), độ phức tạp của thuật toán được cải thiện thành O((E + V) \* log(V)) với E là số cạnh trong đồ thị. Độ phức tạp về không gian là O(V) khi sử dụng hàng đợi ưu tiên [1]

### 2.2.3 Nguyên lí giải thuật

Theo [2], thuật toán Dijkstra sẽ có 5 bước hoạt động:

* Gán giá trị tất cả các Node(đỉnh) thành vô hạn và Node bắt đầu bằng 0.
* Đánh dấu các Node là “chưa truy cập” và Node bắt đầu là “bắt đầu”.
* Xem xét các đỉnh kề “chưa truy cập” và tính toán khoảng cách từ Node bắt đầu. So sánh các giá trị mới tính toán được với khoảng cách từ Node bắt đầu và gán lại giá trị nhỏ hơn.
* Khi tất cả Node kề được truy cập từ Node bắt đầu, đánh dấu các đỉnh thành “đã truy cập”. Các đỉnh này sẽ không được kiểm tra lại và giá trị lưu lần cuối sẽ là giá trị nhỏ nhất.
* Thiết lập các Node “đã truy cập” như là một Node bắt đầu mới, quay lại bước 3.

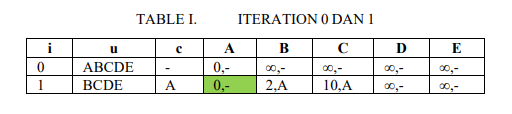
### 2.2.4 Minh họa

Hình 5. Đồ thị trọng số

Theo [2], ta có đồ thị trên, đỉnh bắt đầu là A và đích là đỉnh E

Bước 1:

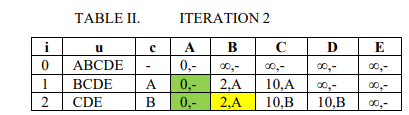
* Tạo bảng kết quả khoảng cách, i là số lần lặp, u là các Node chưa truy cập, v là các node đã truy cập, c là Node đang xét,
* i = 0 là điều kiện ban đầu
* Khi i = 1, AB = 2, AC = 10. AB là nhỏ nhất 🡪 B là node bắt đầu khi i = 2



Hình 6. Vòng lặp đầu tiên

Bước 2:

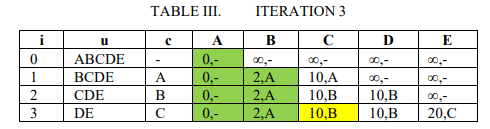
* Khi i = 2, BC = 10, BD = 10
* Node hiện tại là C[10, B] có cùng khoảng cách với C[10, A]. Vậy kết quả khoảng cách Node C có thể là một trong 2
* C là Node bắt đầu cho i = 3



Hình 7. Vòng lặp thứ hai

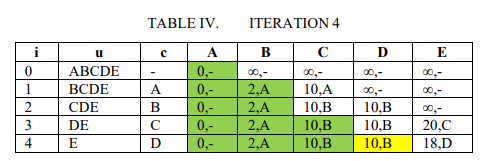
Bước 3:

* Khi i = 3, CD = 15, CE = 20.
* Tại Node đang xét D[15, C] có giá trị khoảng cách lớn hơn D[10, B], vậy khoảng cách sẽ thay đổi từ D[15, C] thành D[10, B].
* Chọn D là Node bắt đầu cho i = 4.



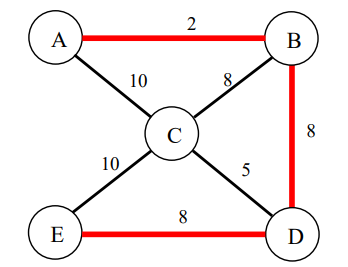
Hình 8. Vòng lặp thứ ba

Bước 4:

* Khi i = 4, DE = 18.
* Node đang xét là E[20, C] lớn hơn E[18, D] vậy giá trị khoảng cách của D không thay đổi.
* Vì E là đích nên kết thúc.

Hình 9. Vòng lặp cuối

Vậy ta có kết quả đường đi là A-B-D-E như hình dưới



Hình 10. Kết quả minh họa

# CHƯƠNG 3. KẾT QUẢ NGHIÊN CỨU VÀ THẢO LUẬN

## 3.1 Giới thiệu

Sau khi tìm hiểu về thuật toán Dijkstra, em sẽ tìm hiểu sâu hơn về cấu trúc dữ liệu để xây dựng thuật toán.

### 3.1.1 Tổng quan về Code

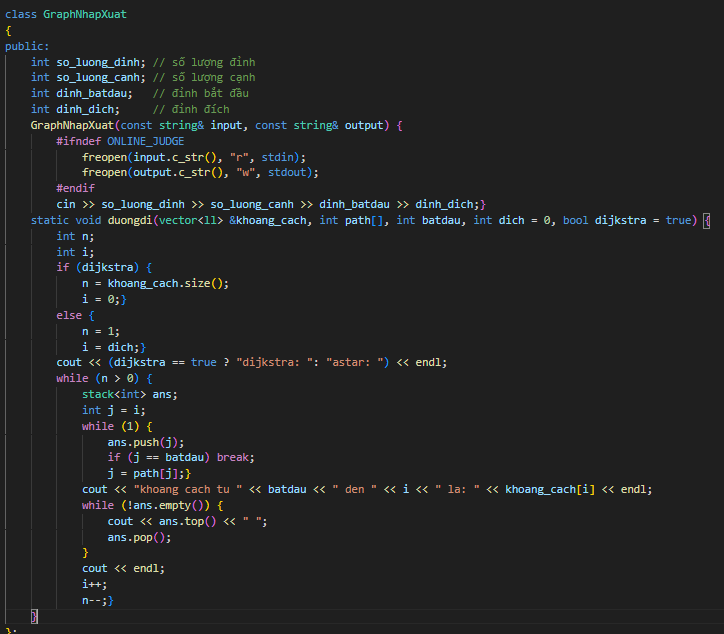
#### 3.1.1.1 Cấu trúc dữ liệu Canh và Dinh

Hình 11. Cấu trúc cạnh và đỉnh

Cấu trúc Canh dùng để biểu diễn một cạnh tương ứng của đồ thị. Khi khởi tạo một biến Canh sẽ gồm có 2 phần tử: int dinh để lưu các giá trị đỉnh(hoặc đỉnh kề tùy mục đích sử dụng) và int trong\_so để lưu trong số tương ứng trên cạnh. Nếu không có các đối số đầu vào, khởi tạo 2 giá trị thành INF(một giá trị rất lớn).

Trong đồ thị, một đỉnh có thể kết nối đến một hoặc nhiều đỉnh khác vì vậy một đối tượng đỉnh có thể sẽ có một hoặc nhiều cấu trúc Canh, mỗi Canh có giá trị dinh và trong\_so tương ứng. Vì không sử dụng vector nên em tạo thêm một biến int so\_canh\_ke để lưu trữ số cạnh kề để thuận tiện khi sử dụng. Và khi không có đối số đầu vào, so\_canh\_ke sẽ mang giá trị 0 và Canh sẽ là INF.

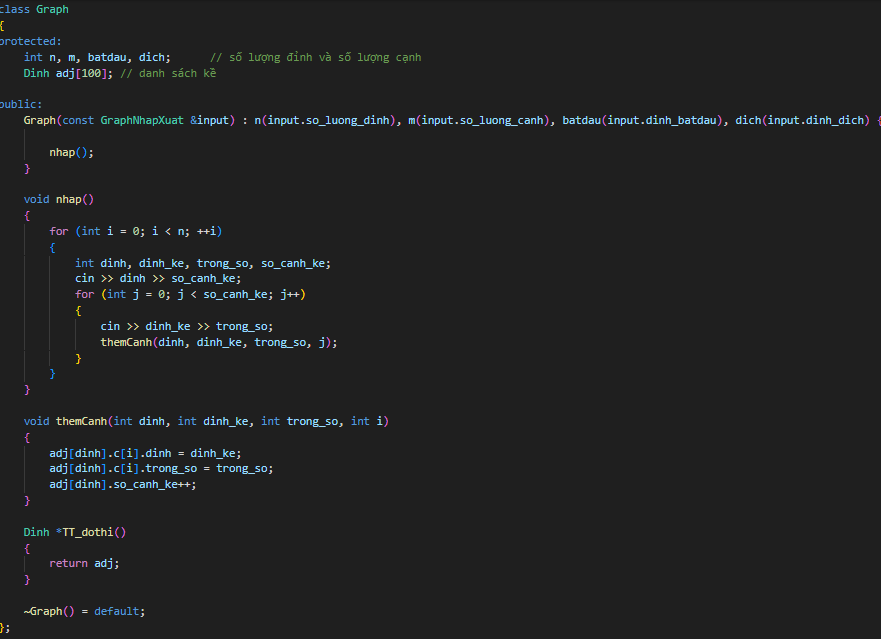
#### 3.1.1.2 Lớp nhập dữ liệu ban đầu



Hình 12. Lớp GraphNhapXuat

Lớp trên nhận 4 giá trị đầu vào đầu tiên là số lượng đỉnh, số lượng cạnh, đỉnh bắt đầu và đỉnh đích, đồng thời khi khởi tạo đối tượng cần thêm 2 đối số là file dữ liệu đầu vào và file đầu ra. Phương thức **duongdi** được gọi từ bên ngoài để ghi lại các kết quả .

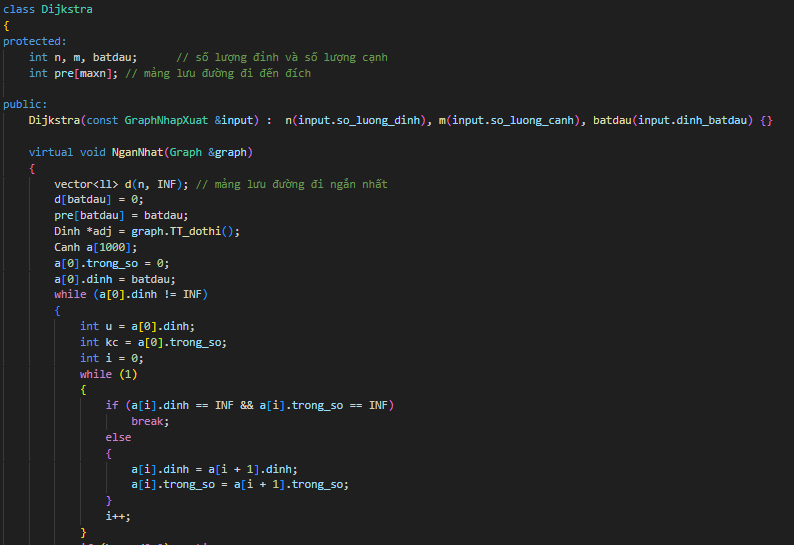
#### 3.1.1.3 Lớp Graph

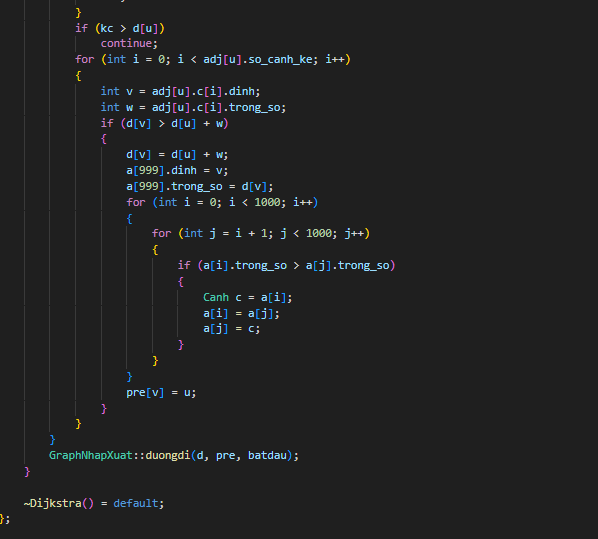


Hình 13. Lớp Graph

Lớp này đảm nhận khởi tạo thông tin đồ thị nhận các giá trị từ lớp GraphNhapXuat đã nhập trước đó, từ đó ta có thêm Dinh adj[100] để lưu thông tin đỉnh, đỉnh kề và trọng số. Phương thức TT\_dothi() dùng để lấy thông tin đồ thị khi được gọi đến

#### 3.1.1.4 Lớp Dijkstra





Hình 14. Lớp Dijkstra

Lớp này chứa thông tin về đồ thị và thuật toán Dijkstra tìm đường đi ngắn nhất với đối số là một đối tượng lớp Graph lưu giữ thông tin về đỉnh, đỉnh kề và trọng số. Ngoài ra lớp có thêm mảng int pre[maxn] để cấu trúc đường đi từ đỉnh bắt đầu tới các đỉnh còn lại và sử dụng phương thức **duongdi(d, pre, batdau)** để ghi kết quả ra file.

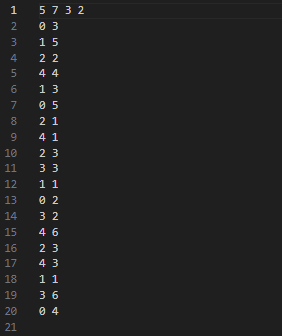
#### 3.1.1.5 Hàm main

Hình 15. Hàm main

Trong main chứa các đối tượng được khởi tạo và phương thức được thực hiện.

### 3.1.2 Thông tin trong file đầu vào và kết quả đầu ra

#### 3.1.2.1 File chứa dữ liệu đầu vào



Hình 16. File .txt chứa các dòng dữ liệu

Dòng đầu tiên chứa các giá trị lần lượt là số lượng đỉnh, số lượng cạnh, đỉnh bắt đầu và đỉnh đích

Dòng thứ 2 trở đi là các thông tin về đồ thị:

* Dòng thứ 2 (0, 3) chứa tên đỉnh(0) và số lượng cạnh kề(3) của đỉnh đó, dựa vào số lượng cạnh kề mà (3) dòng tiếp theo nhập thông tin tương ứng
* Các dòng tiếp theo chưa tên đỉnh kề và trọng số của cạnh

Giải thích lí do đồ thị chỉ có 7 cạnh nhưng có đến 14 lần nhập giá trị cho các cạnh là bởi đồ thị được xây dựng trong bài là đồ thị vô hướng nên nếu có đường đi từ 1 đỉnh A đến 1 đỉnh B thì sẽ có từ đường đi từ đỉnh B đến A

#### 3.1.2.2 Kết quả đầu ra

Hình 17. Kết quả khi thực hiện thuật toán

Dòng đầu tiên in ra khoảng cách ngắn nhất từ đỉnh bắt đầu tới các đỉnh còn lại trên đồ thị. Dòng thứ hai ghi ra đường đi từ đỉnh đích đến các đỉnh còn lại.

## Chi tiết thuật toán

GraphNhapXuat userInput("input1.txt", "output4124.txt");

Trước tiên, ta cần khởi tạo đối tượng lớp GraphNhapXuat, để chương trình biết được nguồn nhập dữ liệu file ghi kết quả đầu ra. Đồng thời lưu các giá trị để khởi tạo đồ thị.

Graph graph(userInput);

 void nhap()

    {

        for (int i = 0; i < n; ++i)

        {

            int dinh, dinh\_ke, trong\_so, so\_canh\_ke;

            cin >> dinh >> so\_canh\_ke;

            for (int j = 0; j < so\_canh\_ke; j++)

            {

                cin >> dinh\_ke >> trong\_so;

                themCanh(dinh, dinh\_ke, trong\_so, j);

            }

        }

    }

    void themCanh(int dinh, int dinh\_ke, int trong\_so, int i)

    {

        adj[dinh].c[i].dinh = dinh\_ke;

        adj[dinh].c[i].trong\_so = trong\_so;

        adj[dinh].so\_canh\_ke++;

    }

Tiếp theo ta cần nhập thông tin của đồ thị khi khởi tạo đối tượng lớp Graph với đối số là đối tượng userInput, các thuộc tính của graph sẽ nhận các giá trị của userInput, khởi tạo danh sách Đồng thời gọi đến phương thức nhap().

Vòng lặp đầu tiên sẽ lặp lớp số lần ứng với số đỉnh của đồ thị, vòng lặp thứ hai lặp số lần ứng với số cạnh kề của đỉnh đó. Phương thức themCanh(int dinh, int dinh\_ke, int trong\_so, int i) nhận các đối số đỉnh, đỉnh kề, trọng số và thứ thự đỉnh kề. adj sẽ gán tương ứng các giá trị. Sau mỗi lần gán sẽ tăng số cạnh kề lên 1.

Dijkstra dijkstra(userInput);

dijkstra.NganNhat(graph);

Sau đó ta khởi tạo đối tượng dijkstra nhận userInput làm đối số, dijkstra cũng sẽ nhận các thông tin từ userInput. Để thực hiện việc tìm đường, ta gọi phương thức NganNhat(graph) nhận đối tượng graph làm đối số, mục đích là để sử dụng phương thức TT\_doithi() để trả về thông tin của đồ thị.

vector<ll> d(n, INF); // mảng lưu đường đi ngắn nhất

        d[batdau] = 0;

        pre[batdau] = batdau;

        Dinh \*adj = graph.TT\_dothi();

        Canh a[1000];

        a[0].trong\_so = 0;

        a[0].dinh = batdau;

Trong phương thức NganNhat(), ta khởi tạo các thông tin cần thiết để bắt đầu vòng lặp. Khởi tạo d: mảng lưu đường đi ngắn nhất từ đích và cho mọi giá trị bằng INF. Sau đó, d[batdau] = 0 khoảng cách từ đỉnh bắt đầu tới chính nó bằng 0, mảng pre được khởi tạo giá trị đầu tiên để lưu đỉnh trước nó. Vì là đỉnh ban đầu nên đỉnh trước đó cũng bằng chính nó.

Thay vì sử dụng hàng đợi ưu tiên, em sử dụng mảng Canh a[1000] để thay thế, nên phần tử đầu tiên của mảng luôn luôn là nhỏ nhất về trọng số. Em cho đỉnh bắt đầu vào mảng và bắt đầu vòng lặp.

while (a[0].dinh != INF)

Khi sử dụng hàng đợi ưu tiên vòng lặp sẽ kết thúc khi không còn phần tử nào trong hàng đợi. Vậy khi em sử dụng Canh, vòng lặp sẽ kết thúc khi phần tử đầu tiên là INF vì a[0] là phần tử bé nhất và giá trị INF là biểu hiện của mảng không có phần tử.

int u = a[0].dinh;

      int kc = a[0].trong\_so;

Khởi tạo 2 biến lưu giá trị của phần tử đầu tiên của mảng, sau đó thực hiện xóa phần từ đầu tiên

int i = 0;

            while (1)

            {

                if (a[i].dinh == INF && a[i].trong\_so == INF)

                    break;

                else

                {

                    a[i].dinh = a[i + 1].dinh;

                    a[i].trong\_so = a[i + 1].trong\_so;

                }

                i++;

            }

Mỗi lần xét 1 giá trị a[0] mới, khởi tạo i = 0. Để xóa phần tử khỏi mảng a, ta thực hiện di chuyển các phần tử về sau 1 chỉ số. Cuối mỗi vòng lặp ta tăng i lên 1 đơn vị để tiếp tục di chuyển. Vòng lặp này sẽ dừng nếu i di chuyển đến giá trị INF, điều này có nghĩa là tại đó không có phần tử.

if (kc > d[u]) continue;

Để giảm số vòng lặp em dùng một điều kiện so sánh trọng số của đỉnh đang xét với mảng lưu đường đi ngắn nhất thay vì tạo thêm 1 mảng lưu các đỉnh đã được truy cập(True hoặc False). Nếu điều kiện sai, em đi qua vòng lặp tiếp theo.

for (int i = 0; i < adj[u].so\_canh\_ke; i++)

            {

                int v = adj[u].c[i].dinh;

                int w = adj[u].c[i].trong\_so;

                if (d[v] > d[u] + w)

                {

                    d[v] = d[u] + w;

                    a[999].dinh = v;

                    a[999].trong\_so = d[v];

                    for (int i = 0; i < 1000; i++)

                    {

                        for (int j = i + 1; j < 1000; j++)

                        {

                            if (a[i].trong\_so > a[j].trong\_so)

                            {

                                Canh c = a[i];

                                a[i] = a[j];

                                a[j] = c;

                            }

                        }

                    }

                    pre[v] = u;

                }

Em kiểm tra từng cạnh của đồ thị, số vòng lặp là số cạnh kề của đỉnh. Khởi tạo 2 biến v và w, gán cho chúng giá trị đỉnh kề và trọng số. So sánh giữa d[v] khoảng cách ngắn nhất từ đỉnh bắt đầu tới v với khoảng cách ngắn nhất từ đỉnh u cộng với trọng số từ u tới v. Nếu đúng, thực hiện việc gán lại d[v] = d[u] + w, đẩy đỉnh v vào mảng a bằng cách gán. Cuối cùng ta thực hiện sắp xếp lại mảng a theo thứ tự tăng dần của trọng số và thực hiện gán đỉnh trước đó của v là u: pre[v] = u.

GraphNhapXuat::duongdi(d, pre, batdau);

static void duongdi(vector<ll> &khoang\_cach, int path[], int batdau, int dich = 0, bool dijkstra = true) {

        int n;

        int i;

        if (dijkstra) {

            n = khoang\_cach.size();

            i = 0;}

        else {

            n = 1;

            i = dich;}

        cout << (dijkstra == true ? "dijkstra: ": "astar: ") << endl;

        while (n > 0) {

            stack<int> ans;

            int j = i;

            while (1) {

                ans.push(j);

                if (j == batdau) break;

                j = path[j];}

            cout << "khoang cach tu " << batdau << " den " << i << " la: " << khoang\_cach[i] << endl;

            while (!ans.empty()) {

                cout << ans.top() << " ";

                ans.pop();

            }

            cout << endl;

            i++;

            n--;}

    }

Để ghi kết quả ra file, ta gọi phương thức duongdi(d, pre, batdau). Các đối số được đưa vào là mảng lưu đường đi ngắn nhất d, mảng lưu đỉnh trước đó pre và đỉnh bắt đầu. Hai đối số phía sau sử dụng giá trị mặc định.

Kiểm tra biến dijkstra xem có phải được gợi từ lớp Dijkstra hay không. Nếu đúng n = khoang\_cach.size(): số lần lặp bằng số đỉnh, i = 0: thuật toán sẽ xây dựng đường đi từ đỉnh bắt đầu đến các đỉnh từ i.

Vòng lặp bắt đầu đến khi số lần lặp bằng 0. Tạo một cấu trúc dữ liệu stack để lưu đường đi. Khi đường đi đã xây dựng thành công, thực hiện ghi ra kết quả. Cuối vòng lặp tăng đỉnh i lên 1 đơn vị để xây dựng đường đi tới đỉnh mới và giảm số lần lặp n xuống 1 đơn vị.

## So sánh với thuật toán A\*

### 3.3.1 Giới thiệu A\*

Theo [2], thuật toán A\* được mô tả bởi Peter Hart, Nils Nisson, Bertram Rapheal vào năm 1968. Nó kết hợp giữa thuật toán tìm kiếm theo chiều rộng (BFS) và thuật toán tìm kiếm theo chi phí thấp nhất (Dijkstra). Nó sử dụng một hàm heuristic để ước lượng chi phí còn lại từ một điểm bất kỳ đến đích. Hàm heuristic này được thiết kế để đưa ra một ước lượng tốt nhất về chi phí thực tế, giúp thuật toán lựa chọn các đường đi tiềm năng tốt nhất để tiến tới đích.

### 3.3.2 Code thuật toán A\*

int heuristic(int from, int to)

    {

        return abs(from - to);

    }

vector<ll> g(n, INF); // đường đi ngắn nhất

        g[batdau] = 0;

        vector<ll> h(n, INF); // heuristic

        h[batdau] = heuristic(batdau, dich);

        priority\_queue<pair<int, int>, vector<pair<int,int>>,greater<pair<int, int>>> Q;

        Q.push({g[batdau] + h[batdau], batdau});

        Dinh \*adj = graph.TT\_dothi();

        while (Q.top().second != dich)

        {

            int u = Q.top().second;

            int f = Q.top().first;

            Q.pop();

            if (f > g[u] + h[u])

                continue;

            for (int i = 0; i < adj[u].so\_canh\_ke; i++)

            {

                int v = adj[u].c[i].dinh;

                int w = adj[u].c[i].trong\_so;

                if (g[v] > g[u] + w)

                {

                    g[v] = g[u] + w;

                    h[v] = heuristic(v, dich);

                    Q.push({g[v] + h[v], v});

                    pre[v] = u;

                }

            }

        }

        GraphNhapXuat::duongdi(g, pre, batdau, dich, false);

Về cơ bản, code của thuật toán A\* khác so với Dijkstra ở chỗ thuật toán có thêm 1 hàm heuristic để ước lượng chi phí từ 1 điểm bất kì đến đích. Hàm heuristic sẽ ảnh hưởng đến tốc độ tìm kiếm của thuật toán. Trong code của em chỉ là hàm heuristic cơ bản và có thể không ảnh hưởng đến tốc độ thuật toán.

h[batdau] = heuristic(batdau, dich);

if (f > g[u] + h[u]) continue;

if (g[v] > g[u] + w)

g[v] = g[u] + w;

h[v] = heuristic(v, dich);

Q.push({g[v] + h[v], v});

Thay vì chỉ so sánh với trọng số như thuật toán Dijkstra, thuật toán A\* sẽ tính toán thêm giá trị hàm heuristic. Thuật toán A\* sẽ dừng lại vòng lặp duyệt qua đỉnh đích và gọi hàm duongdi(g, pre, batdau, dich, false) để ghi kết quả. Các đối số lần lượt là mảng lưu đường đi ngắn nhất g, mảng lưu đỉnh trước đó pre, đỉnh bắt đầu, đỉnh đích và false để phương thức thực hiện ghi ra kết quả đúng với A\*.

### 3.3.3 So sánh

while (a[0].dinh != INF) và

  for (int i = 0; i < adj[u].so\_canh\_ke; i++)

for (int i = 0; i < 1000; i++)

      {

              for (int j = i + 1; j < 1000; j++)

Ở thuật toán Dijkstra, ta duyệt qua tất cả các đỉnh trong đồ thị(n). Và mỗi đỉnh ta xét các cạnh của đỉnh đó(m). Thao tác sắp xếp lại các đỉnh trong mảng a lấy thêm 1000^2 thời gian. Qua đó, ta có độ phức tạp là O((n + 2m)\*1000^2).

  while (Q.top().second != dich)

for (int i = 0; i < adj[u].so\_canh\_ke; i++)

Ở A\* thuật toán cũng duyệt qua các đỉnh của đồ thị và các cạnh ứng với các đỉnh. Nhưng ở A\* sử dụng cấu trúc hàng đợi ưu tiên nên việc lấy ra đỉnh có trọng số thấp nhất là logn. Qua đó, độ phức tạp là O((n + 2m)logn). Dù A\* sử dụng hàm heuristic để tính toán nhưng nếu hàm không tối ưu cũng không thể giảm độ phức tạp.

Qua hai ví dụ trên, em rút ra kết luận: dựa trên phạm vi nghiên cứu và các kết quả đạt được, thuật toán A\* nhanh hơn so với Dijkstra.

# CHƯƠNG 4. KẾT LUẬN

Sau khi đi qua những lí thuyết liên quan và tìm hiểu kĩ càng về thuật toán Dijkstra, em đã thử thay đổi cấu trúc dữ liệu mà thuật toán Dijkstra thường sử dụng là hàng đợi ưu tiên bằng lớp Dinh tự xây dựng. Điều này giúp em hiểu rõ hơn về các cấu trúc dữ liệu trong ngôn ngữ C++.

Sau khi chạy thuật toán, kết quả được thấy đáp ứng được sự chính xác cần thiết của một thuật toán tìm đường đi trong đồ thị dù sử dụng một cách tiếp cận khác và độ phức tạp có tăng lên. Điểm hạn chế tiếp theo khi xây dựng theo cách trên đó là giới hạn không gian lưu trữ, cấu trúc Canh và lớp Dinh cần có một đối số khởi tạo cố định làm giảm độ linh hoạt của thuật toán. Cuối cùng em vẫn chưa hoàn thiện cách khi xây dựng thuật toán theo OOP.

Trong kết quả của nghiên cứu, em đã chỉ ra được kết quả khi so sánh hai thuật toán tìm đường đi ngắn nhất. Kết quả vẫn chỉ là dựa trên code, em không đủ khả năng để có thể chứng minh một cách chi tiết hơn qua thời gian chạy thực tế.

Đề tài nghiên cứu đã hoàn thành mục tiêu ban đầu đề ra. Em đã nắm được cấu trúc dữ liệu thuật toán sử dụng và cách mà thuật toán Dijkstra thực hiện các bước để tìm kiếm đường đi ngắn nhất trong đồ thị. Điều này sẽ giúp em sẽ là một cơ sở vững chắc để em tiếp tục tìm hiểu thêm các vấn đề về đường đi trong định tuyến cũng như tiếp cận các thuật toán tìm kiếm khác. Sau nghiên cứu lần này, có thể em sẽ tìm những phương pháp mới để có thể cải thiện thuật toán một cách hoàn chỉnh hơn và xây dựng một ứng dụng như Google Map để áp dụng thuật toán. Do hạn chế về mặt thời gian và kiến thức tích lũy chưa đủ nhiều nên bài nghiên cứu của em sẽ có những sai sót nhất định. Trong thời gian tới em sẽ cố gắng bổ sung kiến thức, mở rộng lĩnh vực để có thể có những nghiên cứu chất lượng và chính xác hơn

# TÀI LIỆU THAM KHẢO

|  |  |
| --- | --- |
| [1] | M. Barbehenn, "A Note on the Complexity," *IEEE TRANSACTIONS ON COMPUTERS,* p. 1, 1998. |
| [2] | Ade Candra, Mohammad Andri Budiman, Kevin Hartanto, "Dijkstra's and A-Star in Finding the Shortest Path: a," *International Conference on Data Science, Artificial Intelligence, and Business Analytics,* 2020. |