

KHOA KỸ THUẬT VÀ CÔNG NGHỆ  
BỘ MÔN CÔNG NGHỆ THÔNG TIN



THỰC TẬP ĐỒ ÁN CƠ SỞ NGÀNH  
HỌC KỲ I, NĂM HỌC 2024-2025

**Viết chương trình mô phỏng giải thuật  
Floyd Warshall**

*Giảng viên hướng dẫn:*  
TS. Trần Hoàng Nam

*Sinh viên thực hiện:*  
Họ tên: Phạm Minh Nhật  
MSSV: 110122130  
Lớp : DA22TTB

Trà Vinh, tháng 12 năm 2024

[illegible]

**Giáo viên hướng dẫn**  
(Ký tên và ghi rõ họ tên)

[illegible]

**Thành viên hội đồng**  
(Ký tên và ghi rõ họ tên)

## LỜI CẢM ƠN

Báo cáo đồ án cơ sở ngành với chủ đề "Viết chương trình mô phỏng thuật toán Floyd Warshall" là kết quả của sự nỗ lực không ngừng của bản thân tôi và sự giúp đỡ, động viên của thầy cô, bạn bè. Thông qua báo cáo này, tôi xin gửi lời cảm ơn đến những người đã giúp đỡ tôi trong quá trình nghiên cứu và hoàn thành đồ án. Tôi xin bày tỏ sự kính trọng và biết ơn sâu sắc tới thầy Trầm Hoàng Nam vì đã trực tiếp hướng dẫn và cung cấp những thông tin cần thiết cho đồ án này. Tôi xin chân thành cảm ơn lãnh đạo Trường Đại học Trà Vinh, Khoa Kỹ thuật và Công nghệ, bộ môn Công nghệ thông tin đã tạo điều kiện để tôi hoàn thành thành công đồ án. Trong quá trình thực hiện đồ án, do kiến thức còn hạn chế nên còn nhiều thiếu sót mong các thầy cô có thể bổ sung thêm để đồ án hoàn thiện hơn.

Xin chân thành cảm ơn!

## MỤC LỤC

CHƯƠNG 1: TỔNG QUAN .....	9
1. Cơ sở lý thuyết.....	9
1.1 Đồ thị.....	9
1.1.1 Định nghĩa 1 .....	9
1.1.2 Định nghĩa 2 .....	9
1.1.3 Định nghĩa 3 .....	9
1.2 Đồ thị có trọng số .....	9
1.3 Chu trình âm .....	10
CHƯƠNG 2: NGHIÊN CỨU LÝ THUYẾT .....	11
2.1 Bài toán Floyd-Warshall .....	12
2.1.1 Định nghĩa .....	12
2.2 Giả thuyết khoa học .....	12
2.3 Một số định lý của thuật toán Floyd-Warshall .....	12
2.3.1 Tính chất đường đi ngắn nhất .....	12
2.3.2 Hội tụ .....	12
2.3.3 Tính chính xác .....	12
2.3.4 Bất biến .....	12
2.4. Giải thuật Floyd-Warshall .....	12
2.4.1. Cách tính các ma trận trong giải thuật Floyd-Warshall .....	12
2.4.2. Xác định đường đi ngắn nhất .....	13
2.4.3. Ví dụ minh họa .....	13
2.4.4. Biểu diễn các bước trên giao diện .....	15
CHƯƠNG 3: HIỆN THỰC HÓA NGHIÊN CỨU .....	16

3. Cài đặt chương trình .....	16
3.1 Khai báo các thư viện cần thiết .....	16
3.2 Khởi tạo đồ thị .....	16
3.2.1 Hàm thực hiện giải thuật Floyd-Warshall .....	17
3.2.2 Xác định các đỉnh kế tiếp trên đường đi trực tiếp từ đỉnh đến đích .....	17
3.2.3 Tính toán giải thuật Floyd-Warshall .....	18
3.2.4 Tái dựng đường đi ngắn nhất .....	19
3.2.5 Hàm tạo đồ thị mẫu .....	19
3.2.6 Hàm cập nhật đồ thị khi thay đổi số đỉnh .....	20
3.2.7 Tạo và vẽ đồ thị mẫu .....	20
3.2.8 Hàm cập nhật trọng số của cạnh khi chỉnh sửa .....	21
3.2.9 Cập nhật, vẽ ma trận và hiển thị đồ thị .....	22
3.2.10 Cập nhật khi thay đổi điểm bắt đầu, kết thúc .....	23
3.2.11 Tìm đường đi ngắn nhất .....	24
3.2.12 Tạo khung và hiển thị kết quả .....	25
3.2.13 Khởi tạo .....	28
3.2.14 Chạy chương trình .....	28
CHƯƠNG 4: KẾT QUẢ NGHIÊN CỨU .....	30
4.1 Giao diện người dùng .....	30
4.1.1 Giao diện chính .....	30
4.1.2 Khung nhập thủ công khoảng cách giữa các đỉnh trong ma trận .....	30
4.1.3 Vị trí nhập đỉnh, điểm bắt đầu và điểm đích .....	30
4.2 Cách sử dụng .....	31
4.2.1 Thêm đỉnh cho đồ thị .....	31

---

4.2.2 Thay đổi độ dài giữa các đỉnh của đồ thị .....	32
4.2.3 Chọn điểm bắt đầu và kết thúc của đồ thị .....	33
4.3 Hiệu năng .....	35
4.4 Trải nghiệm người dùng (UX) .....	36
4.5 Giao diện chức năng .....	36
4.6 Minh họa giao diện .....	37
4.7 Ưu điểm và nhược điểm .....	37
CHƯƠNG 5: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN .....	38
5.1 Kết Luận .....	38
5.2 Hướng phát triển .....	38
DANH MỤC TÀI LIỆU THAM KHẢO .....	39
PHỤ LỤC .....	40

## DANH MỤC HÌNH ẢNH - BẢNG BIỂU

Hình 1.1 Đồ thị có trọng số .....	10
Hình 1.2. Ví dụ chu trình âm .....	11
CHƯƠNG 2: NGHIÊN CỨU LÝ THUYẾT .....	12
Hình 2.1. Ma trận ví dụ 1 .....	14
Hình 2.2. Ma trận ví dụ 2 .....	14
Hình 2.3. Ma trận ví dụ 3 .....	14
Hình 2.4. Ma trận ví dụ 4 .....	15
Hình 3.1. Khai báo thư viện .....	16
Hình 3.2. Giá trị vô cực inf .....	16
Hình 3.3 Hàm thực hiện giải thuật Floyd-Warshall .....	17
Hình 3.4. Xác định các đỉnh .....	17
Hình 3.5. Tính toán giải thuật .....	18
Hình 3.6. Tái dựng đường đi ngắn nhất .....	19
Hình 3.7. Hàm tạo đồ thị mẫu .....	19
Hình 3.8. Hàm cập nhật khi thay đổi số đỉnh .....	20
Hình 3.9. Tạo và vẽ đồ thị mẫu .....	20
Hình 3.10. Hàm cập nhật trọng số của cạnh .....	21
Hình 3.11. Cập nhật, vẽ ma trận và hiển thị đồ thị .....	22
Hình 3.12. Cập nhật khi thay đổi điểm bắt đầu, kết thúc .....	23
Hình 3.13. Tìm đường đi ngắn nhất .....	24
Hình 3.14. Tạo khung và hiển kết quả .....	25
Hình 3.15. Khởi tạo đồ thị .....	28
Hình 3.16. Chạy chương trình .....	28
Hình 4.1. Màn hình giao diện .....	30
Hình 4.2. Khung nhập ma trận .....	30
Hình 4.3. Vị trí nhập đỉnh, điểm đầu và điểm đích .....	30
Hình 4.4. Nhập số đỉnh đồ thị .....	31
Hình 4.5. Đồ thị ví dụ 1 .....	31
Hình 4.6. Nút tạo đồ thị .....	31
Hình 4.7. Đồ thị kết quả 1 .....	32
Hình 4.8. Đồ thị ví dụ 2 .....	33
Hình 4.9. Đồ thị kết quả 2 .....	33
Hình 4.10. Đồ thị ví dụ 3 .....	34
Hình 4.11. Khung nhập điểm bắt đầu và kết thúc .....	34
Hình 4.12. Đồ thị kết quả 3 .....	35



## TÓM TẮT ĐỒ ÁN CƠ SỞ NGÀNH

Đồ án "Viết chương trình mô phỏng giải thuật Floyd-Warshall" nhằm cung cấp cho sinh viên cái nhìn tổng quan và sâu sát về một trong những giải thuật quan trọng của lý thuyết đồ thị trong ngành khoa học máy tính.

Trong bài toán này, tôi đã nghiên cứu các nguyên lý hoạt động của giải thuật Floyd-Warshall, áp dụng các nguyên tắc để xây dựng chương trình mô phỏng nhằm tìm đường đi ngắn nhất trong đồ thị có trọng số. Để tối ưu hoá việc xử lý, chương trình được cài đặt bằng Python và sử dụng giao diện tkinter giúp người dùng dễ dàng tương tác.

Kết quả đạt được bao gồm:

- Mô phỏng chính xác hoạt động của giải thuật Floyd-Warshall.
- Minh hoạ giao diện đồ thị và đường đi ngắn nhất.

Các hướng tiếp cận bao gồm nghiên cứu về lý thuyết đồ thị, xây dựng thuật toán và mô phỏng trên giao diện trực quan.

## MỞ ĐẦU

- **Lý do chọn đề tài**

Trong bối cảnh phát triển mạnh mẽ của công nghệ thông tin, việc giải quyết các bài toán tối ưu trên đồ thị ngày càng trở nên quan trọng và phổ biến trong nhiều lĩnh vực như giao thông, mạng máy tính, logistics và trí tuệ nhân tạo. Trong đó, bài toán tìm đường đi ngắn nhất giữa các cặp đỉnh trong đồ thị đóng vai trò quan trọng, đặc biệt trong việc tối ưu hóa chi phí, thời gian và tài nguyên.

Giải thuật **Floyd-Warshall**, với khả năng tính toán hiệu quả đường đi ngắn nhất giữa tất cả các cặp đỉnh trong một đồ thị có trọng số, là một trong những giải thuật quan trọng nhất trong lý thuyết đồ thị. Để hiểu rõ hơn cách thức hoạt động cũng như ứng dụng thực tiễn của giải thuật này, việc xây dựng một chương trình mô phỏng trực quan là cần thiết.

- **Mục đích nghiên cứu**

- Hiểu rõ lý thuyết và cơ chế hoạt động của giải thuật Floyd-Warshall thông qua việc lập trình và mô phỏng.
- Xây dựng một chương trình trực quan, thân thiện, giúp người dùng có thể tương tác và trải nghiệm cách giải thuật hoạt động trên các đồ thị cụ thể.
- Ứng dụng **Python** và **Tkinter** để minh họa đồ thị, thể hiện đường đi ngắn nhất và quá trình tính toán của giải thuật.
- Cung cấp một công cụ hỗ trợ giảng dạy, học tập và nghiên cứu giải thuật Floyd-Warshall.

- **Đối tượng nghiên cứu:**

- Giải thuật Floyd-Warshall và các ứng dụng liên quan đến bài toán tìm đường đi ngắn nhất trong đồ thị.
- Các thành phần chính của ngôn ngữ Python, thư viện Tkinter và các công cụ hỗ trợ vẽ đồ thị.

- **Phạm vi nghiên cứu:**

- Xây dựng chương trình xử lý đồ thị, với khả năng nhập ma trận trọng số tùy chỉnh.
- Giao diện đồ họa cho phép người dùng chỉnh sửa đồ thị, nhập điểm bắt đầu và kết thúc để tìm đường đi ngắn nhất.
- Ứng dụng chỉ tập trung vào đồ thị có trọng số (có thể dương hoặc âm) nhưng không có chu trình âm.

## CHƯƠNG 1: TỔNG QUAN

### 1. Cơ sở lý thuyết

#### 1.1. Đồ thị

Đồ thị là một cấu trúc rời rạc bao gồm các đỉnh và các cạnh nối các đỉnh này. Chúng ta phân biệt các loại đồ thị khác nhau bởi kiểu và số lượng cạnh nối hai đỉnh nào đó của đồ thị.[1]

##### 1.1.1. Định nghĩa 1

Đồ thị vô hướng  $G = (V, E)$  bao gồm  $V$  là tập các đỉnh, và  $E$  là tập các cặp không có thứ tự gồm hai phần tử khác nhau của  $V$  gọi là các cạnh.[1]

##### 1.1.2. Định nghĩa 2

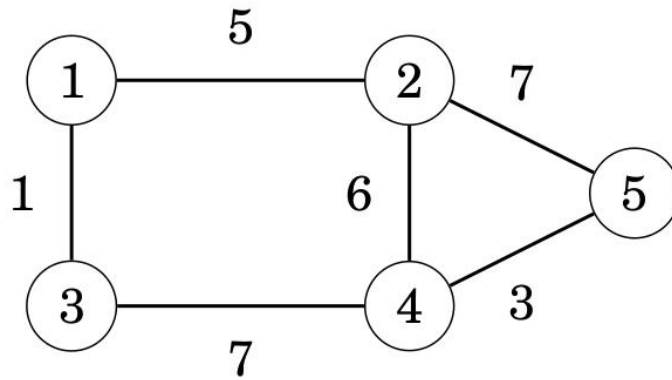
Hai đỉnh được gọi là liên kề nhau nếu có cạnh nối hai đỉnh đó với nhau. [1]  
Cạnh nối 2 đỉnh lại với nhau được gọi là cạnh liên thuộc. Hai cạnh được gọi là kề nhau nếu giữa chúng có đỉnh chung. Nếu  $e = (v, v)$  là một cạnh của đồ thị thì  $e$  được gọi là một khuyên.[1]

##### 1.1.3. Định nghĩa 3

- Nếu mỗi cạnh  $e = (u, v)$  là không phân biệt thứ tự của các đỉnh  $u$  và  $v$  (từ  $u$  tới  $v$  hoặc từ  $v$  tới  $u$  không kể hướng) thì ta nói đồ thị  $G = (V, E)$  là đồ thị vô hướng.[1]
- Nếu mỗi cạnh  $e = (u, v)$  có phân biệt thứ tự của các đỉnh  $u$  và  $v$  (tức là từ  $u$  tới  $v$  khác từ  $v$  tới  $u$ ) thì ta nói đồ thị  $G = (V, E)$  là đồ thị có hướng. Cạnh của đồ thị có hướng được gọi là cung.[1]

#### 1.2. Đồ thị có trọng số

- Đồ thị có trọng số (weighted) là đồ thị có các cạnh được gán một trọng số. Các trọng số thường thể hiện chiều dài của cạnh.[2]
- Ví dụ, đồ thị sau là đồ thị có trọng số:

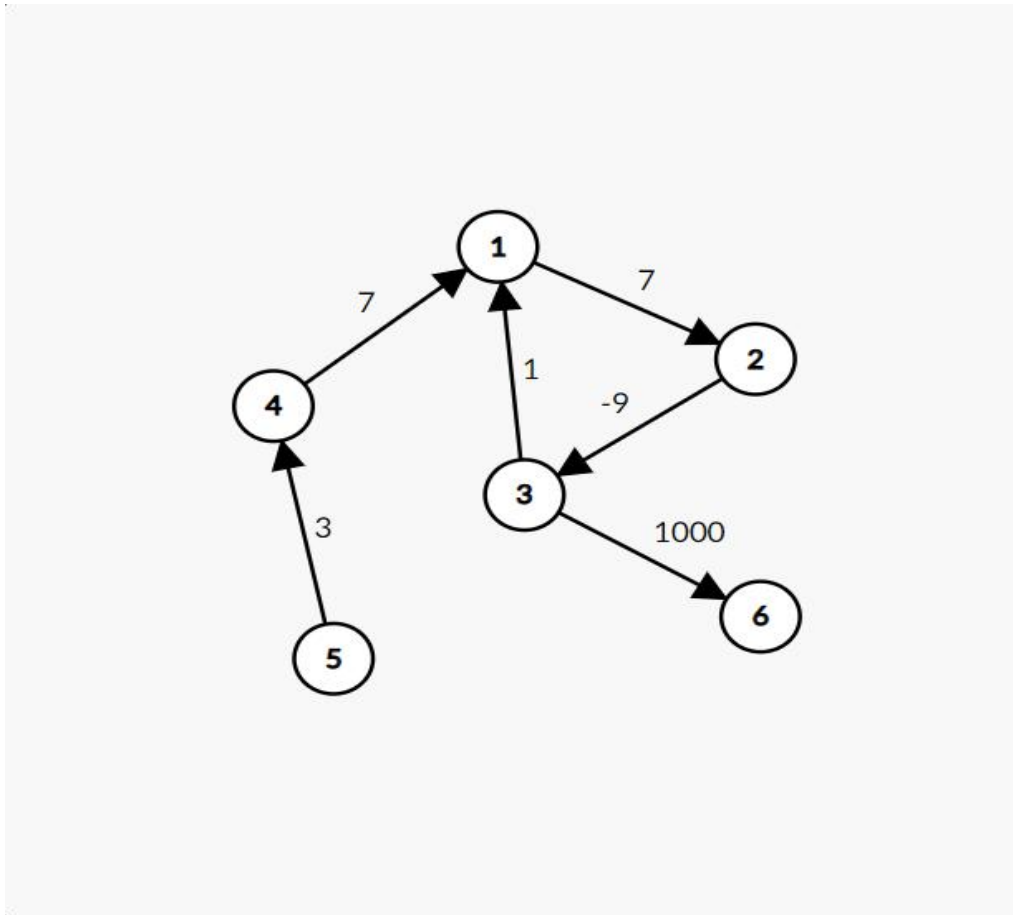


Hình 1.1 Đồ thị có trọng số

- Chiều dài của con đường trong đồ thị trọng số là tổng trọng số của các cạnh trên đường đi. Ví dụ, trong đồ thị trên, chiều dài của đường đi  $1 \rightarrow 2 \rightarrow 5$  là 12, và chiều dài của đường đi  $1 \rightarrow 3 \rightarrow 4 \rightarrow 5$  là 11. Đường đi sau là đường đi ngắn nhất từ đỉnh 1 đến đỉnh 5.

### 1.3. Chu trình âm

- Chu trình âm là một chu trình trong đó tổng trọng số các cạnh là số âm. [4]
- Ví dụ: trong bài toán trên, ta có một chu trình âm  $1-2-3-1$  có tổng trọng số là  $7-9+1=-1$ . Nếu trên đường đi từ  $u$  đến  $v$  chứa chu trình âm thì độ dài đường đi ngắn nhất từ  $u$  đến  $v$  sẽ là âm vô cực. Vì vậy nên một số cặp đỉnh không tồn tại đường đi ngắn nhất do có chu trình âm trên đường đi giữa chúng (chỉ tồn tại đường đi có độ dài âm vô cực). Ở đồ thị này, đường đi ngắn nhất từ 4 đến 6 sẽ có cách đi là vô hạn lần qua chu trình âm đã nhắc đến, sau đó mới đi đến 6. Như vậy không có đường đi ngắn nhất.



Hình 1.2. Ví dụ chu trình âm

## CHƯƠNG 2: NGHIÊN CỨU LÝ THUYẾT

### 2.1. Bài toán Floyd-Warshall

#### 2.1.1. Định nghĩa

Thuật toán Floyd-Warshall còn được gọi là thuật toán Floyd được Robert Floyd tìm ra năm 1962 là thuật toán để tìm đường đi ngắn nhất giữa mọi cặp đỉnh. Floyd hoạt động được trên đồ thị có hướng, có thể có trọng số âm, tuy nhiên không có chu trình âm.[2]

### 2.2. Giả thuyết khoa học

Thuật toán Floyd-Warshall khẳng định rằng thuật toán này có thể tìm ra đường đi ngắn nhất giữa tất cả các cặp đỉnh trong một đồ thị có trọng số, miễn là không có chu trình âm. Điều này làm cho thuật toán trở thành một công cụ hữu ích trong nhiều lĩnh vực khác nhau, từ mạng máy tính đến hệ thống giao thông

### 2.3. Một số định lý của thuật toán Floyd-Warshall

#### 2.3.1. Tính chất đường đi ngắn nhất

Đường đi ngắn nhất của Floyd-Warshall dựa trên nguyên tắc rằng mọi đường đi có thể được chia nhỏ, và việc đi qua các đỉnh trung gian  $k$  sẽ cải thiện hoặc giữ nguyên khoảng cách ngắn nhất từ  $i$  đến  $j$ . Công thức đệ quy đảm bảo tính chính xác và hiệu quả của thuật toán.

#### 2.3.2. Hội tụ

Thuật toán hội tụ sau  $n$  bước (với  $n$  là số đỉnh)

#### 2.3.3. Tính chính xác

Kết quả đúng nếu đồ thị không có chu trình âm.

#### 2.3.4. Bất biến

Thuật toán sẽ luôn kết thúc sau  $n$  bước.

### 2.4. Giải thuật Floyd-Warshall

#### 2.4.1. Cách tính các ma trận trong giải thuật Floyd-Warshall

- Giải thuật Floyd-Warshall sử dụng hai ma trận chính:
  - **Ma trận khoảng cách (dist):** Lưu trữ khoảng cách ngắn nhất giữa các cặp đỉnh.
  - **Ma trận truy vết (next\_node):** Lưu thông tin về đỉnh tiếp theo trên đường đi ngắn nhất từ đỉnh i đến j.
- Quy trình tính toán ma trận:
  - **Khởi tạo ma trận dist và next\_node:**
    - $dist[i][j]$ : Bằng trọng số cạnh  $i \rightarrow j$  nếu có cạnh, và bằng vô cực (INF) nếu không có cạnh nối.
    - $next\_node[i][j]$ : Ban đầu, nếu tồn tại cạnh  $i \rightarrow j$ , giá trị sẽ là j. Nếu không, gán giá trị None.
  - **Cập nhật ma trận với đỉnh trung gian k:**
    - Dùng ba vòng lặp lồng nhau để duyệt qua tất cả các cặp đỉnh i,j kiểm tra xem việc đi qua đỉnh trung gian k có rút ngắn được khoảng cách hay không:
    - Nếu  $dist[i][k] + dist[k][j] < dist[i][j]$ :  
$$dist[i][j] = dist[i][k] + dist[k][j]$$
$$next\_node[i][j] = next\_node[i][k]$$

#### 2.4.2. Xác định đường đi ngắn nhất

- Sau khi tính toán, sử dụng ma trận next\_node để truy vết đường đi ngắn nhất giữa hai đỉnh i và j.
- Quy trình tái dựng đường đi:
  - đầu từ đỉnh i, dùng next\_node[i][j] để tìm đỉnh kế tiếp.
  - Lặp lại quá trình đến khi đạt đến đỉnh j.

#### 2.4.3. Ví dụ minh họa

Giả sử đồ thị có 5 đỉnh với ma trận trọng số ban đầu:



$$\begin{bmatrix} 0 & 4 & \infty & 7 & \infty \\ 4 & 0 & 3 & \infty & 6 \\ \infty & 3 & 0 & 2 & \infty \\ 7 & \infty & 2 & 0 & 5 \\ \infty & 6 & \infty & 5 & 0 \end{bmatrix}$$

Hình 2.1. Ma trận ví dụ 1

1. **Khởi tạo ma trận dist và next\_node:**

- dist:

$$\begin{bmatrix} 0 & 4 & \infty & 7 & \infty \\ 4 & 0 & 3 & \infty & 6 \\ \infty & 3 & 0 & 2 & \infty \\ 7 & \infty & 2 & 0 & 5 \\ \infty & 6 & \infty & 5 & 0 \end{bmatrix}$$

Hình 2.2. Ma trận ví dụ 2

- next\_node:

$$\begin{bmatrix} None & 1 & None & 3 & None \\ 0 & None & 2 & None & 4 \\ None & 1 & None & 3 & None \\ 0 & None & 2 & None & 4 \\ None & 1 & None & 3 & None \end{bmatrix}$$

Hình 2.3. Ma trận ví dụ 3

**Cập nhật qua các đỉnh trung gian:** Ví dụ: Sử dụng đỉnh trung gian k=2:

- $dist[0][3] = \min(dist[0][3], dist[0][2] + dist[2][3])$
- Nếu có cải thiện, cập nhật giá trị dist và next\_node.

**Kết quả cuối cùng (sau khi cập nhật):**

- dist:

$$\begin{bmatrix} 0 & 4 & 7 & 7 & 11 \\ 4 & 0 & 3 & 5 & 6 \\ 7 & 3 & 0 & 2 & 7 \\ 7 & 5 & 2 & 0 & 5 \\ 11 & 6 & 7 & 5 & 0 \end{bmatrix}$$

Hình 2.4. Ma trận ví dụ 4

- Đường đi ngắn nhất từ đỉnh 0 đến đỉnh 4:
  - Truy vết từ ma trận next\_node:  $0 \rightarrow 1 \rightarrow 4$
  - Khoảng cách ngắn nhất:  $\text{dist}[0][4]=11$

**2.4.4. Biểu diễn các bước trên giao diện**

- Hiển thị ma trận trọng số ban đầu.
- Tính toán và hiển thị ma trận dist và next\_node theo từng bước.
- Vẽ đồ thị trực quan với đường đi ngắn nhất được tô đỏ.
- Cho phép người dùng nhập hoặc chỉnh sửa đồ thị để kiểm tra các kết quả khác.

## CHƯƠNG 3: HIỆN THỰC HÓA NGHIÊN CỨU

### 3. Cài đặt chương trình

#### 3.1. Khai báo các thư viện cần thiết

```
1 import tkinter as tk
2 from tkinter import ttk
3 import numpy as np
4 import networkx as nx
5 from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
6 import matplotlib.pyplot as plt
```

Hình 3.1. Khai báo thư viện

Trong đó:

- Thư viện tkinter hỗ trợ tạo giao diện người dùng.
- Ttk là một thành phần của thư viện tkinter
- Thư viện numpy xử lý mảng và thực hiện các phép toán số học.
- Thư viện networkx hỗ trợ tạo, thao tác với đồ thị(graph), thêm, xóa các đỉnh cạnh tính toán đồ thị và thực hiện các thuật toán đồ thị như tìm đường đi ngắn nhất.
- Matplotlib là thư viện vẽ đồ thị cho Python, tạo ra các biểu đồ và hình ảnh từ dữ liệu. FigureCanvasTkAgg hiển thị các biểu đồ và hình ảnh trực tiếp trong giao diện người dùng của ứng dụng Tkinter.
- Pyplot là một module trong Matplotlib cung cấp một giao diện để vẽ đồ thị, tạo các biểu đồ, đồ thị, và hình ảnh một cách dễ dàng.

#### 3.2. Khởi tạo đồ thị

```
# Giá trị vô cực (đại diện cho đường không kết nối)
INF = float('inf')
```

Hình 3.2. Giá trị vô cực inf

INF : Đại diện cho trọng số của các cạnh không tồn tại trong đồ thị, cho phép thuật toán nhận biết rằng không có đường đi giữa hai đỉnh đó

### 3.2.1. Hàm thực hiện giải thuật Floyd-Warshall

```
# Hàm thực hiện giải thuật Floyd-Warshall
def floyd_warshall(graph):
    n = len(graph)
    dist = np.copy(graph)
    next_node = np.full((n, n), None)
```

Hình 3.3 Hàm thực hiện giải thuật Floyd-Warshall

- `n = len(graph)`: Xác định số lượng đỉnh trong đồ thị (số hàng hoặc cột của ma trận kề).
- `dist = np.copy(graph)`: Sao chép ma trận `graph` vào ma trận `dist` để lưu trữ khoảng cách ngắn nhất giữa các đỉnh. Ban đầu, ma trận `dist` chính là trọng số trực tiếp của các cạnh trong đồ thị.
- `next_node = np.full((n, n), None)`: Tạo ma trận `next_node` với kích thước  $n \times n$ , ban đầu tất cả các phần tử được gán là `None`. Ma trận này sẽ lưu trữ thông tin về đỉnh kế tiếp trên đường đi ngắn nhất từ đỉnh  $i$  đến đỉnh  $j$ .

### 3.2.2. Xác định các đỉnh kế tiếp trên đường đi trực tiếp từ đỉnh đến đích

```
# Khởi tạo next_node cho các cạnh trực tiếp
for i in range(n):
    for j in range(n):
        if graph[i][j] != INF and i != j:
            next_node[i][j] = j
```

Hình 3.4. Xác định các đỉnh

- `Next_node[i][j]` là **đỉnh kế tiếp** mà bạn cần đi qua trên **đường đi ngắn nhất** từ đỉnh  $i$  đến đỉnh  $j$ .
- Ma trận `next_node`: Ma trận này rất hữu ích để **truy vết toàn bộ đường đi ngắn nhất** (không chỉ khoảng cách) giữa hai đỉnh bất kỳ. Nếu tồn tại cạnh trực tiếp từ  $i$  đến  $j$  (trọng số khác `INF`), thì `next_node[i][j]` được gán là  $j$  (đỉnh kế tiếp trên đường từ  $i$  đến  $j$  là chính  $j$ ). Điều kiện `i != j` đảm bảo rằng không xét đường đi từ một đỉnh đến chính nó.

### 3.2.3. Tính toán giải thuật Floyd-Warshall

```
# Thực hiện giải thuật Floyd-Warshall
for k in range(n):
    for i in range(n):
        for j in range(n):
            if dist[i][k] + dist[k][j] < dist[i][j]:
                dist[i][j] = dist[i][k] + dist[k][j]
                next_node[i][j] = next_node[i][k]

return dist, next_node
```

---

Hình 3.5. Tính toán giải thuật

#### Ý nghĩa của 3 vòng lặp:

- **Vòng ngoài (for k in range(n)):**

Đỉnh k được sử dụng làm trung gian để kiểm tra xem liệu có thể cải thiện đường đi giữa hai đỉnh i và j thông qua k.

- **Vòng giữa (for i in range(n)):**

Xét tất cả các đỉnh i làm đỉnh xuất phát.

- **Vòng trong (for j in range(n)):**

Xét tất cả các đỉnh j làm đỉnh đích.

- **Kiểm tra**  $\text{dist}[i][k] + \text{dist}[k][j] < \text{dist}[i][j]$ : Nếu đường đi từ i đến j thông qua k có tổng trọng số nhỏ hơn đường đi hiện tại từ i đến j, thì ta cập nhật khoảng cách ngắn nhất.
- **dist**: Ma trận khoảng cách ngắn nhất giữa tất cả các cặp đỉnh.
- **next\_node**: Ma trận chứa thông tin về đỉnh kế tiếp trên đường đi ngắn nhất giữa tất cả các cặp đỉnh.

### 3.2.4. Tái dựng đường đi ngắn nhất

```
# Hàm tái dựng đường đi ngắn nhất
def reconstruct_path(next_node, start, end):
    if next_node[start][end] is None:
        return []
    path = [start]
    while start != end:
        start = next_node[start][end]
        path.append(start)
    return path
```

Hình 3.6. Tái dựng đường đi ngắn nhất

- Hàm này **tái dựng đường đi ngắn nhất** giữa hai đỉnh start (đỉnh bắt đầu) và end (đỉnh kết thúc), dựa vào ma trận next\_node được tạo ra từ giải thuật Floyd-Warshall.

### 3.2.5. Hàm tạo đồ thị mẫu

```
# Hàm tạo đồ thị mẫu
def create_sample_graph(num_vertices):
    graph = np.full((num_vertices, num_vertices), INF)
    np.fill_diagonal(graph, 0) # Khoảng cách từ đỉnh đến chính nó là 0
    for i in range(num_vertices):
        for j in range(num_vertices):
            if i != j and np.random.rand() > 0.5: # Xác suất tạo cạnh
                graph[i][j] = np.random.randint(1, 10) # Trọng số ngẫu nhiên từ 1 đến 9
    return graph
```

Hình 3.7. Hàm tạo đồ thị mẫu

- Tạo một đồ thị mẫu ngẫu nhiên với num\_vertices đỉnh, sử dụng ma trận trọng số để biểu diễn đồ thị.
- Tạo một ma trận graph kích thước num\_vertices x num\_vertices, ban đầu gán tất cả các trọng số bằng INF (vô cực) để biểu diễn rằng các đỉnh ban đầu không được kết nối.
- Gán giá trị đường chéo của ma trận (từ đỉnh đến chính nó) bằng 0, vì khoảng cách từ một đỉnh đến chính nó luôn là 0.

### 3.2.6. Hàm cập nhật đồ thị khi thay đổi số đỉnh

```

# Hàm cập nhật đồ thị khi thay đổi số đỉnh
def update_graph_vertices():
    global graph, dist, next_node
    try:
        num_vertices = int(vertex_entry.get())
        if num_vertices < 2 or num_vertices > 10: # Giới hạn số lượng đỉnh
            raise ValueError("Số đỉnh phải từ 2 đến 10.")
    except ValueError:
        result_label.config(text="Vui lòng nhập số đỉnh hợp lệ (2-10).")
    return

```

Hình 3.8. Hàm cập nhật khi thay đổi số đỉnh

- Khai báo các biến toàn cục để cập nhật ma trận đồ thị (graph), ma trận khoảng cách (dist), và ma trận truy vết (next\_node).
- Lấy số lượng đỉnh từ giao diện nhập liệu (vertex\_entry.get()).
- Nếu số đỉnh nằm ngoài phạm vi [2, 10], ném ngoại lệ để hiển thị thông báo lỗi.
- Nếu người dùng nhập dữ liệu không hợp lệ (không phải số nguyên hoặc ngoài phạm vi), hiển thị thông báo lỗi trên giao diện.

### 3.2.7. Tạo và vẽ đồ thị mẫu

```

# Tạo đồ thị mẫu mới
graph = create_sample_graph(num_vertices)
dist, next_node = floyd_warshall(graph)

# Reset các ô nhập liệu và kết quả
start_entry.delete(0, tk.END)
start_entry.insert(0, "0")
end_entry.delete(0, tk.END)
end_entry.insert(0, str(num_vertices - 1))
result_label.config(text="")

# Hiển thị ma trận trọng số để chỉnh sửa
show_weight_matrix(graph)

# Vẽ đồ thị mới
draw_graph(canvas_frame, graph)

```

Hình 3.9. Tạo và vẽ đồ thị mẫu



- `create_sample_graph(num_vertices)`: Tạo đồ thị mới với số đỉnh là `num_vertices`, trọng số các cạnh ngẫu nhiên.
- `floyd_warshall(graph)`: Thực hiện giải thuật Floyd-Warshall trên đồ thị vừa tạo để tính toán ma trận khoảng cách `dist` và ma trận truy vết `next_node`.
- `start_entry` và `end_entry`: Đặt lại giá trị mặc định cho điểm bắt đầu và điểm kết thúc.
  - Điểm bắt đầu mặc định: 0.
  - Điểm kết thúc mặc định: `num_vertices - 1` (đỉnh cuối cùng trong đồ thị).
- `result_label`: Xóa thông báo kết quả trước đó.

`show_weight_matrix(graph)`

- Hàm này hiển thị ma trận trọng số của đồ thị để người dùng có thể chỉnh sửa trực tiếp.

### 3.2.8. Hàm cập nhật trọng số của cạnh khi chỉnh sửa

```
# Hàm cập nhật trọng số của cạnh khi chỉnh sửa
def update_edge_weight(i, j, widget):
    try:
        value = widget.get()
        if value == "":
            graph[i][j] = INF
        else:
            weight = int(value)
            if weight < 0:
                raise ValueError("Trọng số phải >= 0.")
            graph[i][j] = weight
    except ValueError:
        widget.delete(0, tk.END)
        widget.insert(0, "" if graph[i][j] == INF else int(graph[i][j]))
    update_floyd_warshall()
```

Hình 3.10. Hàm cập nhật trọng số của cạnh

- **Kiểm tra giá trị nhập vào:**
  - Nếu giá trị trống (""), đặt trọng số là INF.
  - Nếu giá trị là số nguyên, kiểm tra xem có nhỏ hơn 0 không.



- **Cập nhật ma trận đồ thị** `graph[i][j]`: Ghi nhận giá trị mới.
- `update_floyd_warshall()`: Cập nhật lại ma trận khoảng cách `dist` và ma trận truy vết `next_node` với đồ thị mới.

### 3.2.9. Cập nhật, vẽ ma trận và hiển thị đồ thị

```
# Hàm vẽ đồ thị
def draw_graph(canvas_frame, graph, shortest_path=None):
    G = nx.DiGraph()
    n = len(graph)

    # Thêm các cạnh và trọng số vào đồ thị
    for i in range(n):
        for j in range(n):
            if graph[i][j] != INF and i != j:
                G.add_edge(i, j, weight=graph[i][j])

    pos = nx.spring_layout(G) # Tính toán vị trí các đỉnh
    edge_labels = nx.get_edge_attributes(G, 'weight')

    # Tạo đồ thị với Matplotlib
    fig, ax = plt.subplots(figsize=(5, 5))
    nx.draw(G, pos, with_labels=True, node_color='lightblue', node_size=3000, font_size=10, ax=ax)
    nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, ax=ax)

    # Vẽ đường đi ngắn nhất (nếu có)
    if shortest_path:
        edges = [(shortest_path[i], shortest_path[i + 1]) for i in range(len(shortest_path) - 1)]
        nx.draw_networkx_edges(G, pos, edgelist=edges, edge_color='red', width=2, ax=ax)

    # Hiển thị đồ thị trên Tkinter Canvas
    for widget in canvas_frame.winfo_children():
        widget.destroy() # Xóa đồ thị cũ
    canvas = FigureCanvasTkAgg(fig, master=canvas_frame)
    canvas.draw()
    canvas.get_tk_widget().pack(fill=tk.BOTH, expand=True)
```

Hình 3.11. Cập nhật, vẽ ma trận và hiển thị đồ thị

- `def draw_graph(canvas_frame, graph, shortest_path=None)`: Hiển thị đồ thị trên giao diện, bao gồm các đỉnh, cạnh và đường đi ngắn nhất (nếu có).
  - **Tạo đồ thị với NetworkX:**
    - `nx.DiGraph()`: Đồ thị có hướng.
    - `G.add_edge(i, j, weight=graph[i][j])`: Thêm cạnh và trọng số vào đồ thị.
  - **Tính toán vị trí các đỉnh:**
    - `nx.spring_layout(G)`: Bố trí vị trí các đỉnh.
  - **Hiển thị đồ thị:**

Sử dụng Matplotlib để vẽ đồ thị trên giao diện Tkinter.

- **Đường đi ngắn nhất:**

Nếu `shortest_path` không rỗng, tô màu đỏ các cạnh trên đường đi ngắn nhất.

- `draw_graph(canvas_frame, graph, shortest_path)`

Cập nhật thông tin đường đi ngắn nhất giữa hai đỉnh và hiển thị lại đồ thị.

- **Kiểm tra giá trị nhập vào:**

Lấy giá trị `start` và `end` từ giao diện.

Kiểm tra xem hai đỉnh này có hợp lệ không.

- **Tìm đường đi ngắn nhất:**

Sử dụng `reconstruct_path(next_node, start, end)` để tái dựng đường đi.

Hiển thị kết quả trong `result_label`.

- **Vẽ lại đồ thị:**

Tô màu đường đi ngắn nhất trên đồ thị (nếu có).

### 3.2.10. Cập nhật khi thay đổi điểm bắt đầu, kết thúc

```
# Hàm cập nhật đồ thị khi thay đổi điểm bắt đầu và kết thúc
def update_graph():
    try:
        start = int(start_entry.get())
        end = int(end_entry.get())
        if start < 0 or start >= len(graph) or end < 0 or end >= len(graph):
            raise ValueError("Điểm bắt đầu hoặc kết thúc không hợp lệ.")
    except ValueError:
        result_label.config(text="Vui lòng nhập số hợp lệ (0-{}).".format(len(graph) - 1))
    return
```

Hình 3.12. Cập nhật khi thay đổi điểm bắt đầu, kết thúc

- **Lấy giá trị từ ô nhập liệu:**

`start_entry.get()`: Lấy giá trị của điểm bắt đầu.

`end_entry.get()`: Lấy giá trị của điểm kết thúc.

- **Kiểm tra tính hợp lệ của đầu vào:** Giá trị phải là số nguyên nằm trong khoảng từ 0 đến  $\text{len}(\text{graph}) - 1$  (tức là chỉ số các đỉnh trong đồ thị). Nếu không hợp lệ, chương trình báo lỗi và yêu cầu nhập lại.
- **Thông báo lỗi:**
  - Nếu đầu vào không phải số hoặc nằm ngoài khoảng hợp lệ, hàm báo lỗi qua `result_label` (hiển thị dưới dạng nhãn trên giao diện).
  - `return`: Dừng hàm nếu phát hiện lỗi.

### 3.2.11. Tìm đường đi ngắn nhất

```
# Tìm đường đi ngắn nhất
shortest_path = reconstruct_path(next_node, start, end)
if shortest_path:
    distance = dist[start][end]
    result_label.config(text=f"Đường đi ngắn nhất: {shortest_path}\nKhoảng cách: {distance}")
else:
    result_label.config(text=f"Không có đường đi từ {start} đến {end}.")
```

Hình 3.13. Tìm đường đi ngắn nhất

- `reconstruct_path` tái dựng đường đi ngắn nhất từ điểm bắt đầu `start` đến điểm kết thúc `end` dựa trên ma trận truy vết `next_node` (đã được tính từ thuật toán Floyd-Warshall).
- `shortest_path`:
  - Là danh sách các đỉnh tạo nên đường đi ngắn nhất.
  - Nếu không tồn tại đường đi giữa hai điểm, trả về danh sách rỗng `[]`.
- **Kiểm tra đường đi:**

Nếu `shortest_path` không rỗng: `dist[start][end]`: Lấy giá trị khoảng cách ngắn nhất từ ma trận `dist`. Hiển thị đường đi (danh sách các đỉnh) và khoảng cách tương ứng trên giao diện (qua `result_label`).

Nếu `shortest_path` rỗng: Hiển thị thông báo không tồn tại đường đi giữa hai điểm.

- `draw_graph` : Vẽ đồ thị mới với các cạnh và trọng số từ ma trận graph. Nếu tồn tại đường đi ngắn nhất, đường đi này sẽ được tô màu đỏ trên đồ thị để người dùng dễ dàng nhận biết.

### 3.2.12. Tạo khung và hiện kết quả

```
# Khung chứa đồ thị
canvas_frame = tk.Frame(root)
canvas_frame.pack(side=tk.TOP, fill=tk.BOTH, expand=True)

# Khung chứa ma trận trọng số
matrix_frame = tk.Frame(root)
matrix_frame.pack(side=tk.TOP, padx=10, pady=10)

# Khung điều khiển
control_frame = tk.Frame(root)
control_frame.pack(side=tk.BOTTOM, fill=tk.X, padx=10, pady=10)

# Điều khiển số đỉnh
tk.Label(control_frame, text="Số đỉnh:").pack(side=tk.LEFT)
vertex_entry = ttk.Entry(control_frame, width=5)
vertex_entry.insert(0, "5")
vertex_entry.pack(side=tk.LEFT, padx=5)

vertex_button = ttk.Button(control_frame, text="Tạo đồ thị", command=update_graph_vertices)
vertex_button.pack(side=tk.LEFT, padx=10)

# Điều khiển điểm bắt đầu và kết thúc
tk.Label(control_frame, text="Điểm bắt đầu:").pack(side=tk.LEFT)
start_entry = ttk.Entry(control_frame, width=5)
start_entry.insert(0, "0")
start_entry.pack(side=tk.LEFT, padx=5)

tk.Label(control_frame, text="Điểm kết thúc:").pack(side=tk.LEFT)
end_entry = ttk.Entry(control_frame, width=5)
end_entry.insert(0, "4")
end_entry.pack(side=tk.LEFT, padx=5)

update_button = ttk.Button(control_frame, text="Cập nhật", command=update_graph)
update_button.pack(side=tk.LEFT, padx=10)

# Nhấn kết quả
result_label = tk.Label(control_frame, text="")
result_label.pack(side=tk.LEFT, padx=10)
```

Hình 3.14. Tạo khung và hiện kết quả

- `tk.Tk()`: Tạo cửa sổ chính của ứng dụng Tkinter.
- `root.title("Giải thuật Floyd-Warshall")`: Đặt tiêu đề cho cửa sổ.
- `root.geometry("1000x800")`: Thiết lập kích thước cửa sổ (1000 pixel rộng, 800 pixel cao).
- **Khung chứa đồ thị**

- `tk.Frame(root)`: Tạo một khung (frame) bên trong cửa sổ chính. Khung này sẽ được dùng để hiển thị đồ thị bằng `matplotlib`.
  - `pack(side=tk.TOP, fill=tk.BOTH, expand=True)`:
    - `side=tk.TOP`: Đặt khung ở phía trên cùng của cửa sổ.
    - `fill=tk.BOTH`: Kéo giãn khung theo cả chiều ngang và chiều dọc để lấp đầy không gian.
    - `expand=True`: Cho phép khung mở rộng khi cửa sổ được thay đổi kích thước.
  - **Khung chứa ma trận trọng số**
    - `tk.Frame(root)`: Tạo khung khác để chứa ma trận trọng số của đồ thị.
    - `pack(side=tk.TOP, padx=10, pady=10)`:
      - `side=tk.TOP`: Đặt khung này ngay dưới khung chứa đồ thị.
      - `padx=10, pady=10`: Tạo khoảng cách 10 pixel theo cả hai chiều x (trái/phải) và y (trên/dưới).
  - **Khung điều khiển**
    - `tk.Frame(root)`: Tạo một khung khác để chứa các nút điều khiển.
    - `pack(side=tk.BOTTOM, fill=tk.X, padx=10, pady=10)`:
      - `side=tk.BOTTOM`: Đặt khung này ở phía dưới cùng của cửa sổ.
      - `fill=tk.X`: Kéo giãn khung theo chiều ngang để lấp đầy không gian.
      - `padx=10, pady=10`: Tạo khoảng cách 10 pixel theo hai chiều ngang và dọc.
  - **Điều khiển số đỉnh**
    - `tk.Label`: Tạo nhãn hiển thị với nội dung "Số đỉnh:".
    - `ttk.Entry`:
      - Tạo ô nhập liệu cho người dùng để nhập số đỉnh của đồ thị.
      - `width=5`: Đặt chiều rộng ô nhập liệu (cho phép nhập tối đa 5 ký tự).
-

- `insert(0, "5")`: Mặc định đặt giá trị "5" trong ô nhập liệu.
- `ttk.Button`:
  - Tạo một nút với nội dung "Tạo đồ thị".
  - `command=update_graph_vertices`: Gọi hàm `update_graph_vertices` khi nhấn nút để tạo đồ thị mới với số đỉnh được chỉ định.
- **Điều khiển điểm bắt đầu và kết thúc**
  - `tk.Label`: Tạo các nhãn hiển thị "Điểm bắt đầu:" và "Điểm kết thúc:".
  - `tk.Entry`:
    - Tạo hai ô nhập liệu để người dùng nhập điểm bắt đầu và điểm kết thúc.
    - `insert(0, "0")`: Mặc định giá trị điểm bắt đầu là 0.
    - `insert(0, "4")`: Mặc định giá trị điểm kết thúc là 4.
- **Nút cập nhật**
  - `ttk.Button`:
    - Tạo nút "Cập nhật".
    - `command=update_graph`: Gọi hàm `update_graph` khi nhấn nút để tính toán và hiển thị đường đi ngắn nhất giữa điểm bắt đầu và kết thúc.
- **Nhãn hiển thị kết quả**
  - `tk.Label`:
    - Tạo một nhãn rỗng (`text=""`) để hiển thị kết quả đường đi ngắn nhất và khoảng cách.
    - Nhãn này sẽ được cập nhật nội dung thông qua hàm `result_label.config()` trong các phần xử lý khác.

### 3.2.13. Khởi tạo

```
# Khởi tạo đồ thị mẫu
graph = create_sample_graph(5)
dist, next_node = floyd_warshall(graph)

# Hiển thị ma trận trọng số
show_weight_matrix(graph)

# Vẽ đồ thị ban đầu
draw_graph(canvas_frame, graph)
```

Hình 3.15. Khởi tạo đồ thị

- `Create_sample_graph(5)`: Gọi hàm tạo một đồ thị mẫu với **5 đỉnh**
  - Ma trận trọng số ban đầu của đồ thị mẫu có dạng như sau:  
[[0, 4, INF, 7, INF],  
[4, 0, 3, INF, 6],  
[INF, 3, 0, 2, INF],  
[7, INF, 2, 0, 5],  
[INF, 6, INF, 5, 0]]
- `floyd_warshall(graph)`: Thực hiện thuật toán Floyd-Warshall trên đồ thị `graph`, giúp tái dựng lại đường đi ngắn nhất khi cần thiết.
- `show_weight_matrix(graph)`: Hiển thị ma trận trọng số của đồ thị lên giao diện người dùng.
- `draw_graph(canvas_frame, graph)`: Vẽ đồ thị ban đầu lên một **frame canvas** trong giao diện Tkinter.

### 3.2.14. Chạy chương trình

```
root.mainloop()
```

Hình 3.16. Chạy chương trình

Đây là phương thức chạy vòng lặp sự kiện của ứng dụng Tkinter.

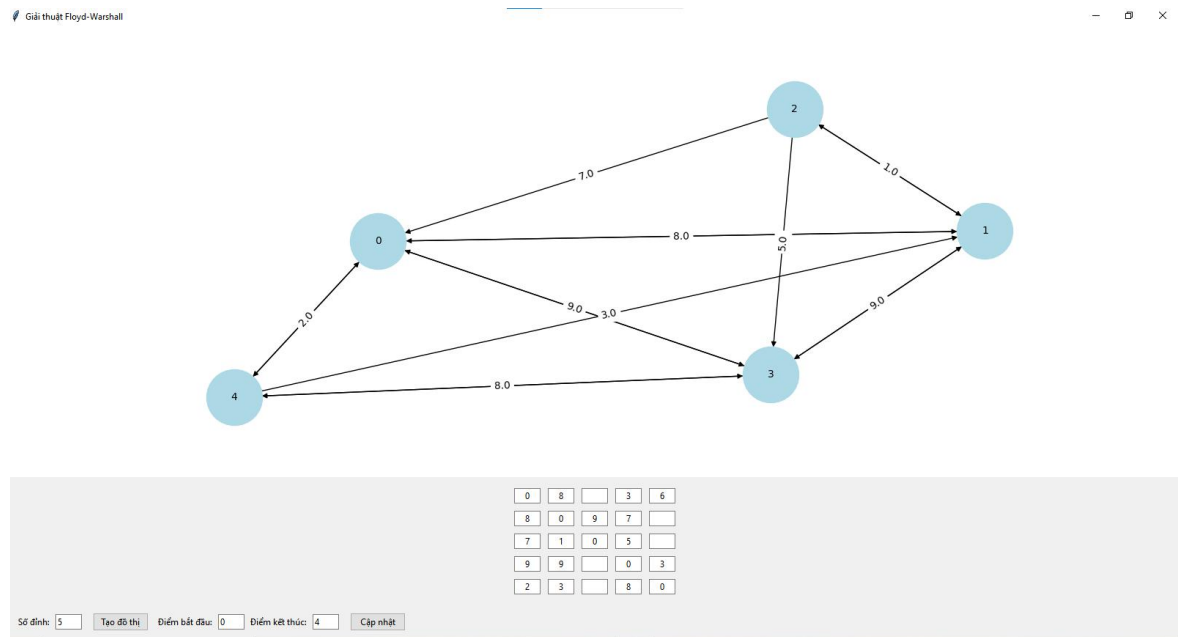
- Khi được gọi, ứng dụng Tkinter sẽ bắt đầu nhận và xử lý các sự kiện người dùng
- Chương trình sẽ tiếp tục chạy và hiển thị giao diện cho đến khi người dùng đóng cửa sổ ứng dụng.



## CHƯƠNG 4: KẾT QUẢ NGHIÊN CỨU

### 4.1. Giao diện người dùng

#### 4.1.1. Giao diện chính



Hình 4.1. Màn hình giao diện

#### 4.1.2. Khung nhập thủ công khoảng cách giữa các đỉnh trong ma trận

0	8		3	6
8	0	9	7	
7	1	0	5	
9	9		0	3
2	3		8	0

Hình 4.2. Khung nhập ma trận

#### 4.1.3. Vị trí nhập đỉnh, điểm bắt đầu và điểm đích

Số đỉnh:  Tạo đồ thị Điểm bắt đầu:  Điểm kết thúc:  Cập nhật

Hình 4.3. Vị trí nhập đỉnh, điểm đầu và điểm đích

## 4.2. Cách sử dụng

### 4.2.1. Thêm đỉnh cho đồ thị

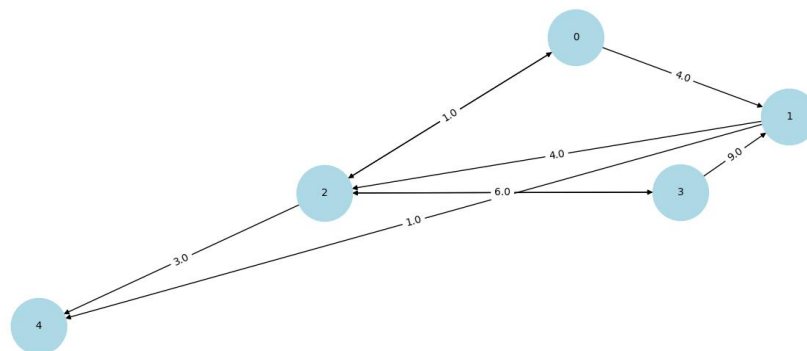
- Nhập số đỉnh muốn tạo vào ô , sau đó click vào nút “Tạo đồ thị ”, khi đó 1 đồ thị mới với số đỉnh đã nhập sẽ được tạo ra.
- Ví dụ: Đồ thị mẫu ban đầu có 5 đỉnh.



Số đỉnh:

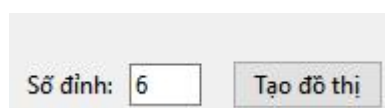
Hình 4.4. Nhập số đỉnh đồ thị

Giải thuật Floyd-Warshall



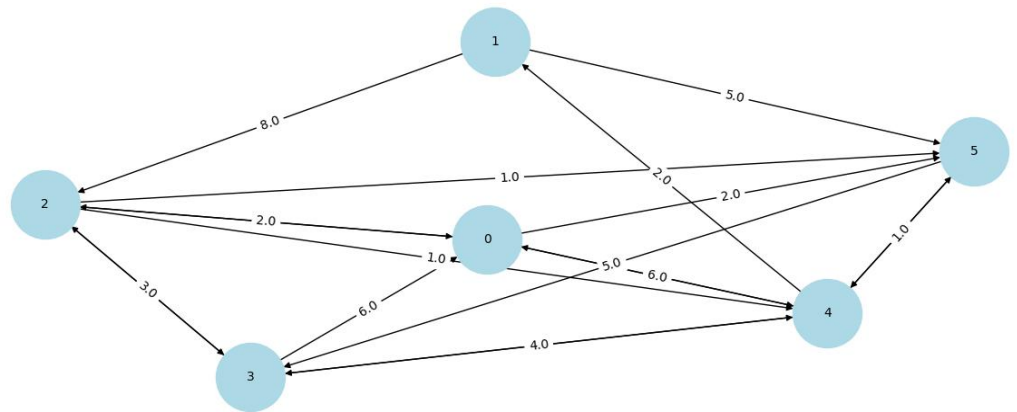
Hình 4.5. Đồ thị ví dụ 1

Sau khi thay đổi số đỉnh và nhấn nút “Tạo đồ thị ”, ta sẽ có hình như bên dưới:



Số đỉnh:

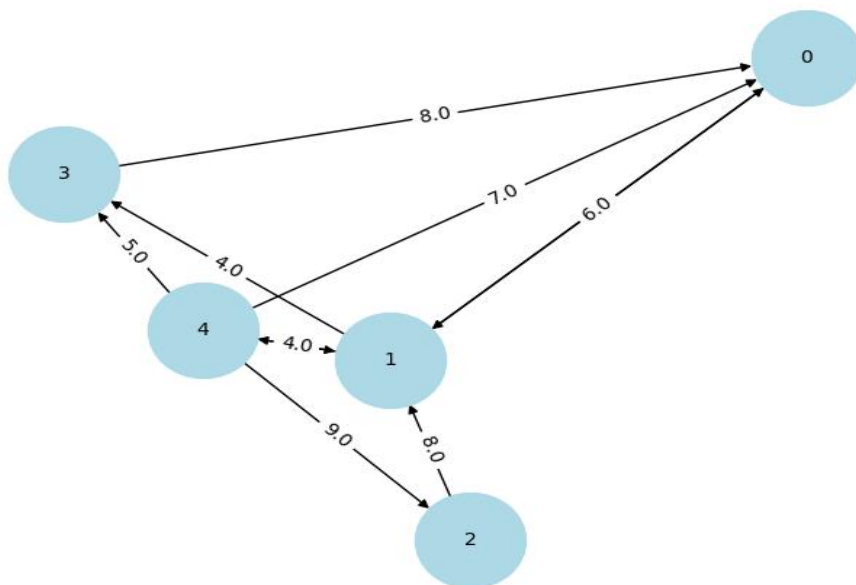
Hình 4.6. Nút tạo đồ thị



Hình 4.7. Đồ thị kết quả 1

#### 4.2.2. Thay đổi độ dài giữa các đỉnh của đồ thị

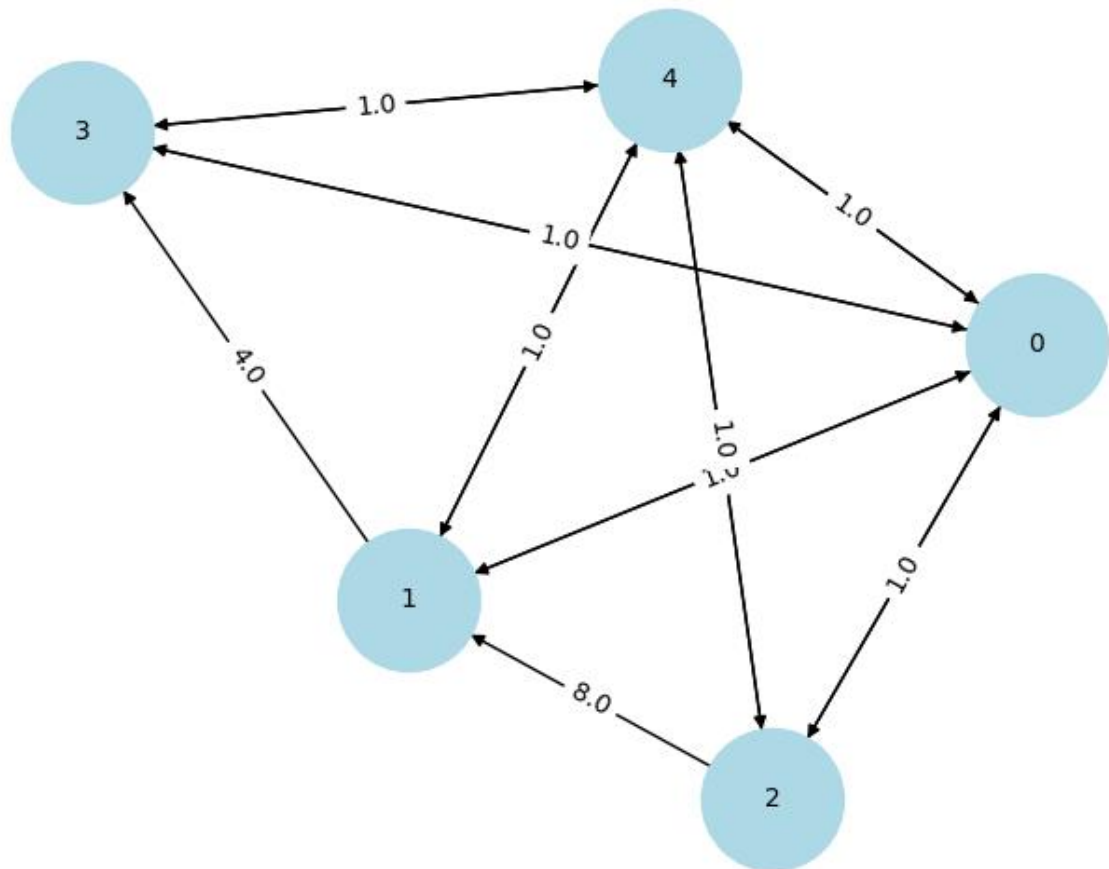
- Sau khi tạo đồ thị có số đỉnh tương ứng, người dùng có thể trực tiếp thay đổi độ dài giữa các đỉnh của đồ thị thông qua khung ma trận trong giao diện
- Ví dụ ,một đồ thị với 5 đỉnh với độ dài giữa các đỉnh khác nhau:



0	1			
6	0		4	2
	8	0		
8			0	
7	4	9	5	0

Hình 4.8. Đồ thị ví dụ 2

Sau khi thay đổi độ dài giữa các đỉnh, ta được hình như sau:



0	1	1	1	1
1	0		4	1
1	8	0		1
1			0	1
1	1	1	1	0

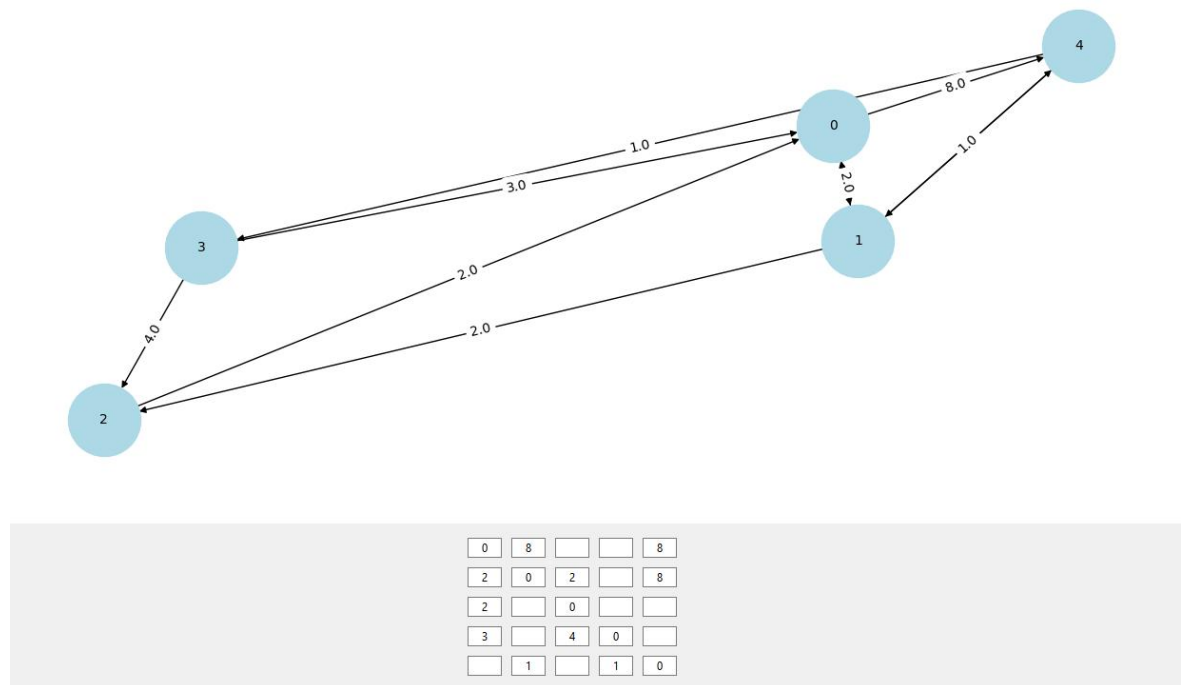
Hình 4.9. Đồ thị kết quả 2

#### 4.2.3. Chọn điểm bắt đầu và kết thúc của đồ thị

- Trong đồ thị, Khi chọn điểm bắt đầu và kết thúc và nhấn nút “Cập nhật”, đồ thị sẽ hiển thị đường đi ngắn nhất từ điểm bắt đầu đến điểm kết

thúc mà người dùng đã chọn , đường đi ngắn nhất sẽ được tô đỏ để hiển thị kết quả.

- Ví dụ ,cho một đồ thị có 5 đỉnh như sau:



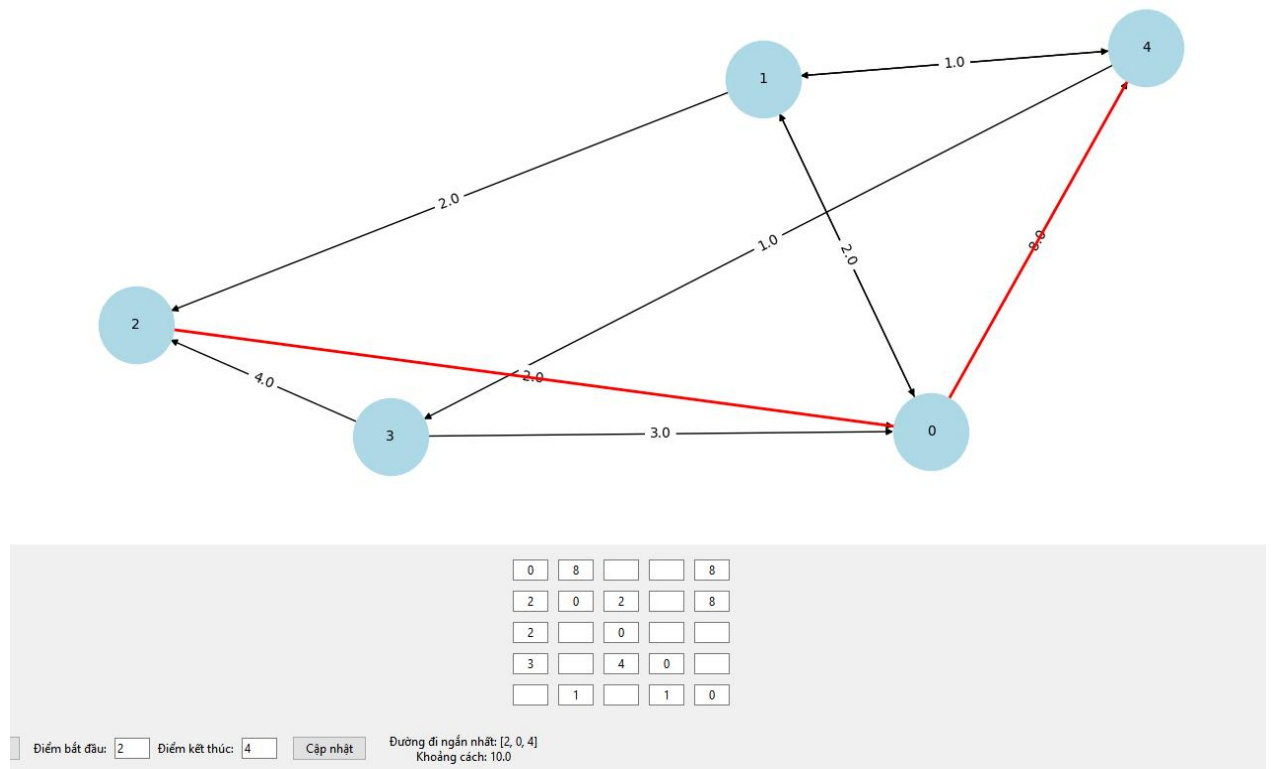
Hình 4.10. Đồ thị ví dụ 3

Ta chọn điểm bắt đầu là 2 , điểm kết thúc là 4,sau đó nhấn “Cập nhật”:

Điểm bắt đầu:  Điểm kết thúc:

Hình 4.11. Khung nhập điểm bắt đầu và kết thúc

Ta được kết quả như sau:



Hình 4.12. Đồ thị kết quả 3

### 4.3. Hiệu năng

- **Tính toán hiệu quả:**

- Giải thuật Floyd-Warshall được triển khai thành công với thời gian chạy  $O(V^3)$ , phù hợp với các đồ thị nhỏ (tối đa 10 đỉnh trong giao diện này).
- Ma trận trọng số được cập nhật động khi người dùng chỉnh sửa, và kết quả luôn phản ánh chính xác các thay đổi.

- **Trực quan hóa:**

- Đồ thị được vẽ bằng thư viện **NetworkX** kết hợp **Matplotlib**, cho phép hiển thị các cạnh và trọng số một cách trực quan.
- Đường đi ngắn nhất được tô màu đỏ, giúp người dùng dễ dàng nhận biết.

#### 4.4. Trải nghiệm người dùng (UX)

- **Tương tác động:**

Giao diện thân thiện, cho phép:

- Nhập số đỉnh mong muốn (giới hạn từ 2 đến 10).
  - Chỉnh sửa trọng số cạnh trực tiếp trong ma trận trọng số.
  - Thay đổi điểm bắt đầu và kết thúc ngay trong giao diện để tính toán lại đường đi ngắn nhất.
- **Kết quả tức thời:** Khi người dùng thay đổi trọng số, số đỉnh hoặc điểm bắt đầu/kết thúc, giao diện sẽ tự động cập nhật đồ thị và đường đi ngắn nhất mà không cần khởi động lại ứng dụng.
  - **Thông báo lỗi rõ ràng:** Nếu người dùng nhập số đỉnh hoặc điểm không hợp lệ, thông báo lỗi sẽ xuất hiện để hướng dẫn cách sửa.

#### 4.5. Giao diện chức năng

- **Nhập số đỉnh mong muốn:** Người dùng có thể chọn số lượng đỉnh (từ 2 đến 10). Giao diện sẽ tạo đồ thị tương ứng và hiển thị ma trận trọng số.
- **Chỉnh sửa ma trận trọng số:** Các ô nhập liệu cho phép thay đổi trọng số cạnh giữa các đỉnh. Giá trị để trống hoặc âm sẽ được coi là không kết nối.
- **Hiển thị đường đi ngắn nhất:** Khi nhập điểm bắt đầu và kết thúc, giao diện hiển thị đường đi ngắn nhất, bao gồm:
  - Các đỉnh trong đường đi (theo thứ tự).
  - Tổng khoảng cách từ điểm bắt đầu đến điểm kết thúc.
  - Đường đi ngắn nhất được tô đỏ trên đồ thị.
- **Vẽ đồ thị động:** Mỗi lần cập nhật, đồ thị được vẽ lại để phản ánh chính xác trọng số cạnh và đường đi ngắn nhất hiện tại.

#### 4.6. Minh họa giao diện

##### Giao diện ban đầu:

- Tạo đồ thị mẫu 5x5, hiển thị ma trận trọng số với các giá trị ngẫu nhiên.
- Đồ thị được vẽ với các cạnh, trọng số, và các đỉnh.

##### Tùy chỉnh ma trận:

- Người dùng có thể chỉnh sửa trực tiếp trọng số của các cạnh trong ma trận.
- Khi thay đổi, đồ thị và kết quả đường đi ngắn nhất tự động cập nhật.

##### Tính toán đường đi ngắn nhất:

Nhập điểm bắt đầu và kết thúc. Đường đi ngắn nhất được tính toán và hiển thị.

#### 4.7. Ưu điểm và nhược điểm

##### • Ưu điểm

- Ứng dụng trực quan, dễ sử dụng.
- Cho phép tương tác động, thay đổi thông số đồ thị và tính toán lại mà không cần khởi động lại.
- Giao diện đơn giản nhưng đầy đủ chức năng, hỗ trợ tốt cho việc học tập và minh họa giải thuật Floyd-Warshall.

##### • Nhược điểm

- Giới hạn số lượng đỉnh (2 đến 10) do hiệu năng và khả năng hiển thị.
- Chưa hỗ trợ nhập đồ thị từ file hoặc lưu kết quả.



## CHƯƠNG 5: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

### 5.1. Kết Luận

Sau quá trình nghiên cứu và thực hiện, đồ án đã đạt được các kết quả sau:

- Xây dựng thành công chương trình mô phỏng giải thuật Floyd-Warshall bằng ngôn ngữ Python, sử dụng thư viện đồ họa Tkinter để minh họa trực quan.
- Chương trình cho phép người dùng tương tác với đồ thị, bao gồm nhập số lượng đỉnh, chỉnh sửa ma trận trọng số, và thay đổi điểm bắt đầu và kết thúc để tìm đường đi ngắn nhất.
- Kết quả tính toán đường đi ngắn nhất được hiển thị trực quan trên giao diện, với các thông tin đầy đủ về lộ trình và tổng khoảng cách giữa các đỉnh.
- Ứng dụng góp phần hỗ trợ việc giảng dạy, học tập và nghiên cứu lý thuyết đồ thị, đặc biệt trong lĩnh vực tối ưu hóa trên đồ thị.

### 5.2. Hướng phát triển

Mở rộng ứng dụng để hỗ trợ xử lý các đồ thị lớn hơn, tối ưu hóa hiệu suất và thời gian tính toán. Thêm tính năng nhập và xuất dữ liệu từ file để hỗ trợ phân tích đồ thị phức tạp hơn. Phát triển chức năng kiểm tra và xử lý chu trình âm trong đồ thị. Tích hợp các giải thuật tìm đường đi ngắn nhất khác như Dijkstra, Bellman-Ford để so sánh hiệu quả và ứng dụng.

## DANH MỤC TÀI LIỆU THAM KHẢO

- [1] T. H. Nam, TÀI LIỆU GIẢNG DẠY MÔN LÝ THUYẾT ĐỒ THỊ, Trà Vinh, 2013.
- [2] Viblo algorithm ,Thuật toán Floyd-Warshall,Việt Nam,2021.  
Nguồn: <https://viblo.asia/p/thuat-toan-floyd-warshall-GrLZDBng5k0>
- [3] Michael Sambol, Thuật toán Floyd–Warshall trong 4 phút, Hoa Kỳ,2016.  
Nguồn: <https://www.youtube.com/watch?v=4OQeCuLYj-4>
- [4] Dương Thế Hưng,Các thuật toán tìm đường đi ngắn nhất trên đồ thị có trọng số (Phần 3) – Bellman-Ford,Việt Nam, 2023.  
Nguồn: <https://codedream.edu.vn/bellman-ford/>

## PHỤ LỤC