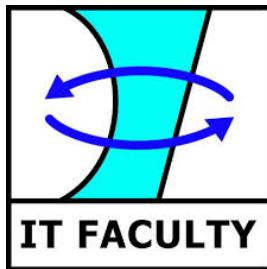


TRƯỜNG ĐẠI HỌC BÁCH KHOA  
KHOA CÔNG NGHỆ THÔNG TIN



ĐỒ ÁN LẬP TRÌNH TÍNH TOÁN

ĐỀ TÀI: TÌM ĐƯỜNG ĐI NGẮN  
NHẤT

**Người hướng dẫn:** TS. Nguyễn Văn Hiệu

**Sinh viên thực hiện:**

Tên sinh viên 1: Nguyễn Hữu Rin

Lớp: 25Nh.15A

Tên sinh viên 2: Huỳnh Nguyễn Hồng Nhi

Lớp: 25Nh.15A

## Mục lục

<b>MỤC LỤC</b>	<b>i</b>
<b>DANH MỤC HÌNH VẼ</b>	<b>iii</b>
<b>MỞ ĐẦU</b>	<b>1</b>
<b>1 TỔNG QUAN ĐỀ TÀI</b>	<b>2</b>
1.0.1 Bài toán tìm đường đi ngắn nhất và tầm quan trọng . . . . .	2
1.0.2 Mục đích và mục tiêu thực hiện đề tài . . . . .	2
1.0.3 Phạm vi và đối tượng nghiên cứu . . . . .	3
1.0.4 Phương pháp nghiên cứu . . . . .	3
1.0.5 Cấu trúc môn học . . . . .	3
<b>2 CƠ SỞ LÝ THUYẾT</b>	<b>4</b>
2.1 Ý tưởng . . . . .	4
2.1.1 Thuật toán Dijkstra – phương pháp tham lam . . . . .	4
2.1.2 Thuật toán Bellman Ford – phương pháp quy hoạch động . . . . .	4
2.1.3 So sánh . . . . .	4
2.2 Cơ sở lý thuyết . . . . .	5
2.2.1 Kiến thức nền tảng về lý thuyết đồ thị . . . . .	5
2.2.2 Nguyên lý tối ưu và tính đúng đắn . . . . .	6
<b>3 TỔ CHỨC CẤU TRÚC DỮ LIỆU VÀ THUẬT TOÁN</b>	<b>8</b>
3.1 Phát biểu bài toán . . . . .	8
3.2 Cấu trúc dữ liệu . . . . .	8
3.2.1 Các thư viện được sử dụng trong chương trình . . . . .	8
3.2.2 Các cấu trúc dữ liệu tự xây dựng . . . . .	9
3.3 Thuật toán . . . . .	12
3.3.1 Thuật toán Dijkstra . . . . .	12
3.3.2 Thuật toán Bellman-Ford . . . . .	21
3.3.3 Tổng kết . . . . .	29
<b>4 CHƯƠNG TRÌNH VÀ KẾT QUẢ</b>	<b>30</b>
4.1 Tổ chức chương trình . . . . .	30
4.1.1 Tổng quan cấu trúc . . . . .	30
4.1.2 Chi tiết chức năng . . . . .	31
4.1.3 Phân loại và mô tả chức năng các hàm trong chương trình . . . . .	31
4.2 Ngôn ngữ cài đặt . . . . .	37
4.2.1 Nhận xét . . . . .	37
<b>5 KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN</b>	<b>38</b>
5.1 Kết luận . . . . .	38
5.2 Hướng phát triển . . . . .	38
<b>TÀI LIỆU THAM KHẢO</b>	<b>39</b>

<b>PHỤ LỤC</b>	<b>40</b>
.1 Toàn bộ chương trình và cài đặt . . . . .	40
.2 Mã nguồn chương trình . . . . .	40
.2.1 Mã nguồn chương trình chính : main.cpp . . . . .	40
.2.2 Mã nguồn chương trình ghi dữ liệu : graph.cpp . . . . .	45
.2.3 Mã nguồn chương trình thực hiện thuật toán : algorithms.cpp . . . . .	50
.2.4 Mã nguồn chương trình so sánh : comparison.cpp . . . . .	54
.2.5 Mã nguồn chương trình vẽ giao diện : GUI.cpp . . . . .	58
.2.6 Mã nguồn chương trình vẽ khung : graphics.cpp . . . . .	72
.3 Mã nguồn các thư viện tự cài đặt . . . . .	76
.3.1 Mã nguồn thư viện dữ liệu đồ thị : graph.h . . . . .	76
.3.2 Mã nguồn thư viện thiết lập màu : Colors.h . . . . .	77
.3.3 Mã nguồn thư viện toàn cục : Global.h . . . . .	77
.3.4 Mã nguồn thư viện đồ họa : GUI.h . . . . .	78
.3.5 Mã nguồn thư viện thuật toán : algorithms.h . . . . .	78
.3.6 Mã nguồn thư viện so sánh : comparison.h . . . . .	79

## Danh sách hình vẽ

1.1	Hình minh họa lý thuyết đồ thị . . . . .	2
2.1	Optimal Substructure . . . . .	6
3.1	Cấu trúc dữ liệu Edge . . . . .	9
3.2	Cấu trúc dữ liệu Graph . . . . .	10
3.3	Cấu trúc dữ liệu PathResult . . . . .	10
3.4	Class Algorithms . . . . .	10
3.5	Cấu trúc dữ liệu PerformanceMetrics . . . . .	11
3.6	Cấu trúc dữ liệu ComparisonReport . . . . .	11
3.7	Class Comparison . . . . .	11
3.8	Class GUI . . . . .	11
3.9	Đồ thị minh họa thuật toán Dijkstra . . . . .	15
3.10	Hình minh họa trạng thái ban đầu . . . . .	16
3.11	Các bước cập nhật trọng số đường đi . . . . .	16
3.12	Các bước cập nhật trọng số đường đi . . . . .	17
3.13	Các bước cập nhật trọng số đường đi . . . . .	18
3.14	Các bước cập nhật trọng số đường đi . . . . .	18
3.15	Các bước cập nhật trọng số đường đi . . . . .	19
3.16	Các bước cập nhật trọng số đường đi . . . . .	19
3.17	Kết quả cuối cùng của thuật toán . . . . .	20
3.18	Hình minh hóa thuật toán Bellman-Ford . . . . .	25
3.19	Hình minh hóa thuật toán Bellman-Ford . . . . .	26
3.20	Hình minh hóa thuật toán Bellman-Ford . . . . .	26
3.21	Hình minh hóa thuật toán Bellman-Ford . . . . .	27
3.22	Hình minh hóa thuật toán Bellman-Ford . . . . .	28
3.23	Hình minh hóa thuật toán Bellman-Ford . . . . .	28
3.24	Hình minh hóa thuật toán Bellman-Ford . . . . .	28
4.1	Tổ chức chương trình . . . . .	30
4.2	Tổ chức tệp tin . . . . .	30

## MỞ ĐẦU

Sự phát triển của lý thuyết đồ thị trong thế kỷ XX đã đặt nền móng cho cuộc cách mạng trong khoa học máy tính và tối ưu hóa. Trong số các bài toán kinh điển, bài toán tìm đường đi ngắn nhất nổi lên như một thách thức trọng tâm với vô vàn ứng dụng thực tiễn. Việc xác định lộ trình tối ưu không chỉ đơn thuần là tìm kiếm khoảng cách vật lý nhỏ nhất mà còn liên quan đến việc tối ưu hóa chi phí, thời gian và tài nguyên trong các mạng lưới phức tạp. Đề tài '**Phân tích và đánh giá hiệu năng giữa thuật toán Dijkstra và Bellman-Ford**' được thực hiện nhằm mục đích giải soát các đặc tính thuật toán, phân tích sự tương quan về độ phức tạp và xác lập các điều kiện tối ưu cho từng kịch bản, tình huống dữ liệu cụ thể.

Nghiên cứu này tập trung phân tích chuyên sâu về nguyên lý hoạt động, các bất biến thuật toán và đặc thù cấu trúc của Dijkstra cùng Bellman-Ford. Mục tiêu trọng tâm là thiết lập một hệ thống đánh giá hiệu năng đa chiều dựa trên ba chỉ số: thời gian thực thi, độ phức tạp bộ nhớ và tính ổn định khi xử lý đồ thị có trọng số âm.

Để đảm bảo tính khách quan và khả năng tái lập, nghiên cứu được triển khai thực nghiệm trên các bộ dữ liệu tổng hợp, kết hợp các kỹ thuật trực quan hóa kết quả, tiến trình thuật toán. Phạm vi nghiên cứu bao trùm trên cả cấu trúc đồ thị có hướng và vô hướng, đồng thời xác lập các điều kiện biên về trọng số (đặc biệt là xử lý trường hợp trọng số âm và chu trình âm). Phương pháp luận được xây dựng dựa trên sự phối hợp giữa mô hình hóa toán học và thực nghiệm định lượng. Cấu trúc đề tài được tổ chức tuyến tính từ cơ sở lý thuyết, phân tích thuật toán, đến kiểm chứng thực nghiệm, so sánh, tổng hợp kết quả và cuối cùng là rút ra nhận xét.

Do thời gian và trình độ chuyên môn còn hạn chế, chúng em nhận thức rằng đồ án không tránh khỏi những thiếu sót. Chúng em xin chân thành cảm ơn sự hướng dẫn tận tâm của thầy **TS. Nguyễn Văn Hiệu** trong suốt quá trình thực hiện đề tài. Nhóm chúng em rất mong nhận được những ý kiến đóng góp quý báu từ quý thầy cô để hoàn thiện hơn nữa đề tài này.

Nhóm sinh viên thực hiện.

# Chương 1. TỔNG QUAN ĐỀ TÀI

## 1.0.1 Bài toán tìm đường đi ngắn nhất và tầm quan trọng

Bài toán đường đi ngắn nhất một nguồn (single-source shortest path) yêu cầu xác định khoảng cách nhỏ nhất từ một đỉnh nguồn  $s$  tới tất cả các đỉnh còn lại trong đồ thị có trọng số  $G = (V, E)$ . Trọng số trên các cạnh mô tả chi phí, chiều dài hoặc thời gian di chuyển; tùy thuộc vào bản chất của trọng số (dương, bằng 0 hay âm), thuật toán sẽ có nhiều cách giải quyết khác nhau và có độ phức tạp riêng.

Lý thuyết đồ thị và bài toán đường đi ngắn nhất nổi lên vào giữa thế kỷ XX cùng với sự phát triển của khoa học máy tính và nhu cầu tối ưu hóa mạng lưới giao thông, viễn thông và logistics. Hiện nay, các thuật toán tìm đường đi ngắn nhất là hạt nhân của nhiều hệ thống thực tế như định tuyến GPS, giao thông thông minh và giao thức mạng (OSPF, IS-IS). Vì vậy, tìm hiểu và lựa chọn thuật toán phù hợp là nhiệm vụ quan trọng trong cả nghiên cứu lẫn ứng dụng.

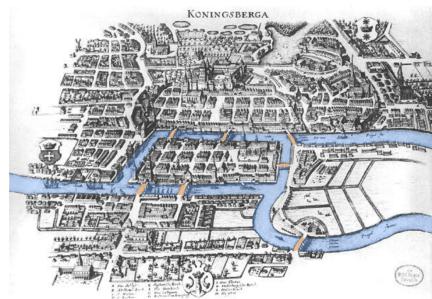
### Lịch sử phát triển các thuật toán liên quan

- **Thuật toán Dijkstra** (Edsger W. Dijkstra, 1956) sử dụng phương pháp **tham lam** kết hợp hàng đợi ưu tiên để chọn đỉnh có nhãn tạm thời nhỏ nhất và cập nhật khoảng cách. Thuật toán này hoạt động hiệu quả trên đồ thị có trọng số không âm và đạt độ phức tạp thời gian  $O(|E| \log |V|)$  khi sử dụng hàng đợi ưu tiên thích hợp
- **Thuật toán Bellman–Ford** (1956–1958) dựa trên quy hoạch động và thực hiện quá trình nối lồng toàn bộ các cạnh  $|V| - 1$  lần; ưu điểm của nó là xử lý được trọng số âm và phát hiện chu trình âm nhưng độ phức tạp thời gian cao hơn,  $O(|V||E|)$
- **Thuật toán Floyd–Warshall** (1962) là thuật toán động quy hoạch giải bài toán đường đi ngắn nhất cho mọi cặp đỉnh (all-pairs), với độ phức tạp  $O(|V|^3)$ . Đây là phương án tổng quát nhưng không phù hợp khi chỉ cần đường đi ngắn nhất từ một nguồn do chi phí tính toán lớn.

Với đề tài này, nhóm lựa chọn so sánh hai thuật toán **Dijkstra** và **Bellman–Ford** vì chúng là những phương pháp kinh điển cho bài toán đường đi ngắn nhất một nguồn. Dijkstra nổi bật về hiệu năng trên đồ thị có trọng số không âm và được ứng dụng rộng rãi; trong khi đó, Bellman–Ford có khả năng phát hiện và xử lý trọng số âm nhưng tốn thời gian hơn. Việc phân tích và đánh giá hai thuật toán này giúp xác định điều kiện áp dụng phù hợp và đóng góp thiết thực cho các ứng dụng thực tiễn.

## 1.0.2 Mục đích và mục tiêu thực hiện đề tài

Mục đích của đề tài là nghiên cứu sâu về cách thức vận hành của hai thuật toán Dijkstra và Bellman–Ford từ cơ sở lý thuyết đến ứng dụng thực tiễn. Đề tài tập trung vào các mục tiêu:



- **Phân tích chi tiết cơ sở toán học:** phương pháp tham lam (Greedy Approach) của Dijkstra và quy hoạch động (Dynamic Programming) của Bellman-Ford.
- **Xây dựng quy trình thực hiện, mô tả cách tổ chức cấu trúc dữ liệu để tối ưu hóa hiệu năng.**
- So sánh sự khác biệt về độ phức tạp thời gian, không gian và khả năng xử lý các trường hợp biên như trọng số âm.
- Khảo sát các ứng dụng thực tiễn trong định tuyến mạng (OSPF, RIP) và tài chính.

#### 1.0.3 Phạm vi và đối tượng nghiên cứu

- **Đối tượng nghiên cứu:** Thuật toán Dijkstra (sử dụng hàng đợi ưu tiên) và thuật toán Bellman-Ford (bao gồm kỹ thuật phát hiện chu trình âm).
- **Phạm vi nghiên cứu:** Đồ thị có hướng, có trọng số ( $G = (V, E)$ ), tập trung vào các trường hợp trọng số không âm và trọng số âm (không chứa chu trình âm hoặc chứa chu trình âm).

#### 1.0.4 Phương pháp nghiên cứu

- **Nghiên cứu lý thuyết:** Nghiên cứu, tổng hợp các tài liệu học thuật về lý thuyết đồ thị, thuật toán tìm đường đi ngắn nhất và nguyên lý relaxation (nối lỏng cạnh).
- **Phân tích thuật toán:** Dựa trên độ phức tạp của thuật toán  $O(\text{time})$  để đánh giá hiệu năng trên các cấu trúc dữ liệu Array, Heap, Edge List.
- **So sánh đối chiếu:** Thiết lập bảng dựa trên ba tiêu chí đã đề ra để rút ra ưu/nhược điểm trong các điều kiện đồ thị khác nhau của từng thuật toán và đề xuất thuật toán phù hợp.

#### 1.0.5 Cấu trúc môn học

Nội dung đồ án được chia thành 5 chương chính với các nội dung cụ thể như sau:

- **Chương 1: Tổng quan đề tài** – Giới thiệu về bài toán tìm đường đi ngắn nhất, tầm quan trọng và lý do lựa chọn so sánh hai thuật toán Dijkstra và Bellman-Ford.
- **Chương 2: Cơ sở lý thuyết** - Giải thích tư tưởng tham lam và quy hoạch động, khái niệm relaxation (nối lỏng cạnh) và nguyên lý tối ưu Bellman.
- **Chương 3: Tổ chức cấu trúc dữ liệu và thuật toán** Phát biểu bài toán, mô tả các cấu trúc dữ liệu được sử dụng, quá trình Input/Output, quy trình thực hiện thuật toán Dijkstra và Bellman-Ford.
- **Chương 4: Chương trình và kết quả** Mô tả môi trường cài đặt (ngôn ngữ C), giao diện chương trình, các chức năng và trình bày kết quả thực thi trên các bộ dữ liệu khác nhau để đánh giá hiệu năng.
- **Chương 5: Kết luận và hướng phát triển** - Tổng hợp, kết luận về bài nghiên cứu đồng thời đưa ra các phương án nhằm tối ưu và phát triển, mở rộng đề tài trong tương lai.

## Chương 2. CƠ SỞ LÝ THUYẾT

### 2.1 Ý tưởng

#### 2.1.1 Thuật toán Dijkstra – phương pháp tham lam

Dijkstra giải bài toán đường đi ngắn nhất một nguồn trên đồ thị có trọng số **không âm**. Ý tưởng cốt lõi là **tham lam**: tại mỗi bước, chọn đỉnh chưa xử lý có nhãn (khoảng cách tạm thời) nhỏ nhất, rồi cập nhật (nối lỏng cạnh) khoảng cách tới các đỉnh kề nếu đi qua đỉnh này cho tổng chi phí tốt hơn. Do mọi trọng số đều không âm, khi một đỉnh được chọn, khoảng cách tới đỉnh đó là tối ưu, vì không thể có đường khác ngắn hơn nữa đi qua các đỉnh chưa được xét. Thuật toán thường sử dụng **hàng đợi ưu tiên (priority queue)** để nhanh chóng chọn đỉnh có khoảng cách nhỏ nhất, nên độ phức tạp thời gian đạt  $O(|E| \log |V|)$  với cấu trúc dữ liệu phù hợp. Đây là cách tiếp cận hiệu quả khi trọng số đều không âm và đồ thị có số cạnh lớn.

#### 2.1.2 Thuật toán Bellman Ford – phương pháp quy hoạch động

Bellman–Ford áp dụng **quy hoạch động** và nguyên lý tối ưu: “bất kỳ đoạn con của một đường đi ngắn nhất cũng là đường đi ngắn nhất giữa hai điểm của nó”. Thuật toán khởi tạo khoảng cách từ nguồn tới mọi đỉnh bằng vô cùng (trừ nguồn bằng 0) và lặp lại quá trình nối lỏng toàn bộ các cạnh  $|V| - 1$  lần. Mỗi lần nối lỏng, nếu khoảng cách hiện tại có thể được giảm bằng cách đi qua cạnh  $(u, v)$ , ta cập nhật giá trị mới. Sau  $|V| - 1$  lần, khoảng cách thu được là tối ưu; một lần nối lỏng nữa sẽ phát hiện chu trình âm nếu tồn tại. Do phải duyệt toàn bộ các cạnh nhiều lần, thuật toán có độ phức tạp thời gian  $O(|V||E|)$ . Ưu điểm của Bellman–Ford là xử lý được trọng số âm và phát hiện chu trình âm; nhược điểm là chạy chậm hơn Dijkstra trên đồ thị lớn.

#### 2.1.3 So sánh

Hai thuật toán đều dựa trên thao tác **nối lỏng cạnh** để cải thiện ước lượng khoảng cách. Dijkstra sử dụng chiến lược tham lam để mở rộng dần “vòng tròn” các đỉnh đã biết khoảng cách tối ưu, tận dụng hàng đợi ưu tiên để đạt hiệu năng cao trên trọng số không âm. Ngược lại, Bellman–Ford sử dụng phương pháp quy hoạch động, lặp lại việc nối lỏng tất cả các cạnh nhằm đảm bảo tính tối ưu dù có trọng số âm. Việc hiểu rõ hai ý tưởng này là tiền đề để phân tích cấu trúc, độ phức tạp và điều kiện áp dụng của từng thuật toán trong các chương tiếp theo.

## 2.2 Cơ sở lý thuyết

### 2.2.1 Kiến thức nền tảng về lý thuyết đồ thị

Để giải quyết bài toán đường đi ngắn nhất một cách hiệu quả, việc nắm vững các kiến thức nền tảng của lý thuyết đồ thị là điều tiên quyết.

#### Định nghĩa và Phân loại

Một đồ thị được mô hình hóa toán học dưới dạng  $G = (V, E)$ , trong đó  $V$  (Vertices) là tập hợp các đỉnh và  $E$  (Edges) là tập hợp các cạnh biểu diễn mối quan hệ giữa chúng. Tùy thuộc vào tính chất liên kết và định lượng, đồ thị được phân loại thành:

- **Có hướng hoặc vô hướng:** Xác định tính chất một chiều (digraph) hoặc hai chiều của các liên kết giữa các cặp nút.
- **Có trọng số hoặc không trọng số:** Mỗi cạnh  $(u, v)$  có thể mang trọng số  $w(u, v)$  đại diện cho các thực thể vật lý như chi phí di chuyển, khoảng cách vật lý hoặc độ trễ thời gian.

#### Cách biểu diễn đồ thị

Về mặt cấu trúc, đồ thị thường được biểu diễn dưới hai dạng phổ biến, mỗi loại có ưu điểm riêng về hiệu quả tính toán và sử dụng bộ nhớ:

- **Ma trận kề (Adjacency Matrix):** Phù hợp cho đồ thị dày, cho phép kiểm tra cạnh trong  $O(1)$  nhưng tiêu tốn  $O(V^2)$  không gian.
- **Danh sách kề (Adjacency List):** Giúp tiết kiệm bộ nhớ cho đồ thị mảnh và là cấu trúc tối ưu cho việc duyệt các lân cận trong thuật toán Dijkstra và Bellman-Ford.

#### Một số khái niệm liên quan

Các khái niệm về cấu trúc đường đi đóng vai trò quan trọng trong việc xác định tính đúng đắn của thuật toán:

- **Đường đi (path):** Một đường đi là một chuỗi các đỉnh liên tiếp  $v_0, v_1, \dots, v_k$  sao cho với mỗi  $i$  tồn tại cạnh nối  $v_i$  và  $v_{i+1}$ . Trong đồ thị có trọng số, **trọng số của đường đi** được tính bằng tổng trọng số các cạnh trên đường đi đó.
- **Đường đi đơn (simple path):** Là đường đi không lặp lại bất kỳ đỉnh nào (mỗi đỉnh chỉ xuất hiện tối đa một lần). Trong đồ thị có  $|V|$  đỉnh, một đường đi đơn chứa tối đa  $|V|$  đỉnh phân biệt và do đó tối đa  $|V| - 1$  cạnh. (Đây là lý do Bellman-Ford chỉ cần lặp  $|V| - 1$  lần để xét mọi đường đi không lặp.)
- **Chu trình (cycle):** Là một đường đi bắt đầu và kết thúc tại cùng một đỉnh, ví dụ  $v_0 \rightarrow v_1 \rightarrow \dots \rightarrow v_0$ . Chu trình được gọi là chu trình đơn nếu không lặp lại các đỉnh khác (ngoài đỉnh đầu trùng với đỉnh cuối).

- **Chu trình âm (Negative cycle):** Trong đồ thị có trọng số, một chu trình âm là chu trình có tổng trọng số các cạnh trên chu trình nhỏ hơn 0:

$$\sum_{e \in \text{cycle}} w(e) < 0$$

Nếu một chu trình âm có thể tiếp cận được từ đỉnh nguồn, thì bài toán đường đi ngắn nhất **không có nghiệm hữu hạn** vì ta có thể đi qua chu trình đó nhiều lần để giảm tổng trọng số vô hạn (thuật toán Bellman-Ford có thể phát hiện chu trình âm và báo hiệu khi phát hiện chu trình này).

- **Đồ thị liên thông (connected graph):** Trạng thái mà mọi cặp đỉnh trong đồ thị đều có ít nhất một đường đi kết nối chúng. Với đồ thị vô hướng, khái niệm liên thông là duy nhất, trong khi với đồ thị có hướng, có thể phân biệt liên thông mạnh và liên thông yếu. Trong bài toán đường đi ngắn nhất, khả năng tiếp cận từ đỉnh nguồn là yếu tố then chốt.

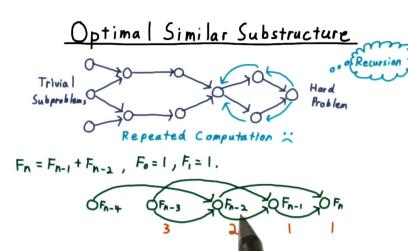
### 2.2.2 Nguyên lý tối ưu và tính đúng đắn

#### Nguyên lý tối ưu Dijkstra:

Dijkstra giải bài toán đường đi ngắn nhất trong đồ thị **không có trọng số âm**. Thuật toán khởi tạo mảng khoảng cách  $d[\cdot]$  với  $\infty$  cho mọi đỉnh, riêng nguồn  $d[s] = 0$ . Ta quản lý hai tập: tập đỉnh đã tìm được khoảng cách tối ưu và tập đỉnh chưa biết. Ở mỗi bước, thuật toán **chọn đỉnh  $u$  có  $d(u)$  nhỏ nhất** trong tập chưa biết rồi **nối lỏng** các cạnh xuất phát từ  $u$ . Do trọng số không âm, khi đỉnh  $u$  được chọn thì  $d(u)$  chắc chắn là tối ưu. Việc lựa chọn tham lam này thường được hỗ trợ bằng **hàng đợi ưu tiên**, giúp tìm nhanh đỉnh có khoảng cách nhỏ nhất và đạt độ phức tạp  $O(|E| \log |V|)$ . Dijkstra sử dụng nguyên lý tối ưu ở dạng tham lam: một khi đỉnh được rút ra khỏi hàng đợi ưu tiên, khoảng cách của nó được “cố định” và sẽ không cần cập nhật nữa.

#### Nguyên lý Tối ưu Bellman:

Một đặc điểm quan trọng của đường đi ngắn nhất là tính chất cấu trúc con tối ưu (Optimal Substructure). Nếu  $P$  là đường đi ngắn nhất từ  $s$  đến  $v$ , thì bất kỳ đường đi con nào của  $P$  cũng phải là đường đi ngắn nhất giữa các đỉnh tương ứng của nó. Đây là nền tảng để các thuật toán xây dựng dần các kết quả tối ưu từ các kết quả nhỏ hơn đã biết.



Hình 2.1: Optimal Substructure

**Tính đúng đắn và sự hội tụ:**

- **Với Dijkstra:** Tính đúng đắn được chứng minh bằng phản chứng dựa trên giả định trọng số không âm. Nếu tồn tại một đường đi khác ngắn hơn đến một đỉnh vừa được kết nạp, đường đi đó buộc phải đi qua một đỉnh chưa được thăm. Do trọng số không âm, khoảng cách đến đỉnh chưa thăm đó đã lớn hơn hoặc bằng khoảng cách đến đỉnh hiện tại, dẫn đến mâu thuẫn.
- **Với Bellman-Ford:** Thuật toán sử dụng quy nạp toán học để chứng minh rằng sau  $i$  lần lặp,  $d[v]$  sẽ là độ dài của đường đi ngắn nhất từ  $s$  đến  $v$  sử dụng tối đa  $i$  cạnh. Sau  $|V| - 1$  lần lặp, mọi đường đi đơn đều được xét đến. Nếu sau lần lặp thứ  $|V|$ , một giá trị  $d[v]$  vẫn có thể giảm xuống, điều đó chứng tỏ tồn tại một chu trình có tổng trọng số âm, khiến cho khái niệm "đường đi ngắn nhất" không còn tồn tại do tổng chi phí có thể giảm xuống âm vô cực bằng cách lặp lại chu trình đó.

**Nguyên lý relaxation(nối lỏng cạnh):**

Cốt lõi của cả hai thuật toán là phép toán **relaxation**: Cho cạnh  $e = (u, v)$  với trọng số  $w(u, v)$ . Gọi  $d[u], d[v]$  là ước lượng khoảng cách từ nguồn  $s$ . Phép relaxation thực hiện như sau:

Nếu  $d[v] > d[u] + w(u, v)$  thì cập nhật  $d[v] = d[u] + w(u, v)$ .

Phép toán này dựa trên bất đẳng thức tam giác trong lý thuyết đồ thị: khoảng cách trực tiếp đến  $v$  không bao giờ được lớn hơn khoảng cách đi vòng qua một đỉnh trung gian  $u$ .

## Chương 3. TỔ CHỨC CẤU TRÚC DỮ LIỆU VÀ THUẬT TOÁN

### 3.1 Phát biểu bài toán

Cho một đồ thị vô hướng hoặc có hướng  $G = (V, E)$ .

- **Dữ liệu vào (Input):**

- Số lượng đỉnh  $n$  và số lượng cạnh  $m$  của đồ thị.
- Danh sách các cạnh: mỗi cạnh được mô tả bởi ba giá trị  $(u, v, w)$  lần lượt là đỉnh xuất phát, đỉnh đích và trọng số  $w$  (chi phí).
- Đỉnh nguồn  $s$  – điểm xuất phát cần tìm đường đi ngắn nhất tới các đỉnh khác.
- Lựa chọn chức năng.

- **Kết quả (Output):**

- Mảng khoảng cách  $d[i]$  biểu diễn chi phí ngắn nhất từ  $s$  tới đỉnh  $i$ . Giá trị  $\infty$  biểu thị đỉnh không thể tới được.
- (Tùy chọn) Mảng đường đi trước  $\text{pred}[i]$  để tái dựng đường đi cụ thể.
- Thời gian thực thi và bộ nhớ sử dụng của thuật toán.

### 3.2 Cấu trúc dữ liệu

#### 3.2.1 Các thư viện được sử dụng trong chương trình

##### C++ Standard Library

Thư viện	Mục đích sử dụng
<code>iostream</code>	Nhập/xuất console.
<code>string</code>	Xử lý chuỗi.
<code>vector</code>	Mảng động.
<code>tuple</code>	Lưu cạnh $(u, v, w)$ .
<code>limits</code>	Giá trị INF.
<code>chrono</code>	Do thời gian chạy thuật toán.
<code>filesystem</code>	Kiểm tra/tạo thư mục, kiểm tra file tồn tại.
<code>fstream</code>	Đọc/ghi file.
<code>algorithm</code>	Các thao tác tìm kiếm/xử lý trong <code>vector</code> .
<code>queue</code>	<code>priority_queue</code> cho Dijkstra.
<code>sstream</code>	Ghép chuỗi/dịnh dạng văn bản (format).
<code>iomanip</code>	Căn lề/dịnh dạng khi in bảng.
<code>cmath</code>	<code>log</code> và các phép tính phục vụ phân tích độ phức tạp.
<code>cctype</code>	Kiểm tra ký tự nhập.
<code>cstdio</code>	Dùng <code>std::getchar()</code> trong graphics.

## Windows

Thư viện	Mô tả
windows.h, cwchar	Bộ thư viện của Windows; dùng để chuyển console sang UTF-8 và thiết lập phông chữ khi biên dịch trên Windows (chỉ áp dụng khi _WIN32 được định nghĩa).

## Các thư viện xây dựng nội bộ trong dự án

Header	Chức năng
graphics.h	Định nghĩa lớp “giả lập đồ họa” bằng ký tự ANSI, cho phép vẽ giao diện trực quan trong console mà không cần thư viện đồ họa ngoài.
Global.h	Khai báo các hằng số toàn cục như kích thước cửa sổ và đường dẫn file mặc định.
Colors.h	Định nghĩa mã màu để sử dụng khi in ra console.
GUI.h	Xây dựng giao diện người dùng, bao gồm menu và prompt tương tác.
Graph.h	Mô tả cấu trúc dữ liệu đồ thị và cung cấp hàm đọc/ghi file.
Algorithms.h	Khai báo và cài đặt các thuật toán Dijkstra và Bellman–Ford.
Comparison.h	Chứa các hàm đo thời gian và bộ nhớ, phục vụ so sánh hiệu năng giữa hai thuật toán.

## Thư viện Python (dùng cho chức năng trực quan hóa)

Thư viện	Mục đích sử dụng
networkx	Hỗ trợ tạo, quản lý và thao tác trên đồ thị.
matplotlib.pyplot	Vẽ đồ thị và biểu diễn trực quan kết quả.
os, sys	Kiểm tra file tồn tại, xử lý tham số dòng lệnh và mã thoát (exit code).

### 3.2.2 Các cấu trúc dữ liệu tự xây dựng

Trong dự án chúng em cũng có xây dựng thêm một số cấu trúc dữ liệu nhằm dễ dàng hơn trong quá trình thực hiện thuật toán và quản lý.

#### Cấu trúc dữ liệu Edge

**Mục đích:** biểu diễn cạnh trong đồ thị.

#### Trường dữ liệu:

- int destination – đỉnh đích.
- int weight – trọng số.

**Dùng trong:** Graph::adjList để lưu danh sách cạnh (danh sách kề).



Hình 3.1: Cấu trúc dữ liệu Edge

### Class Graph

**Mục đích:** lưu cấu trúc đồ thị và hỗ trợ I/O.

**Trường dữ liệu:**

- **int V** — số đỉnh.
- **int E** — số cạnh.
- **vector<vector<Edge>> adjList** — danh sách kề.
- **vector<string> vertexLabels** — nhãn của đỉnh.

**Dùng trong:** Graph::adjList để lưu danh sách cạnh (danh sách kề).

```
● ● ●
class Graph {
private:
    int V;
    int E;
    std::vector<std::vector<Edge>> adjList;
    std::vector<std::string> vertexLabels;
};
```

Hình 3.2: Cấu trúc dữ liệu Graph

### Cấu trúc dữ liệu PathResult

**Mục đích:** đóng gói lại các kết quả của thuật toán/ kết quả của chương trình.

**Trường dữ liệu:**

- **bool success** – trạng thái thực hiện thuật toán (Thành công/ Thất bại)
- **int startVertex** – đỉnh bắt đầu.
- **std::vector<int> distances** – khoảng cách từ đỉnh bắt đầu đến các đỉnh khác trong đồ thị.
- **std::vector<int> previousVertex** – mảng truy vết đường đi.
- **std::vector<int> shortestPath** – lưu đường đi ngắn nhất từ đỉnh bắt đầu đến đỉnh đích.
- **std::vector<std::string> logs** – lưu log thuật toán.
- **bool hasNegativeCycle** – phát hiện chu trình âm đối với thuật toán Bellman-Ford.

**Dùng trong:** Graph::adjList để lưu danh sách cạnh (danh sách kề).

```
● ● ●
struct PathResult {
    bool success;
    int startVertex;
    std::vector<int> distances;
    std::vector<int> previousVertex;
    std::vector<int> shortestPath;
    std::vector<std::string> logs;
    bool hasNegativeCycle;
};

PathResult() : success(false), startVertex(-1), hasNegativeCycle
(false) {}
```

Hình 3.3: Cấu trúc dữ liệu PathResult

### Cấu trúc dữ liệu PathResult

**Mục đích:** triển khai thuật toán Dijkstra và Bellman–Ford.

**Cấu trúc dữ liệu dùng nội bộ:**

- **std::priority\_queue<std::pair<int, int>, ...>** – hàng đợi ưu tiên dùng cho thuật toán dijkstra.
- **std::vector<bool> visited** – đánh dấu các đỉnh đã thăm.

**Liên kết:** giữ const Graph& graph để đọc dữ liệu đồ thị.

```
● ● ●
class Algorithms {
private:
    const Graph& graph;

    void logStep(std::vector<std::string> logs, const std::string& message);
    std::vector<int> reconstructPath(int destination, const std::vector<int>&
previousVertex) const;

public:
    explicit Algorithms(const Graph& g);

    PathResult dijkstra(int start, bool showSteps = false);

    PathResult bellmanFord(int start, bool showSteps = false);

    std::vector<int> getShortestPath(const PathResult& result, int destination) const;

    int getDistance(const PathResult& result, int destination) const;
};
```

Hình 3.4: Class Algorithms

### Cấu trúc dữ liệu PerformanceMetrics

**Mục đích:** lưu thông số hiệu năng của từng thuật toán.

#### Trường dữ liệu:

- std::string algorithmName
- long long executionTimeUs
- long long memoryUsageBytes
- int distancesCalculated
- double complexity
- bool success

```
struct PerformanceMetrics {
    std::string algorithmName;
    long long executionTimeUs;           // tinh bang micro giay
    long long memoryUsageBytes;          // byte
    int distancesCalculated;
    double complexity;                  // do phuc tap
    bool success;

    PerformanceMetrics() : algorithmName(""), executionTimeUs(0),
                           memoryUsageBytes(0), distancesCalculated(0),
                           complexity(0.0), success(false) {}

};
```

Hình 3.5: Cấu trúc dữ liệu PerformanceMetrics

### Cấu trúc dữ liệu ComparisonReport

**Mục đích:** tổng hợp dữ liệu tiến hành thực hiện so sánh hai thuật toán.

#### Trường dữ liệu:

- int startVertex
- int V, int E
- std::vector<PerformanceMetrics> metrics
- std::vector<std::string> logs – log kết quả để hiển thị.

```
struct ComparisonReport {
    int startVertex;
    int V; // số đỉnh
    int E; // cạnh
    std::vector<PerformanceMetrics> metrics;
    std::vector<std::string> logs;

    ComparisonReport() : startVertex(-1), V(0), E(0) {}

};
```

Hình 3.6: Cấu trúc dữ liệu ComparisonReport

### Class Comparison

**Mục đích:** tính toán hiệu năng của các thuật toán và đưa ra kết quả báo cáo.

#### Cấu trúc dữ liệu dùng nội bộ:

- PerformanceMetrics cho từng thuật toán.
- std::vector<std::string> để xuất bảng so sánh ra log.

Gửi const Graph& graph và Algorithms algorithms.

```
class Comparison {
private:
    const Graph& graph;
    Algorithms algorithms;

public:
    explicit Comparison(const Graph& g);

    ComparisonReport comparePerformance(int startVertex, AlgorithmType type =
AlgorithmType::BOTH);

    PerformanceMetrics measureAlgorithm(int startVertex, AlgorithmType type);
};
```

Hình 3.7: Class Comparison

### Class GUI

**Mục đích:** hiển thị menu, nhập dữ liệu từ người dùng, in log.

#### Cấu trúc dữ liệu chính:

- std::vector<std::string> – danh sách log hiển thị.
- std::vector<std::tuple<int, int, int>> – danh sách cạnh khi nhập đồ thị.

```
class GUI {
private:
    int screenWidth;
    int screenHeight;

public:
    GUI();
    ~GUI();
    void drawMenu();
    void drawComparisonScreen(const std::vector<std::string>& logs);
    void clearScreen();
    int promptMenuChoice();
    int promptGraphTitle(const std::string& title, const std::vector<std::tuple<int, int, int>>& edges, const std::string& prompt, const std::string& minValue, const std::string& maxValue);
    int promptFindTitle(const std::string& title, const std::string& prompt, const std::string& minValue, const std::string& maxValue);
    std::string promptString(const std::string& title, const std::string& prompt, const std::string& defaultValue = "");
    bool promptYesNo(const std::string& title, const std::string& prompt);
    void promptGraphInput(bool bidirected, int numVertices, int numEdges, std::vector<std::tuple<int, int, int>>& edges);
    void promptStartFind(int minValue, int maxValue, int startValue, int endValue);
    void showAlgorithmLogs(const std::string& title, const std::vector<std::string>& logs);
    void showUsage(const std::string& title, const std::vector<std::string>& lines);
    void waitForKey();
}
```

Hình 3.8: Class GUI

### 3.3 Thuật toán

#### 3.3.1 Thuật toán Dijkstra

##### Giới thiệu

Thuật toán Dijkstra là một trong những thuật toán nền tảng của lý thuyết đồ thị và khoa học máy tính, được Edsger W. Dijkstra xây dựng vào năm 1956 và công bố chính thức năm 1959 trong bài báo “*A note on two problems in connexion with graphs*”. Thuật toán này giải bài toán đường đi ngắn nhất từ một đỉnh nguồn (*single-source shortest path*) đến mọi đỉnh còn lại trong đồ thị có trọng số, áp dụng được cho cả đồ thị vô hướng lẫn đồ thị có hướng, với điều kiện quan trọng là trọng số cạnh không âm.

Thuật toán Dijkstra ngày nay không chỉ được sử dụng trong định tuyến mạng và hệ thống giao thông mà còn xuất hiện trong các bài toán lập kế hoạch robot, phân tích mạng xã hội, xử lý ngôn ngữ tự nhiên và nhiều lĩnh vực khác. Mặc dù có một số giới hạn như không xử lý được trọng số âm, nhưng với phạm vi áp dụng rộng và khả năng mở rộng linh hoạt, Dijkstra vẫn là một công cụ không thể thiếu trong kho tàng thuật toán đồ thị.

Trong phần này, thuật toán sẽ được trình bày chi tiết theo các nội dung: nguyên lý hoạt động, quy trình thực hiện (kèm ví dụ minh họa từng bước), ứng dụng và phân tích độ phức tạp theo các cách cài đặt phổ biến.

##### Nguyên lý hoạt động

Thuật toán hoạt động theo nguyên tắc tham lam: luôn chọn đỉnh có khoảng cách tạm thời nhỏ nhất (chưa được “chốt” kết quả) để mở rộng. Mỗi khi lựa chọn một đỉnh, thuật toán thực hiện nối lỏng cạnh (*relaxation*) các cạnh xuất phát từ đỉnh đó: nếu đi qua đỉnh hiện tại giúp giảm tổng chi phí tới đỉnh kề, ta cập nhật khoảng cách của đỉnh kề. Do mọi trọng số đều không âm, khi một đỉnh được chọn ra khỏi hàng đợi ưu tiên thì khoảng cách của nó đã là tối ưu và sẽ không bị cải thiện nữa. Để thực hiện việc lựa chọn nhanh, Dijkstra thường sử dụng hàng đợi ưu tiên (*priority queue*) – với đầu hàng đợi là đỉnh có khoảng cách tạm thời nhỏ nhất.

Hoạt động của thuật toán có thể tóm lược qua ba thành phần chủ chốt:

- **Khởi tạo:** gán giá trị vô cùng cho khoảng cách đến tất cả các đỉnh; riêng nguồn có khoảng cách 0. Các đỉnh ban đầu nằm trong tập chưa xét (*unvisited*).
- **Lựa chọn tham lam:** lặp lại cho tới khi không còn đỉnh chưa xét; ở mỗi vòng, chọn đỉnh chưa xét  $u$  có khoảng cách nhỏ nhất. Nếu đỉnh này có khoảng cách là vô cực, các đỉnh còn lại không thể tiếp cận và vòng lặp kết thúc.
- **Nối lỏng các cạnh kề:** đối với mọi đỉnh  $v$  kề  $u$ , tính khoảng cách tạm thời  $d[u] + w(u, v)$ ; nếu giá trị này nhỏ hơn  $d[v]$ , cập nhật

$$d[v] \leftarrow d[u] + w(u, v),$$

và lưu lại đường đi trước của  $v$  là  $u$ . Sau khi nối lỏng xong, loại  $u$  khỏi tập chưa xét, đánh dấu khoảng cách của  $u$  là cuối cùng.

Hoạt động lặp lại cho tới khi tất cả các đỉnh được xử lý. Thuật toán đảm bảo rằng mỗi đỉnh được xử lý đúng một lần và sau đó không cần xem xét lại nhờ trọng số không âm và cách chọn tham lam.

### Các bước thực hiện thuật toán

Dưới đây là phiên bản tóm tắt các bước thực hiện Dijkstra khi dùng danh sách kề và hàng đợi ưu tiên:

1. Khởi tạo mảng khoảng cách  $\text{dist}[]$  với giá trị  $\infty$  cho tất cả các đỉnh; đặt  $\text{dist}[s] \leftarrow 0$  cho đỉnh nguồn  $s$ . Khởi tạo mảng  $\text{pred}[]$  để lưu đường đi trước, gán  $\text{pred}[v] \leftarrow \text{undefined}$ .
2. Khởi tạo hàng đợi ưu tiên  $Q$  chứa cặp  $(\text{dist}[v], v)$  cho các đỉnh; hoặc đơn giản chỉ chèn  $(0, s)$  và chèn thêm các đỉnh khi phát hiện.
3. Lặp khi hàng đợi không rỗng: lấy đỉnh  $u$  có khoảng cách tạm thời nhỏ nhất (*extract-min*); nếu khoảng cách này lớn hơn giá trị đã lưu, bỏ qua.
4. nối lồng các cạnh kề: với mỗi đỉnh kề  $v$  của  $u$  có trọng số  $w(u, v)$ , nếu  $\text{dist}[u] + w(u, v) < \text{dist}[v]$ , cập nhật  $\text{dist}[v] \leftarrow \text{dist}[u] + w(u, v)$  và  $\text{pred}[v] \leftarrow u$ ; đồng thời thêm  $(\text{dist}[v], v)$  vào hàng đợi.
5. Lặp lại cho tới khi hàng đợi trống. Kết quả  $\text{dist}$  chứa khoảng cách ngắn nhất từ  $s$  đến mọi đỉnh;  $\text{pred}$  cho phép phục hồi đường đi.

### Pseudocode

```
function Dijkstra(Graph, source):
    for each vertex v in Graph:
        dist[v] <- INF
        pred[v] <- undefined
    dist[source] <- 0
    Q <- priority queue containing (0, source)
    while Q is not empty:
        (d, u) <- Q.extract_min()
        if d > dist[u]:
            continue
        for each neighbor v of u:
            alt <- dist[u] + weight(u, v)
            if alt < dist[v]:
                dist[v] <- alt
                pred[v] <- u
                Q.insert((dist[v], v))
    return dist, pred
```

### Phân tích độ phức tạp thời gian

Dộ phức tạp của thuật toán phụ thuộc vào cấu trúc dữ liệu dùng cho hàng đợi ưu tiên và biểu diễn đồ thị:

- **Triển khai đơn giản bằng mảng hoặc danh sách tuyến tính:** nếu sử dụng mảng để tìm đỉnh có khoảng cách nhỏ nhất (phép tìm *extract-min* có chi phí  $O(|V|)$ ), mỗi lần nới rộng cần tối đa  $|E|$  thao tác. Tổng thời gian là  $O(|V|^2)$ , thích hợp cho đồ thị dày. Trong trường hợp sử dụng danh sách tuyến tính và ma trận kề, độ phức tạp đơn giản nhất là  $\Theta(|V|^2)$ .
- **Dùng hàng đợi ưu tiên kiểu heap (binary heap hoặc pairing heap):** thao tác *extract-min* và *decrease-key* chạy trong thời gian logarit. Với biểu diễn danh sách kề, thuật toán đạt độ phức tạp

$$O((|V| + |E|) \log |V|),$$

nếu xét đồ thị liên thông và thừa thì thường viết gọn là  $O(|E| \log |V|)$ .

- **Dùng heap Fibonacci:** cải thiện thao tác *decrease-key* để đạt độ phức tạp

$$O(|E| + |V| \log |V|).$$

Nhiều nghiên cứu thực nghiệm cho thấy trên các đồ thị thừa, cài đặt dùng hàng đợi ưu tiên nhị phân hoặc pairing heap cho hiệu năng tốt và dễ triển khai.

### Phân tích bộ nhớ (phức tạp không gian)

Thuật toán Dijkstra yêu cầu lưu trữ:

- Mảng khoảng cách `dist[]` và mảng đường đi trước `pred[]` có kích thước  $|V|$ . Cả hai mảng chiếm không gian  $O(|V|)$ .
- Hàng đợi ưu tiên: chứa tối đa  $|V|$  đỉnh, nên chiếm  $O(|V|)$  không gian.
- Biểu diễn đồ thị: đối với danh sách kề, dung lượng lưu trữ là  $O(|V| + |E|)$ . Nếu sử dụng ma trận kề, dung lượng là  $O(|V|^2)$ .

Vì vậy, tổng bộ nhớ bổ sung của thuật toán (không kể dữ liệu đồ thị) là  $O(|V|)$  cho việc lưu khoảng cách, đường đi và hàng đợi ưu tiên. Trong các ứng dụng thực tế, bộ nhớ chủ yếu bị chi phối bởi cấu trúc lưu trữ đồ thị.

### Ứng dụng và ghi chú

Thuật toán Dijkstra có nhiều ứng dụng:

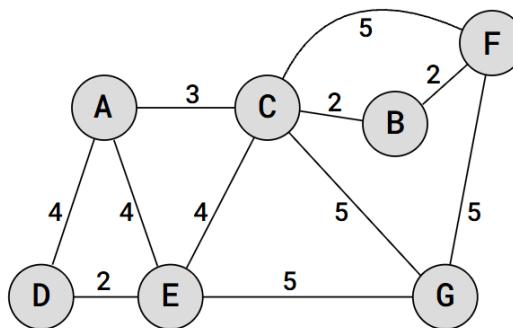
- **Định tuyến mạng:** giao thức OSPF và IS-IS sử dụng Dijkstra để tính toán đường dẫn tối ưu cho gói tin.
- **Hệ thống GPS và chỉ đường:** tìm đường đi ngắn nhất trên bản đồ có trọng số là thời gian hoặc khoảng cách.

- **Phân tích đồ thị:** tính khoảng cách ngắn nhất trong mạng xã hội, đồ thị giao thông, mạng lưới điện...
- **Mô phỏng và trò chơi:** tìm lối đi cho nhân vật trong game hoặc mô phỏng robot di chuyển.

Dijkstra là cơ sở cho nhiều thuật toán mở rộng, như A\* (sử dụng hàm heuristic để định hướng tìm kiếm) và định tuyến hai chiều (*bidirectional Dijkstra*) nhằm giảm thời gian tìm kiếm trên đồ thị lớn. Tuy nhiên, do giả sử trọng số không âm, thuật toán này không áp dụng được cho đồ thị có trọng số âm; khi đó, Bellman–Ford là lựa chọn phù hợp hơn.

### Ví dụ minh họa chi tiết

Giả sử ta có đồ thị vô hướng như hình bên dưới với 7 đỉnh và 11 cạnh. Nhãn của mỗi đỉnh được kí hiệu bằng các chữ cái in hoa từ  $A \rightarrow G$ . Ta bắt đầu từ đỉnh nguồn  $D$  và tìm đường đi ngắn nhất đến các đỉnh còn lại.



Hình 3.9: Đồ thị minh họa thuật toán Dijkstra

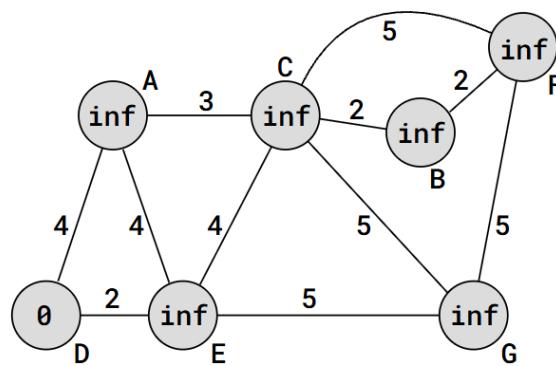
Để tìm đường đi ngắn nhất, thuật toán Dijkstra sử dụng một mảng chứa khoảng cách đến tất cả các đỉnh khác, và ban đầu đặt các khoảng cách này là vô cực. Và khoảng cách đến đỉnh mà ta bắt đầu (đỉnh nguồn) được đặt là 0.

```

distances = [inf, inf, inf, 0, inf, inf, inf]
#vertices = [ A,   B,   C,   D,   E,   F,   G]

```

Hình ảnh bên dưới cho thấy khoảng cách vô hạn ban đầu từ đỉnh xuất phát  $D$  đến các đỉnh khác. Giá trị khoảng cách đến đỉnh  $D$  là 0 vì đó là điểm bắt đầu.



Hình 3.10: Hình minh họa trạng thái ban đầu

Sau bước khởi tạo, thuật toán Dijkstra chọn đỉnh  $D$  làm đỉnh hiện tại (đỉnh chưa thăm có khoảng cách tạm thời nhỏ nhất). Từ  $D$ , thuật toán xét các đỉnh kề của  $D$  và thử cải thiện khoảng cách tạm thời đến chúng bằng cách đi qua  $D$ .

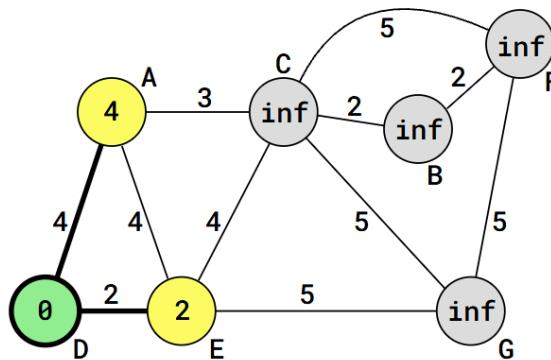
Vì ban đầu khoảng cách từ nguồn đến  $A$  và  $E$  đều là vô hạn (chưa tìm thấy đường đi), nên khi xét các cạnh từ  $D$ :

- Với cạnh  $D \rightarrow A$  có trọng số 4, khoảng cách mới đến  $A$  qua  $D$  là  $\text{dist}[D] + 4$ . Do  $\text{dist}[D] = 0$  (vì  $D$  là nguồn), ta có  $\text{dist}[A] = 4$ .
- Với cạnh  $D \rightarrow E$  có trọng số 2, tương tự  $\text{dist}[E] = \text{dist}[D] + 2 = 2$ .

Thao tác “thử cập nhật  $\text{dist}[v]$  bằng  $\text{dist}[u] + w(u, v)$  nếu nhỏ hơn” này chính là *relax* (nối lỏng) cạnh:

$$\text{nếu } \text{dist}[u] + w(u, v) < \text{dist}[v] \text{ thì cập nhật } \text{dist}[v]$$

và lưu  $\text{prev}[v] = u$  để truy vết đường đi.



Hình 3.11: Các bước cập nhật trọng số đường đi

Sau khi *nối lỏng* các đỉnh kề  $A$  và  $E$ , thuật toán đánh dấu  $D$  là đã thăm (*visited*). Từ thời điểm này, Dijkstra coi khoảng cách ngắn nhất từ nguồn đến  $D$  đã chắc chắn tối ưu, nên  $D$  sẽ không được xét lại nữa.

Tiếp theo, thuật toán cần chọn đỉnh hiện tại mới theo quy tắc:

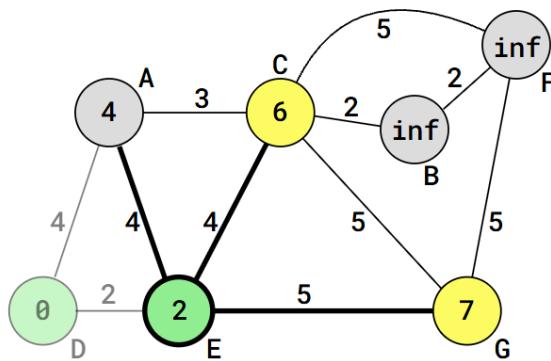
- Trong tất cả các đỉnh chưa thăm, chọn đỉnh có khoảng cách tạm thời  $\text{dist}[\cdot]$  nhỏ nhất tính từ đỉnh nguồn (ở đây là  $D$ ).

Ở bước này, sau khi cập nhật:

$$\text{dist}[A] = 4, \quad \text{dist}[E] = 2,$$

các đỉnh khác vẫn là  $\infty$ .

Vì  $E$  có khoảng cách tạm thời nhỏ nhất (2) trong nhóm chưa thăm, nên  $E$  được chọn làm đỉnh hiện tại ngay sau  $D$ .



Hình 3.12: Các bước cập nhật trọng số đường đi

Giờ thuật toán đặt  $E$  làm đỉnh hiện tại, nên ta sẽ *relax* các đỉnh kề của  $E$  mà chưa được thăm (tức là thử xem đi qua  $E$  có làm đường đi ngắn hơn không). Vì ta đã có  $\text{dist}[E] = 2$ , nên với mỗi đỉnh kề  $v$  của  $E$ , ta tính:

$$\text{alt} = \text{dist}[E] + w(E, v)$$

sau đó so sánh với  $\text{dist}[v]$  hiện tại.

- Đỉnh A:** nếu đi theo  $D \rightarrow E \rightarrow A$  thì

$$\text{alt} = 2 + 4 = 6.$$

Nhưng  $\text{dist}[A]$  hiện đang là 4 (nhỏ hơn 6), nên không cập nhật  $\text{dist}[A]$ .

- Đỉnh C:** đường  $D \rightarrow E \rightarrow C$  cho

$$\text{alt} = 2 + 4 = 6.$$

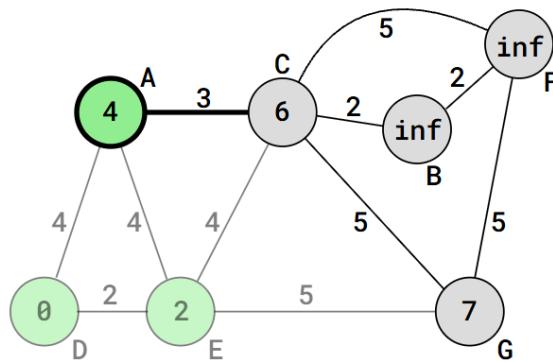
Vì  $\text{dist}[C]$  đang là  $\infty$ , nên cập nhật  $\text{dist}[C] = 6$ .

- Đỉnh G:** đường  $D \rightarrow E \rightarrow G$  cho

$$\text{alt} = 2 + 5 = 7.$$

Vì  $\text{dist}[G]$  đang là  $\infty$ , nên cập nhật  $\text{dist}[G] = 7$ .

Sau khi *relax* xong các đỉnh kề,  $E$  sẽ được đánh dấu đã thăm. Bước tiếp theo, thuật toán tiếp tục chọn đỉnh chưa thăm có  $\text{dist}[\cdot]$  nhỏ nhất. Trong các đỉnh chưa thăm lúc này,  $A$  có  $\text{dist}[A] = 4$  là nhỏ nhất, nên  $A$  sẽ là đỉnh tiếp theo được chọn để thăm.



Hình 3.13: Các bước cập nhật trọng số đường đi

Khi  $A$  được chọn làm đỉnh hiện tại, thuật toán sẽ *relax* các đỉnh kề của  $A$  mà chưa được thăm.

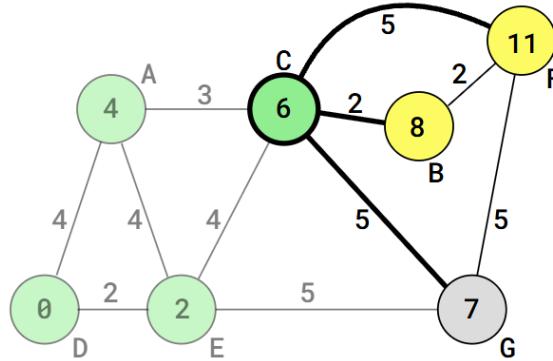
- **Với đỉnh  $C$ :** nếu đi theo đường  $D \rightarrow A \rightarrow C$  thì khoảng cách tạm mới là

$$\text{alt} = \text{dist}[A] + w(A, C) = 4 + 3 = 7.$$

Nhưng trước đó ta đã có  $\text{dist}[C] = 6$  (từ đường  $D \rightarrow E \rightarrow C$ ), nhỏ hơn 7. Vì vậy không cập nhật khoảng cách đến  $C$  (giữ nguyên 6).

Sau khi xét xong các láng giềng,  $A$  được đánh dấu là đã thăm (*visited*).

Tiếp theo, thuật toán chọn đỉnh chưa thăm có  $\text{dist}[\cdot]$  nhỏ nhất. Trong số các đỉnh còn lại chưa thăm,  $C$  đang có  $\text{dist}[C] = 6$  là nhỏ nhất, nên  $C$  sẽ là đỉnh tiếp theo được thăm.



Hình 3.14: Các bước cập nhật trọng số đường đi

Khi  $C$  được chọn làm đỉnh hiện tại (vì  $\text{dist}[C] = 6$ ), thuật toán sẽ *relax* các đỉnh kề của  $C$  mà chưa được thăm, tức là thử cập nhật khoảng cách đến chúng bằng cách đi qua  $C$ .

- **Đỉnh  $F$ :** khoảng cách mới theo đường  $D \rightarrow \dots \rightarrow C \rightarrow F$  là

$$\text{dist}[C] + w(C, F) = 6 + 5 = 11.$$

nên cập nhật  $\text{dist}[F] = 11$ .

- **Đỉnh  $B$ :** khoảng cách mới theo đường  $D \rightarrow \dots \rightarrow C \rightarrow B$  là

$$\text{dist}[C] + w(C, B) = 6 + 2 = 8.$$

nên cập nhật  $\text{dist}[B] = 8$ .

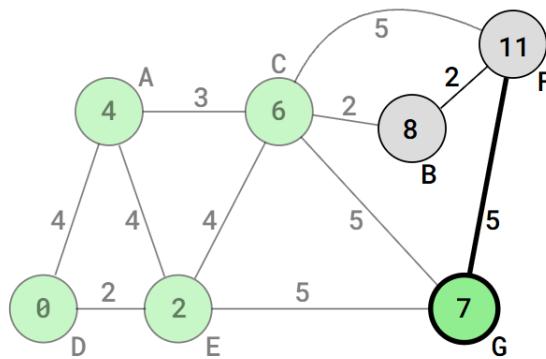
- **Đỉnh G:** nếu đi qua C thì khoảng cách là

$$\text{dist}[C] + w(C, G) = 6 + 5 = 11.$$

Nhưng trước đó đã có  $\text{dist}[G] = 7$  (ngắn hơn), nên không cập nhật khoảng cách đến G.

Sau khi xét xong các láng giềng, C được đánh dấu là đã thăm (*visited*).

Tiếp theo, thuật toán chọn đỉnh chưa thăm có  $\text{dist}[\cdot]$  nhỏ nhất. Trong các đỉnh còn lại chưa thăm, G có khoảng cách nhỏ nhất (7), nên G sẽ là đỉnh tiếp theo được thăm.



Hình 3.15: Các bước cập nhật trọng số đường đi

Khi G được chọn làm đỉnh hiện tại (vì  $\text{dist}[G] = 7$ ), thuật toán xét các đỉnh kề của G mà chưa được thăm và thử cập nhật khoảng cách đến chúng qua G.

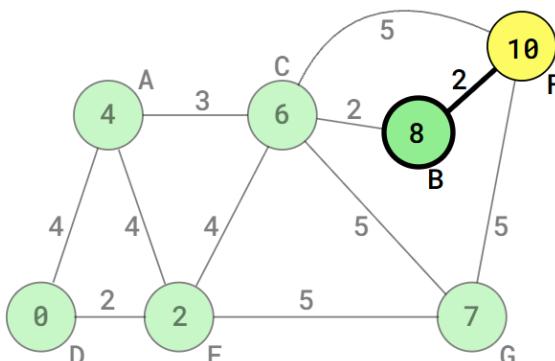
- **Đỉnh F:** hiện đã có  $\text{dist}[F] = 11$ . Nếu đi theo đường  $D \rightarrow \dots \rightarrow G \rightarrow F$  thì khoảng cách mới là

$$\text{dist}[G] + w(G, F) = 7 + 5 = 12.$$

Vì 12 lớn hơn 11, nên không cập nhật khoảng cách đến F.

Sau khi hoàn tất việc *relax*, G được đánh dấu là đã thăm (*visited*).

Tiếp theo, thuật toán chọn đỉnh chưa thăm có  $\text{dist}[\cdot]$  nhỏ nhất. Trong các đỉnh chưa thăm còn lại, B có khoảng cách nhỏ nhất (8), nên B trở thành đỉnh hiện tại.



Hình 3.16: Các bước cập nhật trọng số đường đi

Khi  $B$  được chọn làm đỉnh hiện tại (vì  $\text{dist}[B] = 8$ ), thuật toán sẽ xét các đỉnh kề của  $B$  mà chưa được thăm và thử cập nhật khoảng cách đến chúng bằng cách đi qua  $B$ .

- **Đỉnh  $F$ :** khoảng cách mới theo đường  $D \rightarrow \dots \rightarrow B \rightarrow F$  là

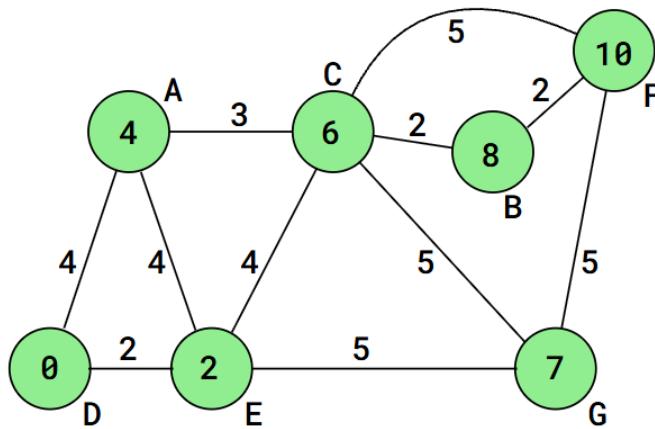
$$\text{dist}[B] + w(B, F) = 8 + 2 = 10.$$

Vì 10 nhỏ hơn khoảng cách hiện tại  $\text{dist}[F] = 11$ , nên cập nhật  $\text{dist}[F] = 10$ .

Sau khi *relax* xong,  $B$  được đánh dấu là đã thăm (*visited*).

Lúc này chỉ còn  $F$  là đỉnh chưa thăm cuối cùng. Vì không còn đỉnh nào khác cần cập nhật thông qua  $F$  (hoặc việc xét thêm cũng không làm thay đổi kết quả khoảng cách ngắn nhất đến các đỉnh đã “chốt”), nên có thể coi thuật toán kết thúc.

### Kết quả



Hình 3.17: Kết quả cuối cùng của thuật toán

Kết quả của Dijkstra là: mỗi đỉnh được chọn làm “đỉnh hiện tại” đúng một lần, và thu được khoảng cách ngắn nhất từ đỉnh nguồn  $D$  đến mọi đỉnh còn lại trong đồ thị.

### 3.3.2 Thuật toán Bellman-Ford

#### Giới thiệu

Thuật toán Bellman–Ford là một trong những thuật toán cổ điển nhất cho bài toán tìm đường đi ngắn nhất từ một đỉnh nguồn đến tất cả các đỉnh còn lại (*single-source shortest path problem – SSSP*) trong đồ thị có trọng số. Về nguồn gốc, ý tưởng của phương pháp này xuất hiện từ các nghiên cứu của Alfonso Shimbel (1955); sau đó Lester Ford Jr. giới thiệu rõ ràng hơn vào năm 1956, và Richard Bellman giúp thuật toán được biết đến rộng rãi qua các công bố vào năm 1958.

Điểm mạnh cốt lõi của Bellman–Ford nằm ở cách tiếp cận “lan truyền dần” bằng thao tác *relax* (nới lỏng) cạnh: thuật toán lặp lại việc xét tất cả các cạnh nhiều lần để cập nhật dần khoảng cách ngắn nhất. Nhờ đó, Bellman–Ford **xử lý được cạnh có trọng số âm** — điều mà Dijkstra không đảm bảo đúng. Không chỉ vậy, thuật toán còn có thể phát hiện chu trình âm: nếu sau khi đã lặp đủ số vòng chuẩn mà vẫn còn cập nhật làm khoảng cách giảm, điều đó cho thấy tồn tại một chu trình có tổng trọng số âm (có thể đi tới từ nguồn), và khi ấy “đường đi ngắn nhất” không còn xác định hữu hạn.

Trong thực tế, Bellman–Ford có giá trị lớn trong các bài toán cần “cập nhật dần theo lan truyền thông tin”, đặc biệt là định tuyến mạng theo kiểu *distance-vector* (mỗi nút cập nhật ước lượng chi phí dựa trên thông tin từ láng giềng). Ngoài ra, khả năng phát hiện chu trình âm cũng hữu ích trong các mô hình tối ưu hóa nơi “vòng lặp có lợi” có thể được biểu diễn như chu trình âm (ví dụ một số bài toán phân tích chênh lệch, lợi nhuận, hoặc ràng buộc chi phí).

#### Nguyên lý hoạt động

Bellman–Ford được xây dựng trên cơ sở **quy hoạch động**. Nó thực hiện quy trình **nới lỏng** (*relaxation*) lặp lại trên các cạnh: nếu khoảng cách tạm thời tối đỉnh  $u$  là  $d[u]$  và cạnh  $(u, v)$  có trọng số  $w(u, v)$ , thì nếu

$$d[u] + w(u, v) < d[v],$$

ta cập nhật

$$d[v] = d[u] + w(u, v).$$

Nguyên lý là đường đi ngắn nhất từ nguồn tới một đỉnh bất kỳ luôn chứa không quá  $|V| - 1$  cạnh (với  $|V|$  là số đỉnh). Do đó, sau khi nới lỏng toàn bộ các cạnh  $|V| - 1$  lần, mọi khoảng cách tối ưu đều đã được cập nhật. Những điểm cốt lõi của phương pháp này được trình bày ở dạng liệt kê:

- Thuật toán đặt khoảng cách tới nguồn bằng 0 và tới các đỉnh khác bằng  $\infty$ .
- Lặp lại nới lỏng tất cả các cạnh đúng  $|V| - 1$  lần – tương ứng với số cạnh tối đa của đường đi ngắn nhất. Mỗi lần nới lỏng giúp tìm các đường đi ngắn hơn thông qua đỉnh trung gian.
- Sau khi hoàn thành  $|V| - 1$  lần nới lỏng, thuật toán kiểm tra thêm một vòng: nếu còn cạnh nào có thể được nới lỏng thì đồ thị chứa **chu trình âm**. Khi đó không tồn

tại đường đi ngắn nhất vì ta có thể lặp chu trình âm để làm giảm chi phí vô hạn.

Khác với Dijkstra – vốn tham lam chọn đỉnh có khoảng cách tạm thời nhỏ nhất – Bellman–Ford không lựa chọn đỉnh cụ thể mà cập nhật toàn bộ các cạnh nhiều lần. Điều này bảo đảm tìm được khoảng cách tối ưu ngay cả khi các trọng số âm tồn tại. Ngoài ra, khả năng phát hiện chu trình âm khiến thuật toán trở nên đáng tin cậy trong các ứng dụng cần kiểm tra tính ổn định của mạng lưới.

### Các bước thực hiện thuật toán

Thuật toán có thể mô tả theo các bước cụ thể sau đây:

#### 1. Khởi tạo:

- Gán  $d[\text{source}] = 0$  và  $d[v] = \infty$  cho mọi đỉnh  $v$  khác.
- Tạo mảng `previousVertex[]` để lưu đỉnh trước đó trên đường đi ngắn nhất.

#### 2. Nối lồng các cạnh:

Lặp  $|V| - 1$  lần:

- Với mỗi cạnh  $(u, v)$  có trọng số  $w(u, v)$ , nếu  $d[u] + w(u, v) < d[v]$  thì:
  - Cập nhật  $d[v] = d[u] + w(u, v)$ .
  - Gán `previousVertex[v] = u`

#### 3. Kiểm tra chu trình âm:

- Duyệt lại tất cả các cạnh.
- Nếu tồn tại cạnh  $(u, v)$  sao cho  $d[u] + w(u, v) < d[v]$ , thì phát hiện chu trình âm và kết luận không tồn tại đường đi ngắn nhất.

### Pseudocode

```

function BellmanFord(G, source):
    // G: danh sách các cạnh (u, v, w), V: số đỉnh
    for v in vertices:
        dist[v] ← INF
        pred[v] ← NIL
    dist[source] ← 0

    // Lặp V-1 lần
    for i from 1 to V-1:
        for each edge (u, v, w) in G:
            if dist[u] + w < dist[v]:
                dist[v] ← dist[u] + w
                pred[v] ← u

    // Kiểm tra chu trình âm
    for each edge (u, v, w) in G:

```

```

if dist[u] + w < dist[v]:
    report "Phát hiện chu trình âm"
    return

return dist, pred

```

### Phân tích độ phức tạp

#### Thời gian

Trong cài đặt cơ bản, thuật toán sử dụng hai vòng lặp lồng nhau: vòng ngoài chạy  $|V| - 1$  lần và vòng trong duyệt tất cả  $|E|$  cạnh. Do đó, độ phức tạp thời gian của Bellman–Ford là:

- **Trường hợp xấu nhất và trung bình:**  $O(|V| \cdot |E|)$  – cần nối lồng toàn bộ các cạnh cho mỗi lần lặp.
- **Trường hợp tốt nhất:** khi không cần nối lồng cạnh (khoảng cách ban đầu đã tối ưu), thuật toán có thể dừng sau một lần duyệt qua các cạnh; khi đó thời gian là  $O(|E|)$ .

Thuật toán chậm hơn Dijkstra (độ phức tạp  $O(|E| \log |V|)$ ) nhưng trở nên cần thiết khi trong đồ thị có trọng số âm hoặc cần phát hiện chu trình âm.

#### Bộ nhớ

Bellman–Ford cần lưu một mảng `dist[]` kích thước  $|V|$  để lưu khoảng cách tạm thời và một mảng `pred[]` để lưu đỉnh trước đó. Danh sách cạnh của đồ thị được cung cấp sẵn hoặc xây dựng từ danh sách kề. Do đó, độ phức tạp không gian bổ sung là  $O(|V|)$  – tuyến tính theo số lượng đỉnh. Không cần cấu trúc dữ liệu phức tạp như hàng đợi ưu tiên; thuật toán chỉ cần mảng các cạnh và mảng khoảng cách.

### Ý tưởng tối ưu Bellman–Ford bằng SPFA (Shortest Path Faster Algorithm)

Bellman–Ford thực hiện  $|V| - 1$  lượt nối lồng mọi cạnh để tìm đường đi ngắn nhất; điều này tốn kém khi đồ thị lớn nhưng chỉ một số đỉnh thực sự bị cập nhật. SPFA khắc phục bằng cách chỉ đưa vào hàng đợi những đỉnh có khoảng cách vừa được cải thiện. Khi hàng đợi rỗng, tức là không còn cạnh nào cần nối lồng, thuật toán dừng. Cách làm này dựa trên quan sát rằng phần lớn thao tác nối lồng trong Bellman–Ford không dẫn đến cập nhật nên có thể bỏ qua.

#### Nguyên lý và tổ chức hàng đợi

- **Hàng đợi vòng (*circular queue*):** SPFA thường sử dụng mảng vòng với hai con trỏ `front` và `rear` để thực hiện các thao tác `push/pop` trong  $O(1)$ . Việc này tránh chi phí phân bổ động như trong `std::queue`.
- **Mảng `dist[]` và `inQueue[]`:** Giống Bellman–Ford, `dist[v]` lưu khoảng cách tạm thời, còn `inQueue[v]` đánh dấu đỉnh đang nằm trong hàng đợi để tránh thêm trùng lặp.

- **Đếm số lần vào hàng đợi:** Để phát hiện chu trình âm, SPFA đếm số lần mỗi đỉnh bị đưa vào hàng đợi ( $\text{cnt}[v]$ ). Nếu một đỉnh bị đưa vào hàng đợi hơn  $|V|$  lần thì tồn tại chu trình âm.

### Các bước thực hiện

1. **Khởi tạo:** gán  $\text{dist}[s] = 0$  cho nguồn  $s$ , các đỉnh khác bằng  $\infty$ ; đặt  $\text{inQueue}[s] = \text{true}$ ; đưa  $s$  vào hàng đợi vòng.
2. **Lặp cho tới khi hàng đợi rỗng:**
  - Lấy đỉnh  $u$  ra khỏi đầu hàng đợi, đặt  $\text{inQueue}[u] = \text{false}$ .
  - Duyệt tất cả các cạnh  $(u, v, w)$ . Nếu  $\text{dist}[u] + w < \text{dist}[v]$  thì cập nhật  $\text{dist}[v]$  và, nếu  $v$  chưa có trong hàng đợi, đưa  $v$  vào cuối (hoặc đầu nếu dùng kỹ thuật *Small Label First*) và đặt  $\text{inQueue}[v] = \text{true}$ .
  - Tăng  $\text{cnt}[v]$  và nếu  $\text{cnt}[v] > |V|$  thì dừng và báo có chu trình âm.

Hàng đợi vòng giúp các thao tác thêm/xóa diễn ra tuần tự mà không cần dịch chuyển phần tử như trong mảng thường; vì đầu và cuối “nối tròn”, khi con trỏ `rear` vượt quá kích thước thì quay lại vị trí đầu mảng.

### Độ phức tạp và đặc tính

- **Trung bình:** SPFA chạy nhanh hơn Bellman–Ford vì chỉ nối lồng các cạnh “có ích”. Trên nhiều đồ thị thực nghiệm, thuật toán đạt gần  $O(|E|)$  thời gian.
- **Tệ nhất:** SPFA vẫn có thể thoái hóa về  $O(|V||E|)$ , giống Bellman–Ford. Những đồ thị có cấu trúc đặc biệt (ví dụ chu trình hướng với trọng số dương) sẽ khiến hầu hết các đỉnh luôn được đưa trở lại hàng đợi.
- **Bộ nhớ:** cần  $O(|V|)$  cho mảng khoảng cách và mảng đánh dấu; ngoài ra hàng đợi vòng có kích thước tối đa  $|V|$ .

### Cách phát hiện chu trình âm

Thuật toán Bellman–Ford thực hiện việc **nối lồng cạnh** theo nguyên tắc quy hoạch động: lặp lại quá trình cập nhật khoảng cách cho tất cả các cạnh trong  $|V| - 1$  lần (với  $|V|$  là số đỉnh) để đảm bảo rằng mọi đường đi đơn giản có tối đa  $|V| - 1$  cạnh đều được xét. Tuy nhiên, để kiểm tra chu trình âm, ta thực hiện thêm một vòng quét nữa qua tất cả các cạnh. Nếu tồn tại một cạnh  $(u, v)$  thỏa mãn điều kiện

$$\text{dist}[u] + w(u, v) < \text{dist}[v],$$

nghĩa là việc nối lồng cạnh này vẫn tiếp tục giảm được khoảng cách tới  $v$  sau  $|V| - 1$  lần lặp, điều đó chỉ có thể xảy ra khi tồn tại một **chu trình âm** trong đồ thị. Khi đó, thuật toán sẽ đánh dấu có chu trình âm và kết thúc, vì không tồn tại khoảng cách ngắn nhất hữu hạn.

Thuật toán có thể trả về mảng khoảng cách với giá trị đặc biệt (ví dụ  $-1$ ) hoặc thông báo lỗi để chỉ ra rằng đồ thị chứa chu trình âm và việc tính đường đi ngắn nhất không khả thi.

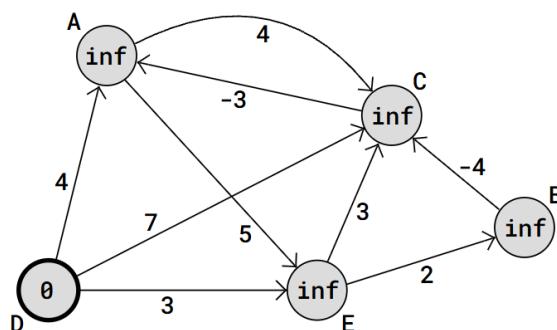
*Thuật toán Bellman-Ford là thuật toán tìm đường đi ngắn nhất từ một điểm xuất phát duy nhất. Nó hoạt động hiệu quả ngay cả trong trường hợp có cạnh âm và cũng có khả năng phát hiện chu trình âm. Thuật toán này hoạt động dựa trên nguyên tắc nối lồng các cạnh.*

### Ứng dụng

- **Giao thức định tuyến (networking protocols)** – Phiên bản phân tán của Bellman–Ford là nền tảng của các giao thức định tuyến theo vector khoảng cách như *Routing Information Protocol (RIP)*. Mỗi router tính khoảng cách tới các router khác, gửi bảng khoảng cách cho hàng xóm và cập nhật lại khi nhận thông tin mới.
- **Phát hiện cơ hội arbitrage** – Trong lĩnh vực tài chính, Bellman–Ford có thể phát hiện chu trình âm bằng cách chuyển đổi tỷ giá hối đoái thành logarit âm. Nếu tồn tại chu trình âm, đó là dấu hiệu cho cơ hội *arbitrage* (mua bán vòng quanh để thu lợi).
- **Hệ thống định tuyến giao thông và GPS** – Thuật toán được áp dụng trong các hệ thống lập lộ trình có cân nhắc tối thiểu yếu tố như kẹt xe và điều kiện đường sá, nơi trọng số (thời gian/chi phí) có thể thay đổi và đổi khi mang giá trị âm do ưu đãi.
- **Sinh học tính toán và game** – Bellman–Ford được sử dụng trong các bài toán xếp chuỗi (*sequence alignment*), mô phỏng protein hoặc tìm đường đi trong môi trường game phức tạp với chi phí di chuyển khác nhau.

### Ví dụ minh họa chi tiết

Giả sử ta có đồ thị có hướng như hình bên dưới với 5 đỉnh và 8 cạnh. Nhãn của mỗi đỉnh được ký hiệu bằng các chữ cái in hoa từ  $A \rightarrow E$ . Với đỉnh bắt đầu là  $D$ .

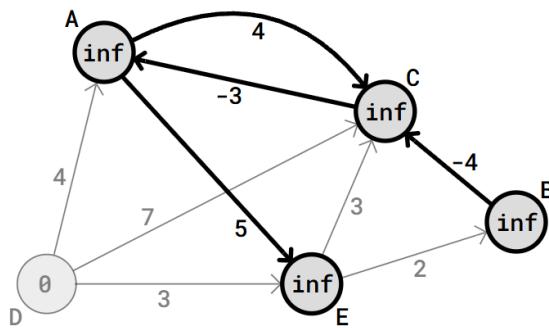


Hình 3.18: Hình minh họa thuật toán Bellman-Ford

Khởi tạo khoảng cách ban đầu.

```
distances = [inf, inf, inf, 0, inf]
#vertices = [ A,     B,     C,     D,     E]
```

Bốn cạnh đầu tiên được xét trong đồ thị là  $A \rightarrow C$ ,  $A \rightarrow E$ ,  $B \rightarrow C$  và  $C \rightarrow A$ . Tuy nhiên, ở thời điểm này các đỉnh xuất phát  $A$ ,  $B$  và  $C$  vẫn có khoảng cách tạm thời vô hạn (chưa có đường đi nào từ đỉnh nguồn đến chúng), nên khi thử *relax* các cạnh trên, ta không thể tạo ra một khoảng cách hữu hạn mới cho các đỉnh đích. Vì vậy, **không có cập nhật nào** xảy ra trong bốn lần kiểm tra cạnh đầu tiên này.



Hình 3.19: Hình minh họa thuật toán Bellman-Ford

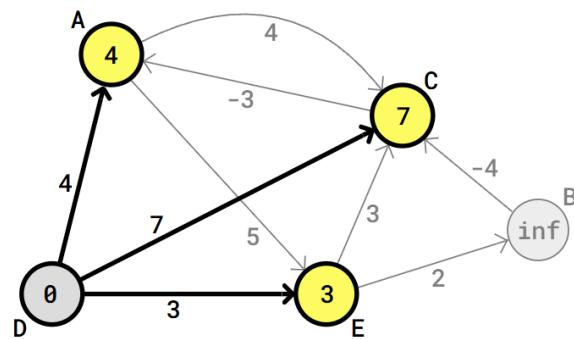
Sau khi đã thử *relax* các cạnh xuất phát từ  $A$ ,  $B$ ,  $C$  mà vẫn không có cập nhật (vì các đỉnh này đang có  $\text{dist} = \infty$ ), thuật toán chuyển sang kiểm tra các cạnh đi ra từ đỉnh  $D$ .

Vì khoảng cách tạm thời của  $D$  bằng 0 ( $\text{dist}[D] = 0$ ), nên khi *relax* từng cạnh  $D \rightarrow v$ , ta có:

$$\text{dist}[v] \leftarrow \min(\text{dist}[v], \text{dist}[D] + w(D, v)) = \min(\infty, 0 + w(D, v)) = w(D, v).$$

Do đó, các đỉnh kề mà  $D$  có cạnh đi ra sẽ được cập nhật khoảng cách đúng bằng trọng số của cạnh:

- $D \rightarrow A$  (trọng số 4)  $\Rightarrow \text{dist}[A] = 4$ .
- $D \rightarrow E$  (trọng số 3)  $\Rightarrow \text{dist}[E] = 3$ .
- $D \rightarrow C$  (trọng số 7)  $\Rightarrow \text{dist}[C] = 7$ .



Hình 3.20: Hình minh họa thuật toán Bellman-Ford

Sau khi các cạnh đi ra từ  $D$  đã làm cho một số đỉnh có khoảng cách hữu hạn (đặc biệt là  $\text{dist}[E] = 3$ ), thuật toán tiếp tục kiểm tra các cạnh đi ra từ  $E$ . Vì  $E$  đã có khoảng cách hữu hạn, nên việc *relax* các cạnh từ  $E$  có thể tạo ra cập nhật mới cho các đỉnh kề:

- **Với cạnh  $E \rightarrow B$  (trọng số 2):**

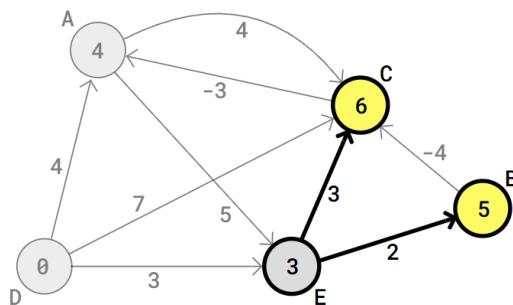
$$\text{dist}[B] \leftarrow \min(\text{dist}[B], \text{dist}[E] + 2) = \min(\infty, 3 + 2) = 5.$$

$\Rightarrow$  cập nhật trọng số  $\text{dist}[B] = 5$ .

- **Với cạnh  $E \rightarrow C$  (trọng số 3):**

$$\text{dist}[C] \leftarrow \min(\text{dist}[C], \text{dist}[E] + 3) = \min(7, 3 + 3) = 6.$$

$\Rightarrow$  cập nhật trọng số  $\text{dist}[C] = 6$



Hình 3.21: Hình minh họa hóa thuật toán Bellman-Ford

Sau khi đã duyệt toàn bộ các cạnh được 1 lượt (vòng lặp thứ nhất), thuật toán vẫn phải lặp đủ  $|V| - 1$  lượt. Với đồ thị có 5 đỉnh ( $A, B, C, D, E$ ) thì tổng số lượt là

$$5 - 1 = 4.$$

Vì vậy, sau lượt đầu tiên, thuật toán còn phải kiểm tra thêm 3 lượt nữa trước khi kết thúc.

Bước sang lượt kiểm tra thứ hai, thuật toán bắt đầu lại từ đầu và xét các cạnh đi ra từ đỉnh  $A$ . Tại thời điểm này ta đang có:

$$\text{dist}[D] = 0, \quad \text{dist}[A] = 4, \quad \text{dist}[E] = 3, \quad \text{dist}[B] = 5, \quad \text{dist}[C] = 6.$$

Khi *relax* các cạnh từ  $A$ :

- **Với cạnh  $A \rightarrow C$  (trọng số 4):**

$$\text{alt} = \text{dist}[A] + 4 = 4 + 4 = 8.$$

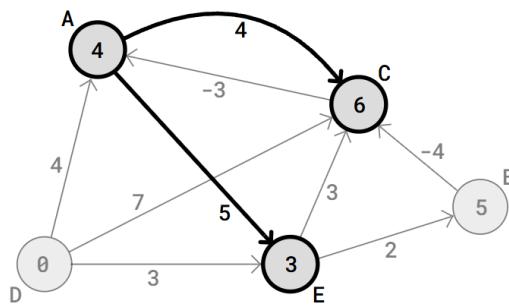
Trong khi đó  $\text{dist}[C] = 6$  đã nhỏ hơn 8, nên không cập nhật khoảng cách đến  $C$ .

- **Với cạnh  $A \rightarrow E$  (trọng số 5):**

$$\text{alt} = \text{dist}[A] + 5 = 4 + 5 = 9.$$

Nhưng  $\text{dist}[E] = 3$  đã nhỏ hơn 9, nên không cập nhật khoảng cách đến  $E$ .

Vì cả hai giá trị tính được đều không tốt hơn khoảng cách hiện tại, nên việc kiểm tra các cạnh  $A \rightarrow C$  và  $A \rightarrow E$  trong lượt thứ hai không tạo ra thay đổi nào. Sau đó, thuật toán sẽ tiếp tục kiểm tra các cạnh còn lại theo thứ tự đã định trong lượt kiểm tra này.



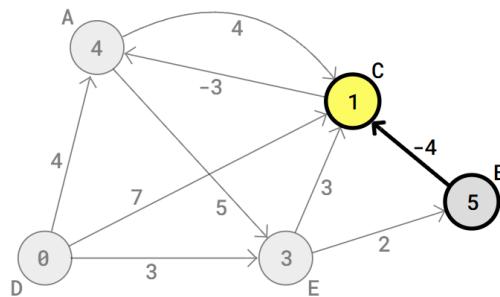
Hình 3.22: Hình minh họa thuật toán Bellman-Ford

Cạnh tiếp theo được xét là  $B \rightarrow C$ . Ở thời điểm này,  $\text{dist}[B] = 5$  và trọng số cạnh  $w(B, C) = -4$ , nên khi *relax* ta tính:

$$\text{alt} = \text{dist}[B] + w(B, C) = 5 + (-4) = 1.$$

Vì 1 nhỏ hơn khoảng cách hiện tại  $\text{dist}[C] = 6$ , nên Bellman-Ford cập nhật:

$$\text{dist}[C] = 1$$



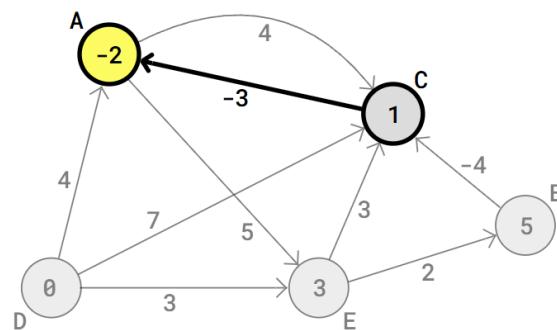
Hình 3.23: Hình minh họa thuật toán Bellman-Ford

Khi xét cạnh  $C \rightarrow A$  với trọng số  $-3$ , tại thời điểm này ta vừa có  $\text{dist}[C] = 1$  nên khoảng cách ứng viên đến  $A$  qua  $C$  là:

$$\text{alt} = \text{dist}[C] + w(C, A) = 1 + (-3) = -2.$$

Vì  $-2$  nhỏ hơn khoảng cách hiện tại  $\text{dist}[A] = 4$ , nên thuật toán cập nhật:

$$\text{dist}[A] = -2.$$



Hình 3.24: Hình minh họa thuật toán Bellman-Ford

Trong vòng lặp thứ 2, cạnh  $C \rightarrow A$  đúng là “cú cập nhật cuối cùng” đối với đồ thị này: sau khi  $\text{dist}[C]$  giảm xuống 1, việc  $\text{relax } C \rightarrow A$  với trọng số  $-3$  làm  $\text{dist}[A]$  tiếp tục giảm, và từ thời điểm đó trở đi không còn cạnh nào tạo ra giá trị nhỏ hơn nữa. Vì vậy, dù thuật toán vẫn phải chạy tiếp thêm 2 vòng (để đủ tổng cộng  $|V| - 1$  vòng), các vòng sau chỉ là kiểm tra xác nhận, không phát sinh cập nhật.

Việc Bellman–Ford phải kiểm tra tất cả các cạnh  $|V| - 1$  lần thoát nhìn có vẻ nhiều, nhưng đây là điểm bảo đảm tính đúng đắn:

- Mỗi vòng *relax* toàn bộ cạnh có thể “lan truyền” thông tin đường đi ngắn thêm một bước cạnh.
- Sau  $k$  vòng, thuật toán đã xét đầy đủ các đường đi có không quá  $k$  cạnh.
- Nếu đồ thị không có chu trình âm, thì mọi đường đi ngắn nhất hữu hạn luôn có thể biểu diễn bởi một đường đi đơn với tối đa  $|V| - 1$  cạnh.

Vì vậy, chạy đủ  $|V| - 1$  vòng sẽ đảm bảo rằng, dù đường đi tối ưu dài và phải “truyền” qua nhiều đỉnh trung gian, thuật toán vẫn tìm được khoảng cách ngắn nhất đến tất cả các đỉnh. “

### 3.3.3 Tổng kết

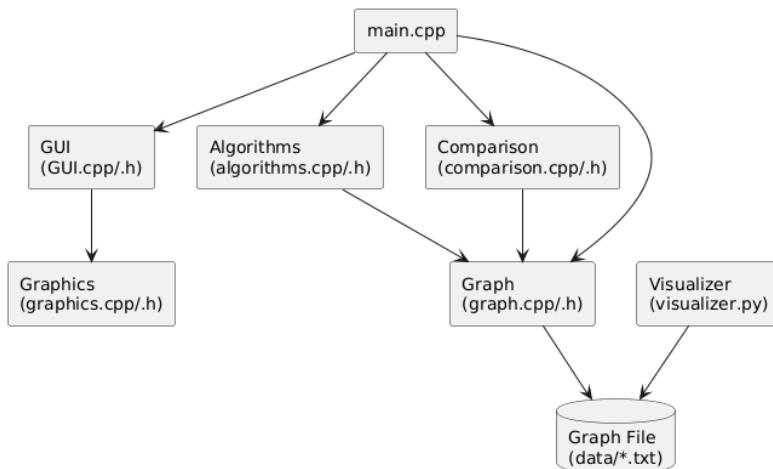
Bảng 3.1: So sánh Dijkstra và Bellman–Ford

Tiêu chí	Dijkstra	Bellman–Ford
<b>Loại bài toán</b>	SSSP	SSSP
<b>Điều kiện trọng số</b>	$w(e) \geq 0$	Có thể âm; phát hiện chu trình âm
<b>Ý tưởng chính</b>	Tham lam + hàng đợi ưu tiên	Quy hoạch động + nối lồng toàn bộ cạnh
<b>Độ phức tạp thời gian</b>	$O(( V  +  E ) \log  V )$	$O( V  E )$
<b>Bộ nhớ (bổ sung)</b>	$O( V )$	$O( V )$
<b>Đồ thị thừa / lớn</b>	Rất phù hợp	Thường chậm
<b>Trọng số âm</b>	Không dùng được	Dùng được
<b>Chu trình âm</b>	Không xác định được	Xác định được

## Chương 4. CHƯƠNG TRÌNH VÀ KẾT QUẢ

### 4.1 Tổ chức chương trình

#### 4.1.1 Tổng quan cấu trúc



Hình 4.1: Tổ chức chương trình

```

Project_SPP/
src/
    main.cpp
    GUI.cpp
    graph.cpp
    algorithms.cpp
    comparison.cpp
    graphics.cpp
    visualizer.py
lib/
    GUI.h
    graph.h
    Algorithms.h
    Comparison.h
    graphics.h
    Global.h
    Colors.h
    graphics_setup.h
data/
    *.txt (graph, temp, report)
  
```

Hình 4.2: Tổ chức tệp tin

#### 4.1.2 Chi tiết chức năng

- `src/main.cpp` xử lý nhập xuất, đọc dữ liệu từ thư mục `data/`, hiển thị menu và gọi tới các thuật toán theo yêu cầu người dùng. Chương trình cho phép lựa chọn nhập đồ thị, thuật toán (Dijkstra hoặc Bellman–Ford), so sánh, trực quan hóa kết quả và in kết quả lên console.
- `graph.cpp` xây dựng danh sách kề dựa trên dữ liệu đọc từ file; có thể chọn cách biểu diễn bằng danh sách hoặc ma trận kề.
- `algorithms.cpp` cài đặt hai thuật toán dựa trên lý thuyết đã trình bày ở Chương 2. Dijkstra sử dụng hàng đợi ưu tiên (min-heap) nên đạt độ phức tạp thời gian  $O((V+E) \log V)$ . Bellman–Ford lặp qua tất cả các cạnh  $V - 1$  lần, do đó có độ phức tạp  $O(V \cdot E)$  và cần thêm một lượt kiểm tra phát hiện chu trình âm.
- `comparison.cpp` đo thời gian thực thi bằng thư viện `<chrono>` và ước lượng bộ nhớ bằng cách theo dõi kích thước cấu trúc dữ liệu (mảng khoảng cách, hàng đợi ưu tiên, danh sách kề). Số liệu bộ nhớ chỉ mang tính tương đối vì còn phụ thuộc vào trình biên dịch và hệ điều hành. Theo phân tích lý thuyết, cả hai thuật toán cần  $O(V + E)$  bộ nhớ để lưu đồ thị và  $O(V)$  cho mảng khoảng cách.
- `GUI.cpp` xây dựng giao diện menu đơn giản trên console, cho phép người dùng nhập đỉnh nguồn, chọn thuật toán và hiển thị kết quả.

#### 4.1.3 Phân loại và mô tả chức năng các hàm trong chương trình

Bảng 4.1: Nhóm xử lý dữ liệu (đồ thị + file I/O)

Hàm	Vị trí	Chức năng
<code>Graph::Graph()</code>	<code>Graph.h</code> , <code>graph.cpp</code>	Khởi tạo đồ thị rỗng, tạo thư mục <code>DATA_FOLDER</code> nếu chưa có.
<code>Graph::Graph(int vertices)</code>	<code>Graph.h</code> , <code>graph.cpp</code>	Khởi tạo đồ thị với số đỉnh ban đầu, đặt nhãn <code>V0..V(n-1)</code> .
<code>Graph:: Graph()</code>	<code>Graph.h</code> , <code>graph.cpp</code>	Giải phóng dữ liệu bằng <code>clear()</code> .
<code>Graph::getVertexCount()</code>	<code>Graph.h</code> , <code>graph.cpp</code>	Trả về số đỉnh <code>V</code> .
<code>Graph::getEdgeCount()</code>	<code>Graph.h</code> , <code>graph.cpp</code>	Trả về số cạnh <code>E</code> .
<code>Graph::getAdjacencyList()</code>	<code>Graph.h</code> , <code>graph.cpp</code>	Trả về danh sách kề (const).
<code>Graph::getVertexLabel(int)</code>	<code>Graph.h</code> , <code>graph.cpp</code>	Trả nhãn đỉnh nếu hợp lệ, ngược lại " <code>Invalid</code> ".

Hàm	Vị trí	Chức năng
Graph::clear()	Graph.h, graph.cpp	Xóa toàn bộ dữ liệu đồ thị, reset V, E.
Graph::addVertex(const std::string&)	Graph.h, graph.cpp	Thêm một đỉnh mới và nhãn tương ứng.
Graph::addEdge(int, int, int)	Graph.h, graph.cpp	Thêm/cập nhật cạnh có trọng số; nếu đã có thì chỉ cập nhật trọng số.
Graph::hasEdge(int, int)	Graph.h, graph.cpp	Kiểm tra tồn tại cạnh source -> destination.
Graph::makeUndirected()	Graph.h, graph.cpp	Bổ sung cạnh ngược để biến đồ thị thành vô hướng.
Graph::fileExists(const std::string&)	Graph.h, graph.cpp	Kiểm tra file có tồn tại không.
Graph::readFromFile(const std::string&, bool&)	Graph.h, graph.cpp	Đọc đồ thị từ file; nếu thiếu file đặt needCreate=true.
Graph::saveToFile(const std::string&)	Graph.h, graph.cpp	Ghi đồ thị ra file (V, nhãn, E, danh sách cạnh).
Graph::exportForPython(const std::string&)	Graph.h, graph.cpp	Xuất đồ thị theo định dạng cho Python.
Graph::exportWithPath(const std::string&, const std::vector<int>&)	Graph.h, graph.cpp	Xuất đồ thị kèm đường đi vào file.
Graph::exportForPython(const std::vector<int>&)	Graph.h, graph.cpp	Xuất đồ thị kèm đường đi vào TEMP_EXPORT_FILE.
Graph::isValid()	Graph.h, graph.cpp	Kiểm tra đồ thị có ít nhất 1 đỉnh.
Graph::hasNegativeWeights()	Graph.h, graph.cpp	Kiểm tra có cạnh trọng số âm không.

Bảng 4.2: Nhóm hàm xử lý thuật toán (Algorithms)

Hàm	Vị trí	Chức năng
Algorithms::Algorithms(const Graph&)	Algorithms.h, algorithms.cpp	Lưu tham chiếu đến đồ thị để xử lý.
Algorithms::dijkstra(int, bool)	Algorithms.h, algorithms.cpp	Chạy Dijkstra, trả PathResult, có log nếu showSteps=true.
Algorithms::bellmanFord(int, bool)	Algorithms.h, algorithms.cpp	Chạy Bellman–Ford, phát hiện chu trình âm.

Hàm	Vị trí	Chức năng
Algorithms::getShortestPath()	Algorithms.h, algorithms.cpp	Trả đường đi ngắn nhất tới đỉnh đích.
Algorithms::getDistance(const PathResult&, int)	Algorithms.h, algorithms.cpp	Trả khoảng cách tới đỉnh đích hoặc -1 nếu lỗi.
Algorithms::logStep()	Algorithms.h, algorithms.cpp	(Private) Ghi log từng bước thuật toán.
Algorithms::reconstructPath(const std::vector<int>&)	Algorithms.h, algorithms.cpp	(Private) Dựng lại đường đi từ mảng previousVertex.
formatDistanceTable(const Graph&, const std::vector<int>&)	algorithms.cpp	(Helper nội bộ) Định dạng bảng khoảng cách để in log.

Bảng 4.3: Bảng 3: Nhóm hàm so sánh/đánh giá

Hàm	Vị trí	Chức năng
Comparison::Comparison(const Graph&)	Comparison.h, comparison.cpp	Khởi tạo bộ so sánh với đồ thị và Algorithms.
Comparison::measureAlgorithm(int, AlgorithmType)	Comparison.h, comparison.cpp	Do thời gian, bộ nhớ, độ phức tạp, trạng thái thành công của thuật toán.
Comparison::comparePerformance(int, AlgorithmType)	Comparison.h, comparison.cpp	Tạo báo cáo so sánh và log tổng hợp.

Bảng 4.4: Nhóm hàm giao diện (GUI)

Hàm	Vị trí	Chức năng
GUI::GUI()	lib/GUI.h, src/GUI.cpp	Khởi tạo kích thước màn hình theo WINDOW_WIDTH/HEIGHT.
GUI:: GUI()	lib/GUI.h, src/GUI.cpp	Hủy đối tượng (không có logic đặc biệt).
GUI::drawMenu()	lib/GUI.h, src/GUI.cpp	Vẽ màn hình menu chính.
GUI::drawComparisonScreen(const std::vector<std::string>& logs)	lib/GUI.h, src/GUI.cpp	Hiển thị log so sánh theo nhiều trang.

Hàm	Vị trí	Chức năng
GUI::clearScreen()	lib/GUI.h, src/GUI.cpp	Xóa màn hình.
GUI::promptMenuChoice()	lib/GUI.h, src/GUI.cpp	Hiển thị menu và đọc lựa chọn 1-6.
GUI::promptChoice(...)	lib/GUI.h, src/GUI.cpp	Hiển thị danh sách lựa chọn và đọc số trong khoảng.
GUI::promptInt(...)	lib/GUI.h, src/GUI.cpp	Đọc số nguyên trong khoảng, có thông báo lỗi.
GUI::promptLine(...)	lib/GUI.h, src/GUI.cpp	Đọc một dòng text, trả defaultValue nếu rỗng.
GUI::promptYesNo(...)	lib/GUI.h, src/GUI.cpp	Hỏi yes/no, trả true/false.
GUI::promptGraphInput(...)	lib/GUI.h, src/GUI.cpp	Nhập số đỉnh, số cạnh và danh sách cạnh từ người dùng.
GUI::promptStartEnd(...)	lib/GUI.h, src/GUI.cpp	Nhập đỉnh bắt đầu và kết thúc, có kiểm tra hợp lệ.
GUI::showAlgorithmLogs(...)	lib/GUI.h, src/GUI.cpp	Hiển thị log thuật toán (nhiều trang).
GUI::showMessage(...)	lib/GUI.h, src/GUI.cpp	Hiển thị hộp thoại thông báo.
GUI::waitForKey()	lib/GUI.h, src/GUI.cpp	Chờ người dùng nhấn phím.

Bảng 4.5: Helper nội bộ GUI

Hàm	Vị trí	Chức năng
approxCharWidth()	GUI.cpp	Ước lượng chiều rộng ký tự theo kích thước cửa sổ.
approxLineHeight()	GUI.cpp	Ước lượng chiều cao dòng.
headerHeight()	GUI.cpp	Tính chiều cao vùng header.
menuHeight()	GUI.cpp	Tính chiều cao vùng menu.
choiceHeight()	GUI.cpp	Tính chiều cao vùng lựa chọn.
utf8CharCount(...)	GUI.cpp	Đếm ký tự UTF-8.
approxTextWidth(...)	GUI.cpp	Ước lượng độ dài text theo ký tự.
wrapLineToWidth(...)	GUI.cpp	Ngắt dòng theo chiều rộng cho 1 dòng.

Hàm	Vị trí	Chức năng
wrapLinesToWidth(...)	GUI.cpp	Ngắt dòng theo chiều rộng cho nhiều dòng.
drawCenteredText(...)	GUI.cpp	Vẽ text căn giữa.
drawLeftAlignedText(...)	GUI.cpp	Vẽ text căn trái.
leftIndentX(...)	GUI.cpp	Tính vị trí lề trái trong khung.
bottomIndentX(...)	GUI.cpp	Tính vị trí lề dưới trong khung.
trim(...)	GUI.cpp	Loại bỏ khoảng trắng đầu/cuối.
tryParseInt(...)	GUI.cpp	Parse chuỗi thành số nguyên, kiểm tra hợp lệ.
tryParseEdgeLine(...)	GUI.cpp	Parse dòng cạnh dạng u v w.
drawInputField(...)	GUI.cpp	Vẽ ô nhập liệu.
clearTextLine(...)	GUI.cpp	Xóa một dòng text trên màn hình.
readLineAt(...)	GUI.cpp	Đọc chuỗi tại tọa độ màn hình, có filter.
drawHorizontalRule(...)	GUI.cpp	Vẽ đường ngang phân cách.
drawFrameBox(...)	GUI.cpp	Vẽ khung có tiêu đề, trả tọa độ bên trong.
drawHeaderFrame(...)	GUI.cpp	Vẽ khung header chương trình.
drawContentFrame(...)	GUI.cpp	Vẽ khung nội dung chính.
drawLogFrame(...)	GUI.cpp	Vẽ khung hiển thị log.

Bảng 4.6: Graphics API

Hàm	Vị trí	Chức năng
initgraph(...)	graphics.h, graphics.cpp	Khởi tạo đồ họa giả lập ANSI, xóa màn hình.
graphresult()	graphics.h, graphics.cpp	Trả trạng thái khởi tạo đồ họa.
closegraph()	graphics.h, graphics.cpp	Reset màu/dóng đồ họa.
cleardevice()	graphics.h, graphics.cpp	Xóa màn hình.
setcolor(int)	graphics.h, graphics.cpp	Đặt màu vẽ hiện tại.
setbkcolor(int)	graphics.h, graphics.cpp	Đặt màu nền (không dùng).

Hàm	Vị trí	Chức năng
outtextxy(int, int, char*)	graphics.h, graphics.cpp	Vẽ text tại (x,y).
rectangle(...)	graphics.h, graphics.cpp	Vẽ khung chữ nhật.
getch()	graphics.h, graphics.cpp	Đọc một ký tự từ stdin.

Bảng 4.7: Helper nội bộ Graphics

Hàm	Vị trí	Chức năng
setConsoleUtf8()	graphics.cpp	Bật UTF-8 cho console Windows.
enableAnsi()	graphics.cpp	Bật ANSI escape cho console.
mapX(int)	graphics.cpp	Ánh xạ tọa độ X ảo → console.
mapY(int)	graphics.cpp	Ánh xạ tọa độ Y ảo → console.
ansiColorCode(int)	graphics.cpp	Ánh xạ mã màu BGI → ANSI.
moveCursor(int, int)	graphics.cpp	Di chuyển con trỏ console.
applyColor()	graphics.cpp	Áp dụng màu hiện tại.
writeUtf8(const char*)	graphics.cpp	Ghi UTF-8 an toàn lên console.

Bảng 4.8: Main + điều phối (src/main.cpp)

Hàm	Vị trí	Chức năng
initConsoleUtf8()	main.cpp	Bật UTF-8 và font console (Windows).
promptGraphDirected()	main.cpp	Hỏi đồ thị có hướng hay vô hướng.
createGraphFromGui(bool)	main.cpp	Tạo đồ thị mới từ dữ liệu nhập GUI.
handleGraphInput()	main.cpp	Xử lý luồng tải/tạo đồ thi.
runAlgorithm(AlgorithmType)	main.cpp	Chạy Dijkstra/Bellman–Ford và hiển thị kết quả.
compareAlgorithms()	main.cpp	So sánh hiệu năng 2 thuật toán.
exportToPython()	main.cpp	Xuất đồ thị/đường đi ra file cho Python.
main()	main.cpp	Khởi tạo hệ thống, vòng lặp menu, dọn dẹp tài nguyên.

## 4.2 Ngôn ngữ cài đặt

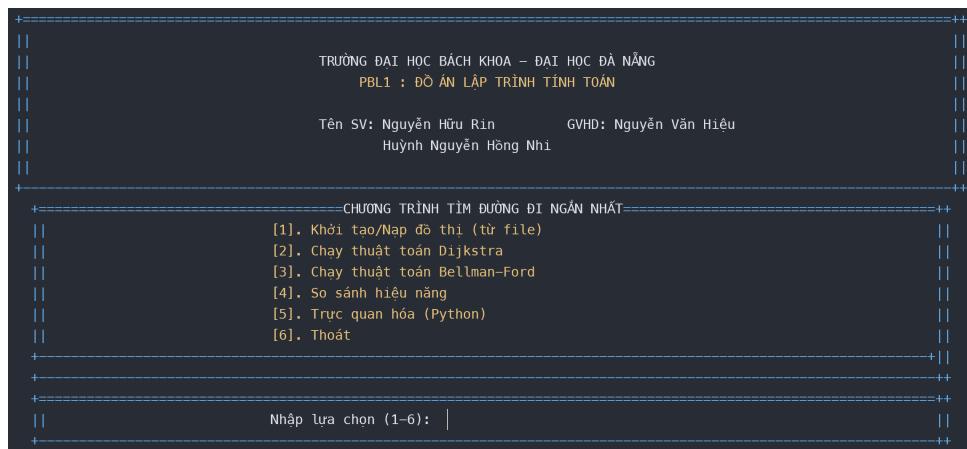
Bài toán được triển khai chủ yếu bằng ngôn ngữ C++ để đảm bảo hiệu năng cao khi thao tác trên đồ thị lớn. Ngoài thư viện chuẩn, chương trình còn sử dụng một số thư viện hỗ trợ:

- `windows.h` và `cwchar` (chỉ trên Windows) – thay đổi chế độ console sang UTF-8 và thiết lập phông chữ khi biên dịch trên Windows; khai báo có điều kiện qua biến `_WIN32`.
- `graphics.h` – lớp giả lập đồ họa trên console ANSI cho phép vẽ sơ đồ và hiển thị trực quan kết quả.
- `Graph.h`, `Algorithms.h`, `Comparison.h`, `GUI.h` – các header được định nghĩa trong thư mục `lib/` dùng để tổ chức mã nguồn thành nhiều module: cấu trúc đồ thị, cài đặt thuật toán Dijkstra và Bellman–Ford, chức năng đo thời gian chạy và bộ nhớ, và giao diện người dùng dạng menu.
- **Python** – được dùng trong script `visualizer.py` (thư mục `src/`) để trực quan hóa đồ thị và kết quả bằng `networkx` và `matplotlib`. Khi chạy trên Linux (WSL) có thể sử dụng Jupyter hoặc chạy trực tiếp bằng lệnh `python visualizer.py`.

## 4.3 Kết quả

### 4.3.1 Giao diện chính

#### Giao diện chính của chương trình



Hình 4.3: Hình ảnh giao diện chính của chương trình

## Giao diện nhập đồ thị



Hình 4.4: Hình ảnh giao diện nhập đồ thị

#### 4.3.2 Kết quả thực thi

## Đồ thị có trọng số dương

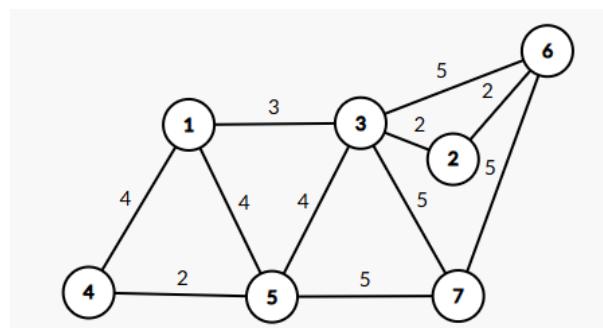
**Ví dụ:** Cho đồ thi vô hướng, không âm với:

- Số đỉnh (V) : 7
  - Số cạnh (E) : 11
  - Danh sách cạnh (u, v, w) :

1 3 3	2 3 2	3 6 5	4 5 2	6 7 5
1 4 4	2 6 2	3 5 4	5 7 5	7 3 5
1 5 4				

Bảng 4.9: Danh sách canh

### **Biểu diễn đồ thi**



Hình 4.5: Biểu diễn đồ thị

### Chức năng Thuật toán Dijkstra

```

+=====
||          TRƯỜNG ĐẠI HỌC BÁCH KHOA – ĐẠI HỌC ĐÀ NẴNG
||          PBL1 : ĐỒ ÁN LẬP TRÌNH TÍNH TOÁN
||
||          Tên SV: Nguyễn Hữu Rin           GVHD: Nguyễn Văn Hiệu
||          Huỳnh Nguyễn Hồng Nhi
||
+=====KẾT QUẢ THUẬT TOÁN DIJKSTRA=====
||          Đường đi ngắn nhất từ 1 đến 6:
||
||          1 -> 3 -> 2 -> 6
||          Khoảng cách = 7
||          Thời gian thực hiện: 178 us
||
||          Nhấn phím bất kỳ để tiếp tục...
|

```

Hình 4.6: Kết quả thuật toán Dijkstra

### Chức năng Thuật toán Bellman-Ford

```

+=====
||          TRƯỜNG ĐẠI HỌC BÁCH KHOA – ĐẠI HỌC ĐÀ NẴNG
||          PBL1 : ĐỒ ÁN LẬP TRÌNH TÍNH TOÁN
||
||          Tên SV: Nguyễn Hữu Rin           GVHD: Nguyễn Văn Hiệu
||          Huỳnh Nguyễn Hồng Nhi
||
+=====KẾT QUẢ THUẬT TOÁN BELLMAN-FORD=====
||          Đường đi ngắn nhất từ 1 đến 6:
||
||          1 -> 3 -> 2 -> 6
||          Khoảng cách = 7
||          Thời gian thực hiện: 142 us
||
||          Nhấn phím bất kỳ để tiếp tục...
|

```

Hình 4.7: Kết quả thuật toán Bellman-Ford

### Chức năng So sánh

```

+-----+
||          TRƯỜNG ĐẠI HỌC BÁCH KHOA – ĐẠI HỌC ĐÀ NẴNG
||          PBL1 : ĐỒ ÁN LẬP TRÌNH TÍNH TOÁN
||
||          Tên SV: Nguyễn Hữu Rin      GVHD: Nguyễn Văn Hiệu
||          Huỳnh Nguyễn Hồng Nhi
||
+-----+-----+-----+
||          SO SÁNH THUẬT TOÁN
||          =====
||          BÁO CÁO SO SÁNH HIỆU NĂNG
||          =====
||          Định bắt đầu: 1  Số đỉnh (V): 7  Số cạnh (E): 22
||          +-----+-----+
||          |       |DIJKSTRA      |BELLMAN-FORD
||          +-----+-----+
||          |Thời gian chạy |22 us           |9 us
||          |Bộ nhớ dùng   |232 bytes        |232 bytes
||          +-----+-----+
||          |Đồ phức tạp  | $O(E \log V) \approx O(42)$  | $O(V \times E) \approx O(154)$ 
||          |Trạng thái    |Thành công         |Thành công
||          +-----+-----+
||          --- SO SÁNH ---
||          Bellman-Ford chậm hơn Dijkstra 0.409091 lần
||          +-----+
||          Nhấn phím bất kỳ để quay lại menu...
+-----+

```

Hình 4.8: Kết quả chức năng so sánh

#### 4.3.3 Nhận xét

## Chương 5. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

### 5.1 Kết luận

### 5.2 Hướng phát triển

## TÀI LIỆU THAM KHẢO

## PHỤ LỤC

### .1 Toàn bộ chương trình và cài đặt

#### Link GitHub

Toàn bộ mã nguồn và thông tin tại: <https://github.com/Nhr1410x/PBL1>

### .2 Mã nguồn chương trình

#### .2.1 Mã nguồn chương trình chính : main.cpp

```

1 #include <iostream>
2 #include <limits>
3 #include <string>
4 #include <vector>
5 #include <chrono>
6 #ifdef _WIN32
7 #include <windows.h>
8 #include <cwchar>
9 #endif
10 #include "../lib/graphics.h"
11 #include "../lib/Global.h"
12 #include "../lib/GUI.h"
13 #include "../lib/graph.h"
14 #include "../lib/Algorithms.h"
15 #include "../lib/Comparison.h"
16
17 // ===== GLOBAL VARIABLES =====
18 Graph graph;
19 Algorithms* algorithms = nullptr;
20 Comparison* comparison = nullptr;
21 GUI* gui = nullptr;
22
23 PathResult lastResult;
24
25 #ifdef _WIN32
26 static void initConsoleUtf8() {
27     SetConsoleOutputCP(CP_UTF8);
28     SetConsoleCP(CP_UTF8);
29     HANDLE hOut = GetStdHandle(STD_OUTPUT_HANDLE);
30     if (hOut != INVALID_HANDLE_VALUE) {
31         CONSOLE_FONT_INFOEX cfi;
32         cfi.cbSize = sizeof(cfi);
33         if (GetCurrentConsoleFontEx(hOut, FALSE, &cfi)) {
34             cfi.FontFamily = FF_DONTCARE;
35             cfi.FontWeight = FW_NORMAL;
36             std::wcscpy(cfi.FaceName, L"Consolas", LF_FACESIZE - 1);
37             cfi.FaceName[LF_FACESIZE - 1] = L'\0';
38             SetCurrentConsoleFontEx(hOut, FALSE, &cfi);
39         }
40     }
41 }
42 #endif

```

```

44 static bool promptGraphDirected() {
45     std::vector<std::string> options = {
46         "[1]. Co huong",
47         "[2]. Vo huong"
48     };
49     int choice = gui->promptChoice("LOAI DO THI", options, "Nhập lựa chọn:", 1, 2);
50     return choice == 1;
51 }
52
53 static void createGraphFromGui(bool isDirected) {
54     graph.clear();
55
56     int numVertices = 0;
57     int numEdges = 0;
58     std::vector<std::tuple<int, int, int>> edges;
59     gui->promptGraphInput(isDirected, numVertices, numEdges, edges);
60     for (int i = 0; i < numVertices; i++) {
61         graph.addVertex(std::to_string(i + 1));
62     }
63
64     for (const auto& edge : edges) {
65         int source = std::get<0>(edge);
66         int dest = std::get<1>(edge);
67         int weight = std::get<2>(edge);
68         graph.addEdge(source, dest, weight);
69     }
70
71     if (!isDirected) {
72         graph.makeUndirected();
73     }
74 }
75
76 static void handleGraphInput() {
77     std::vector<std::string> options = {
78         "[1]. Tai tu file",
79         "[2]. Tao do thi moi (nhap truc tiep)",
80         "[3]. Quay lai"
81     };
82     int choice = gui->promptChoice("TAO/TAI DO THI", options, "Nhập lựa chọn:", 1, 3);
83
84     if (choice == 1) {
85         bool isDirected = promptGraphDirected();
86         std::string filename = gui->promptLine("TAI DO THI",
87             "Nhập đường dẫn file (mặc định: " + DEFAULT_GRAPH_FILE + "): ",
88             DEFAULT_GRAPH_FILE
89         );
90         bool needCreate = false;
91         if (!graph.readFromFile(filename, needCreate)) {
92             if (needCreate) {
93                 bool create = gui->promptYesNo("TAI DO THI", "Không tìm thấy file. Tao
file moi?");
94                 if (create) {
95                     createGraphFromGui(isDirected);
96                     lastResult = PathResult();
97                     if (graph.saveToFile(filename)) {
98                         gui->showMessage("TAI DO THI", {"Đã lưu đồ thị vào: " + filename
99                     });
100                } else {
101                    gui->showMessage("TAI DO THI", {"Lưu đồ thị thất bại."});
102            }
103        }
104    }
105}

```

```

101         }
102     }
103 } else {
104     gui->showMessage("TAI DO THI", {"Tai file that bai."});
105 }
106 } else {
107     if (!isDirected) {
108         graph.makeUndirected();
109     }
110     lastResult = PathResult();
111     gui->showMessage("TAI DO THI", {
112         "Da tai do thi: V = " + std::to_string(graph.getVertexCount()) +
113         ", E = " + std::to_string(graph.getEdgeCount()) + "."
114     });
115 }
116 } else if (choice == 2) {
117     bool isDirected = promptGraphDirected();
118     createGraphFromGui(isDirected);
119
120     lastResult = PathResult();
121     bool save = gui->promptYesNo("TAO DO THI", "Luu do thi ra file?");
122     if (save) {
123         std::string filename = gui->promptLine(
124             "TAO DO THI",
125             "Duong dan file (mac dinh: " + DEFAULT_GRAPH_FILE + "): ",
126             DEFAULT_GRAPH_FILE
127         );
128         if (graph.saveToFile(filename)) {
129             gui->showMessage("TAO DO THI", {"Da luu do thi vao " + filename});
130         } else {
131             gui->showMessage("TAO DO THI", {"Luu do thi that bai."});
132         }
133     }
134 }
135 }
136
137 static void runAlgorithm(AlgorithmType type) {
138     if (!graph.isValid()) {
139         gui->showMessage("THONG BAO", {"Do thi rong. Vui long tao hoac tai do thi truoc."});
140         return;
141     }
142
143     if (type == AlgorithmType::DIJKSTRA && graph.hasNegativeWeights()) {
144         gui->showMessage("THONG BAO", {"Dijkstra khong chay duoc voi canh co trong so am."});
145         return;
146     }
147
148     int V = graph.getVertexCount();
149     int startInput = 1;
150     int endInput = 1;
151     gui->promptStartEnd("CHON DINH", 1, V, startInput, endInput);
152     int start = startInput - 1;
153     int end = endInput - 1;
154
155     PathResult result;
156     auto startTime = std::chrono::high_resolution_clock::now();
157     if (type == AlgorithmType::DIJKSTRA) {

```

```

158     result = algorithms->dijkstra(start, true);
159 } else {
160     result = algorithms->bellmanFord(start, true);
161 }
162 auto endTime = std::chrono::high_resolution_clock::now();
163 auto execUs = std::chrono::duration_cast<std::chrono::microseconds>(endTime -
164 startTime).count();
165
166 if (!result.success) {
167     std::vector<std::string> lines = {"Thuat toan that bai."};
168     if (result.hasNegativeCycle) {
169         lines.push_back("Phat hien chu trinh am.");
170     }
171     gui->showMessage("KET QUA", lines);
172     return;
173 }
174
175 result.shortestPath = algorithms->getShortestPath(result, end);
176 lastResult = result;
177 std::string algoTitle = (type == AlgorithmType::DIJKSTRA)
178 ? "DIEN BIEN THUAT TOAN DIJKSTRA"
179 : "DIEN BIEN THUAT TOAN BELLMAN-FORD";
180
181 std::vector<std::string> displayLogs;
182 std::string startLabel = graph.getVertexLabel(start);
183 std::string endLabel = graph.getVertexLabel(end);
184 int dist = algorithms->getDistance(result, end);
185 if (dist >= 0 && dist != INT_MAX) {
186     displayLogs.push_back("Duong di ngan nhat tu dinh " + startLabel + " den dinh " +
187     endLabel + " la " + std::to_string(dist));
188 }
189 // else {
190 //     displayLogs.push_back("Khong ton tai duong di tu " + startLabel + " den " +
191 endLabel);
192 //}
193
194 if (result.shortestPath.empty()) {
195     displayLogs.push_back("Khong ton tai duong di tu " + startLabel + " den " +
196 endLabel);
197 } else {
198     std::string pathLine = "Duong di: ";
199     for (size_t i = 0; i < result.shortestPath.size(); i++) {
200         if (i > 0) pathLine += " -> ";
201         pathLine += graph.getVertexLabel(result.shortestPath[i]);
202     }
203     displayLogs.push_back(pathLine);
204 }
205
206 displayLogs.push_back("Thoi gian thuc hien: " + std::to_string(execUs) + " us");
207 displayLogs.push_back("");
208 displayLogs.insert(displayLogs.end(), result.logs.begin(), result.logs.end());
209
210 gui->showAlgorithmLogs(algoTitle, displayLogs);
211
212 std::vector<std::string> lines;
213 lines.push_back("Duong di ngan nhat tu " + graph.getVertexLabel(start) +
214 " den " + graph.getVertexLabel(end) + ":");
215 if (result.shortestPath.empty()) {
216     lines.push_back("Khong tim thay duong di.");
217 }

```

```

213     else {
214         lines.push_back("");
215         std::string pathLine;
216         for (size_t i = 0; i < result.shortestPath.size(); i++) {
217             if (!pathLine.empty()) pathLine += " -> ";
218             pathLine += graph.getVertexLabel(result.shortestPath[i]);
219         }
220         lines.push_back(pathLine);
221         int dist = algorithms->getDistance(result, end);
222         lines.push_back("Khoang cach = " + std::to_string(dist));
223         lines.push_back("Thoi gian thuc hien: " + std::to_string(execUs) + " us");
224     }
225     if(type == AlgorithmType::DIJKSTRA){
226         gui->showMessage("KET QUA THUAT TOAN DIJKSTRA", lines);
227     }
228     else {
229         gui->showMessage("KET QUA THUAT TOAN BELLMAN-FORD", lines);
230     }
231 //    gui->showMessage("KET QUA", lines);
232 }
233
234 static void compareAlgorithms() {
235     if (!graph.isValid()) {
236         gui->showMessage("THONG BAO", {"Do thi rong. Vui long tao hoac tai do thi truoc."});
237     }
238     return;
239 }
240
241 int V = graph.getVertexCount();
242 int startInput = gui->promptInt("CHON DINH",
243     "Dinh bat dau (1.." + std::to_string(V) + "): ", 1, V);
244 int start = startInput - 1;
245
246 auto report = comparison->comparePerformance(start, AlgorithmType::BOTH);
247 gui->drawComparisonScreen(report.logs);
248 gui->waitForKey();
249 }
250
251 static void exportToPython() {
252     if (!graph.isValid()) {
253         gui->showMessage("THONG BAO", {"Do thi rong. Vui long tao hoac tai do thi truoc."});
254     }
255     return;
256 }
257
258 bool ok = false;
259 if (!lastResult.shortestPath.empty()) {
260     ok = graph.exportForPython(lastResult.shortestPath);
261 } else {
262     ok = graph.exportForPython(TEMP_EXPORT_FILE);
263 }
264
265 if (ok) {
266     gui->showMessage("XUAT PYTHON", {
267         "Da xuat ra " + TEMP_EXPORT_FILE,
268         "Chay: python visualizer.py"
269     });
270 } else {
271     gui->showMessage("XUAT PYTHON", {"Xuat that bai."});

```

```

270     }
271 }
272 // ====== MAIN FUNCTION ======
273 int main() {
274 #ifdef _WIN32
275     initConsoleUtf8();
276 #endif
277     int gd = DETECT, gm;
278     char bgiPath[] = "";
279     initgraph(&gd, &gm, bgiPath);
280
281     if (graphresult() != grOk) {
282         std::cerr << "Khoi tao do hoa that bai. Chuyen sang che do console.\n";
283     }
284
285     algorithms = new Algorithms(graph);
286     comparison = new Comparison(graph);
287     gui = new GUI();
288
289     bool running = true;
290     while (running) {
291         int choice = gui->promptMenuChoice();
292
293         switch (choice) {
294             case 1:
295                 handleGraphInput();
296                 break;
297             case 2:
298                 runAlgorithm(AlgorithmType::DIJKSTRA);
299                 break;
300             case 3:
301                 runAlgorithm(AlgorithmType::BELLMAN_FORD);
302                 break;
303             case 4:
304                 compareAlgorithms();
305                 break;
306             case 5:
307                 exportToPython();
308                 break;
309             case 6:
310                 running = false;
311                 break;
312             default:
313                 break;
314         }
315     }
316 }
317
318 closegraph();
319 delete algorithms;
320 delete comparison;
321 delete gui;
322
323 return 0;
324 }
```

## 2.2 Mã nguồn chương trình ghi dữ liệu : graph.cpp

```

1 #include "../lib/Graph.h"
2 #include <filesystem>
3 #include <fstream>
4
5 // ====== CONSTRUCTOR & DESTRUCTOR ======
6 Graph::Graph() : V(0), E(0) {
7     std::filesystem::create_directories(DATA_FOLDER);
8 }
9
10 Graph::Graph(int vertices) : V(0), E(0) {
11     std::filesystem::create_directories(DATA_FOLDER);
12     for (int i = 0; i < vertices; i++) {
13         addVertex("V" + std::to_string(i));
14     }
15 }
16
17 Graph::~Graph() {
18     clear();
19 }
20
21 // ====== GETTERS ======
22 int Graph::getVertexCount() const {
23     return V;
24 }
25
26 int Graph::getEdgeCount() const {
27     return E;
28 }
29
30 const std::vector<std::vector<Edge>>& Graph::getAdjacencyList() const {
31     return adjList;
32 }
33
34 std::string Graph::getVertexLabel(int vertex) const {
35     if (vertex >= 0 && vertex < V) {
36         return vertexLabels[vertex];
37     }
38     return "Invalid";
39 }
40
41 // ====== GRAPH MODIFICATION ======
42 void Graph::clear() {
43     adjList.clear();
44     vertexLabels.clear();
45     V = 0;
46     E = 0;
47 }
48
49 void Graph::addVertex(const std::string& label) {
50     adjList.push_back(std::vector<Edge>());
51     vertexLabels.push_back(label);
52     V++;
53 }
54
55 void Graph::addEdge(int source, int destination, int weight) {
56     if (source < 0 || source >= V || destination < 0 || destination >= V) {
57         return;
58     }

```

```

59
60     for (auto& edge : adjList[source]) {
61         if (edge.destination == destination) {
62             edge.weight = weight;
63             return;
64         }
65     }
66
67     adjList[source].push_back(Edge(destination, weight));
68     E++;
69 }
70
71 bool Graph::hasEdge(int source, int destination) const {
72     if (source < 0 || source >= V) {
73         return false;
74     }
75     for (const auto& edge : adjList[source]) {
76         if (edge.destination == destination) {
77             return true;
78         }
79     }
80     return false;
81 }
82
83 void Graph::makeUndirected() {
84     for (int i = 0; i < V; i++) {
85         for (const auto& edge : adjList[i]) {
86             if (!hasEdge(edge.destination, i)) {
87                 addEdge(edge.destination, i, edge.weight);
88             }
89         }
90     }
91 }
92
93 // ===== FILE I/O =====
94 bool Graph::fileExists(const std::string& filename) const {
95     return std::filesystem::exists(filename);
96 }
97
98 bool Graph::readFromFile(const std::string& filename, bool& needCreate) {
99     needCreate = false;
100
101     if (!fileExists(filename)) {
102         needCreate = true;
103         return false;
104     }
105
106     std::ifstream file(filename);
107     if (!file.is_open()) {
108         return false;
109     }
110
111     clear();
112
113     int numVertices = 0;
114     if (!(file >> numVertices)) {
115         return false;
116     }
117

```

```

118     for (int i = 0; i < numVertices; i++) {
119         std::string label;
120         if (!(file >> label)) {
121             label = std::to_string(i + 1);
122         }
123         addVertex(label);
124     }
125
126     int numEdges = 0;
127     if (!(file >> numEdges)) {
128         return false;
129     }
130
131     for (int i = 0; i < numEdges; i++) {
132         int source = 0, dest = 0, weight = 0;
133         if (file >> source >> dest >> weight) {
134             addEdge(source - 1, dest - 1, weight);
135         }
136     }
137
138     return true;
139 }
140
141 bool Graph::saveToFile(const std::string& filename) const {
142     std::ofstream file(filename);
143     if (!file.is_open()) {
144         return false;
145     }
146
147     file << V << "\n";
148     for (const auto& label : vertexLabels) {
149         file << label << " ";
150     }
151     file << "\n";
152
153     file << E << "\n";
154     for (int i = 0; i < V; i++) {
155         for (const auto& edge : adjList[i]) {
156             file << (i + 1) << " " << (edge.destination + 1) << " " << edge.weight << "\n"
157         };
158     }
159
160     return true;
161 }
162
163 bool Graph::exportForPython(const std::string& filename) const {
164     std::ofstream file(filename);
165     if (!file.is_open()) {
166         return false;
167     }
168
169     file << V << "\n";
170     for (const auto& label : vertexLabels) {
171         file << label << " ";
172     }
173     file << "\n";
174
175     for (int i = 0; i < V; i++) {

```

```

176     for (const auto& edge : adjList[i]) {
177         file << (i + 1) << " " << (edge.destination + 1) << " " << edge.weight << "\n"
178         ";
179     }
180 }
181 return true;
182 }

183 bool Graph::exportWithPath(const std::string& filename, const std::vector<int>& path)
184 const {
185     std::ofstream file(filename);
186     if (!file.is_open())
187         return false;
188     }

189     file << V << "\n";
190     for (const auto& label : vertexLabels) {
191         file << label << " ";
192     }
193     file << "\n";

194     for (int i = 0; i < V; i++) {
195         for (const auto& edge : adjList[i]) {
196             file << (i + 1) << " " << (edge.destination + 1) << " " << edge.weight << "\n"
197             ";
198         }
199     }
200 }

201     file << "\nPATH:\n";
202     for (int v : path) {
203         file << (v + 1) << " ";
204     }
205     file << "\n";
206
207     return true;
208 }
209 }

210 bool Graph::exportForPython(const std::vector<int>& path) const {
211     return exportWithPath(TEMP_EXPORT_FILE, path);
212 }
213 }

214 // ===== VALIDATION =====
215 bool Graph::isValid() const {
216     return V > 0;
217 }
218 }

219 bool Graph::hasNegativeWeights() const {
220     for (int i = 0; i < V; i++) {
221         for (const auto& edge : adjList[i]) {
222             if (edge.weight < 0) {
223                 return true;
224             }
225         }
226     }
227     return false;
228 }
229 }
```

### 2.3 Mã nguồn chương trình thực hiện thuật toán : algorithms.cpp

```

1 #include "../lib/Algorithms.h"
2 #include <queue>
3 #include <limits>
4 #include <sstream>
5 #include <iomanip>
6
7 namespace {
8 std::vector<std::string> formatDistanceTable(const Graph& graph, const std::vector<int>&
9     distances) {
10    std::vector<std::string> lines;
11    if (distances.empty()) return lines;
12
13    const int INF = std::numeric_limits<int>::max();
14    std::ostringstream header;
15    std::ostringstream values;
16
17    header << "Dinh :";
18    values << "D(i) :";
19
20    for (size_t i = 0; i < distances.size(); ++i) {
21        const std::string label = graph.getVertexLabel(static_cast<int>(i));
22        header << std::setw(6) << label;
23        if (distances[i] == INF) {
24            values << std::setw(6) << "INF";
25        } else {
26            values << std::setw(6) << distances[i];
27        }
28    }
29
30    lines.push_back(header.str());
31    lines.push_back(values.str());
32    return lines;
33 }
34 // namespace
35 Algorithms::Algorithms(const Graph& g) : graph(g) {}
36
37 void Algorithms::logStep(std::vector<std::string>& logs, const std::string& message) {
38    logs.push_back(message);
39 }
40
41 std::vector<int> Algorithms::reconstructPath(int destination, const std::vector<int>&
42     previousVertex) const {
43    std::vector<int> path;
44    int current = destination;
45    while (current != -1) {
46        path.insert(path.begin(), current);
47        current = previousVertex[current];
48    }
49    return path;
50 }
51
52 // Dijkstra
53 PathResult Algorithms::dijkstra(int start, bool showSteps) {
54    PathResult result;
55    result.startVertex = start;

```

```

56     int V = graph.getVertexCount();
57     const auto& adjList = graph.getAdjacencyList();
58
59     const int INF = std::numeric_limits<int>::max();
60     result.distances.assign(V, INF);
61     result.previousVertex.assign(V, -1);
62     result.distances[start] = 0;
63
64     std::priority_queue<std::pair<int, int>, std::vector<std::pair<int, int>>, std::
65     greater<std::pair<int, int>>> pq;
66     pq.push({0, start});
67
68     if (showSteps) {
69         logStep(result.logs, "===== THUAT TOAN DIJKSTRA =====");
70         logStep(result.logs, "Dinh bat dau: " + graph.getVertexLabel(start));
71         logStep(result.logs, "Khoi tao khoang cach: tat ca = INF, rieng dinh bat dau = 0");
72     };
73
74     std::vector<bool> visited(V, false);
75     int iterations = 0;
76
77     while (!pq.empty()) {
78         auto [dist, u] = pq.top();
79         pq.pop();
80
81         if (visited[u]) continue;
82         visited[u] = true;
83         iterations++;
84
85         if (showSteps) {
86             logStep(result.logs, "");
87             logStep(result.logs, "[Lan lap " + std::to_string(iterations) + "]");
88             logStep(result.logs, "Xu ly dinh: " + graph.getVertexLabel(u) +
89                     " (khoang cach = " + std::to_string(dist) + ")");
90         }
91
92         for (const auto& edge : adjList[u]) {
93             int v = edge.destination;
94             int weight = edge.weight;
95
96             if (result.distances[u] != INF &&
97                 result.distances[u] + weight < result.distances[v]) {
98
99                 result.distances[v] = result.distances[u] + weight;
100                result.previousVertex[v] = u;
101                pq.push({result.distances[v], v});
102
103                if (showSteps) {
104                    logStep(result.logs, "Cap nhat: " + graph.getVertexLabel(u) + " ->
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
787
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
817
818
819
819
820
821
822
823
824
825
825
826
827
827
828
829
829
830
831
832
832
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1560
1561
1561
1562
1562
1563
1563
1564
1564
1565
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1570
1571
1571
1572
1572
1573
1573
1574
1574
1575
1575
1576
1576
1577
1577
1578
1578
1579
1579
1580
1580
1581
1581
1582
1582
1583
1583
1584
1584
1585
1585
1586
1586
1587
1587
1588
1588
1589
1589
1590
1590
1591
1591
1592
1592
1593
1593
1594
1594
1595
1595
1596
1596
1597
1597
1598
1598
1599
1599
1600
1600
1601
1601
1602
1602
1603
1603
1604
1604
1605
1605
1606
1606
1607
1607
1608
1608
1609
1609
1610
1610
1611
1611
1612
1612
1613
1613
1614
1614
1615
1615
1616
1616
1617
1617
1618
1618
1619
1619
1620
1620
1621
1621
1622
1622
1623
1623
1624
1624
1625
1625
1626
1626
1627
1627
1628
1628
1629
1629
1630
1630
1631
1631
1632
1632
1633
1633
1634
1634
1635
1635
1636
1636
1637
1637
1638
1638
1639
1639
1640
1640
1641
1641
1642
1642
1643
1643
1644
1644
1645
1645
1646
1646
1647
1647
1648
1648
1649
1649
1650
1650
1651
1651
1652
1652
1653
1653
1654
1654
1655
1655
1656
1656
1657
1657
1658
1658
1659
1659
1660
1660
1661
1661
1662
1662
1663
1663
1664
1664
1665
1665
1666
1666
1667
1667
1668
1668
1669
1669
1670
1670
1671
1671
1672
1672
1673
1673
1674
1674
1675
1675
1676
1676
1677
1677
1678
1678
1679
1679
1680
1680
1681
1681
1682
1682
1683
1683
1684
1684
1685
1685
1686
1686
1687
1687
1688
1688
1689
1689
1690
1690
1691
1691
1692
1692
1693
1693
1694
1694
1695
1695
1696
1696
1697
1697
1698
1698
1699
1699
1700
1700
1701
1701
1702
1702
1703
1703
1704
1704
1705
1705
1706
1706
1707
1707
1708
1708
1709
1709
1710
1710
1711
1711
1712
1712
1713
1713
1714
1714
1715
1715
1716
1716
1717
1717
1718
1718
1719
1719
1720
1720
1721
1721
1722
1722
1723
1723
1724
1724
1725
1725
1726
1726
1727
1727
1728
1728
1729
1729
1730
1730
1731
1731
1732
1732
1733
1733
1734
1734
1735
1735
1736
1736
1737
1737
1738
1738
1739
1739
1740
1740
1741
1741
1742
1742
1743
1743
1744
1744
1745
1745
1746

```

```

111     logStep(result.logs, "Khoang cach sau lan lap " + std::to_string(iterations)
112     + ":" );
113     auto table = formatDistanceTable(graph, result.distances);
114     for (const auto& line : table) {
115         logStep(result.logs, line);
116     }
117 }
118
119 if (showSteps) {
120     logStep(result.logs, "");
121     logStep(result.logs, "                                === KHOANG CACH CUOI ===");
122     for (int i = 0; i < V; i++) {
123         if (result.distances[i] == INF) {
124             logStep(result.logs, "dist[" + std::to_string(i+1) + "] = INF (khong ton t
ai duong di)");
125         }
126         else {
127             logStep(result.logs, "dist[" + std::to_string(i+1) + "] = " + std::
to_string(result.distances[i]));
128         }
129     }
130 }
131
132 result.success = true;
133 return result;
134 }
135
136 //bellman
137 PathResult Algorithms::bellmanFord(int start, bool showSteps) {
138     PathResult result;
139     result.startVertex = start;
140     int V = graph.getVertexCount();
141     const auto& adjList = graph.getAdjacencyList();
142
143     const int INF = std::numeric_limits<int>::max();
144     result.distances.assign(V, INF);
145     result.previousVertex.assign(V, -1);
146     result.distances[start] = 0;
147
148     if (showSteps) {
149         logStep(result.logs, "          ====== THUAT TOAN BELLMAN-FORD ======");
150         logStep(result.logs, "Dinh bat dau: " + graph.getLabel(start));
151         logStep(result.logs, "Khoi tao khoang cach: tat ca = INF, rieng dinh bat dau = 0"
);
152     }
153
154     for (int i = 0; i < V - 1; i++) {
155         bool updated = false;
156
157         if (showSteps) {
158             logStep(result.logs, "");
159             logStep(result.logs, "[Luot " + std::to_string(i + 1) + "/" + std::to_string(
V - 1) + "]");
160         }
161
162         for (int u = 0; u < V; u++) {
163             for (const auto& edge : adjList[u]) {
164                 int v = edge.destination;

```

```

165         int weight = edge.weight;
166
167         if (result.distances[u] != INF &&
168             result.distances[u] + weight < result.distances[v]) {
169
170             result.distances[v] = result.distances[u] + weight;
171             result.previousVertex[v] = u;
172             updated = true;
173
174             if (showSteps) {
175                 logStep(result.logs, " Cap nhat: " + graph.getVertexLabel(u) +
176                         "-> " +
177                         graph.getVertexLabel(v) +
178                         " (khoang cach moi = " + std::to_string(
179                         result.distances[v]) + ")");
180             }
181         }
182     }
183
184     if (!updated && showSteps) {
185         logStep(result.logs, " (Khong co cap nhat o luot nay - co the dung som)");
186     }
187
188     if (showSteps) {
189         logStep(result.logs, "Bang khoang cach sau luot " + std::to_string(i + 1) + ":");
190         auto table = formatDistanceTable(graph, result.distances);
191         for (const auto& line : table) {
192             logStep(result.logs, line);
193         }
194     }
195
196     result.hasNegativeCycle = false;
197     if (showSteps) {
198         logStep(result.logs, "");
199         logStep(result.logs, "==== KIEM TRA CHU TRINH AM ===");
200     }
201
202     for (int u = 0; u < V; u++) {
203         for (const auto& edge : adjList[u]) {
204             int v = edge.destination;
205             int weight = edge.weight;
206
207             if (result.distances[u] != INF &&
208                 result.distances[u] + weight < result.distances[v]) {
209                 result.hasNegativeCycle = true;
210
211                 if (showSteps) {
212                     logStep(result.logs, "PHAT HIEN CHU TRINH AM!");
213                     logStep(result.logs, "Canh: " + graph.getVertexLabel(u) + " -> " +
214                         graph.getVertexLabel(v) +
215                         " (trong so = " + std::to_string(weight) + ")");
216                 }
217             }
218         }
219     }
220

```

```

221     if (!result.hasNegativeCycle && showSteps) {
222         logStep(result.logs, "Khong phat hien chu trinh am.");
223     }
224
225     if (showSteps) {
226         logStep(result.logs, "");
227         logStep(result.logs, "==== KHOANG CACH CUOI ===");
228         for (int i = 0; i < V; i++) {
229             if (result.distances[i] == INF) {
230                 logStep(result.logs, graph.getVertexLabel(i) + " = INF (khong toi duoc)");
231             } else {
232                 logStep(result.logs, graph.getVertexLabel(i) + " = " + std::to_string(
233                     result.distances[i]));
234             }
235         }
236     }
237
238     result.success = !result.hasNegativeCycle;
239     return result;
240 }
241 std::vector<int> Algorithms::getShortestPath(const PathResult& result, int destination)
242     const {
243     if (destination < 0 || destination >= result.previousVertex.size()) {
244         return {};
245     }
246     if (result.distances.empty()) {
247         return {};
248     }
249     if (result.distances[destination] == std::numeric_limits<int>::max()) {
250         return {};
251     }
252     return reconstructPath(destination, result.previousVertex);
253 }
254 int Algorithms::getDistance(const PathResult& result, int destination) const {
255     if (destination < 0 || destination >= result.distances.size()) {
256         return -1;
257     }
258     return result.distances[destination];
259 }
```

## 2.4 Mã nguồn chương trình so sánh : comparison.cpp

```

1 #include "../lib/Comparison.h"
2 #include <cmath>
3
4 // ===== CONSTRUCTOR =====
5 Comparison::Comparison(const Graph& g) : graph(g), algorithms(g) {}
6
7 // ===== PERFORMANCE MEASUREMENT =====
8 PerformanceMetrics Comparison::measureAlgorithm(int startVertex, AlgorithmType type) {
9     PerformanceMetrics metrics;
10    int V = graph.getVertexCount();
11    int E = graph.getEdgeCount();
12
13    if (type == AlgorithmType::DIJKSTRA) {
```

```

14     metrics.algorithmName = "Dijkstra";
15
16     // Check for negative weights
17     if (graph.hasNegativeWeights()) {
18         metrics.success = false;
19         metrics.executionTimeUs = 0;
20         return metrics;
21     }
22
23     auto start = std::chrono::high_resolution_clock::now();
24     PathResult result = algorithms.dijkstra(startVertex, false);
25     auto end = std::chrono::high_resolution_clock::now();
26
27     metrics.executionTimeUs = std::chrono::duration_cast<std::chrono::microseconds>(
28     end - start).count();
29     metrics.distancesCalculated = result.distances.size();
30     metrics.memoryUsageBytes = (V * sizeof(int) * 2) + (E * sizeof(Edge));
31     metrics.complexity = E * std::log(V);
32     metrics.success = result.success;
33
34 } else if (type == AlgorithmType::BELLMAN_FORD) {
35     metrics.algorithmName = "Bellman-Ford";
36
37     auto start = std::chrono::high_resolution_clock::now();
38     PathResult result = algorithms.bellmanFord(startVertex, false);
39     auto end = std::chrono::high_resolution_clock::now();
40
41     metrics.executionTimeUs = std::chrono::duration_cast<std::chrono::microseconds>(
42     end - start).count();
43     metrics.distancesCalculated = result.distances.size();
44     metrics.memoryUsageBytes = (V * sizeof(int) * 2) + (E * sizeof(Edge));
45     metrics.complexity = V * E;
46     metrics.success = result.success && !result.hasNegativeCycle;
47 }
48
49     return metrics;
50 }
51 // ===== COMPARISON REPORT =====
52 ComparisonReport Comparison::comparePerformance(int startVertex, AlgorithmType type) {
53     ComparisonReport report;
54     report.startVertex = startVertex;
55     report.V = graph.getVertexCount();
56     report.E = graph.getEdgeCount();
57
58     report.logs.push_back("=====");
59     report.logs.push_back("BAO CAO SO SANH HIEU NANG");
60     report.logs.push_back("=====");
61     report.logs.push_back("Dinh bat dau: " + std::to_string(startVertex + 1) + " So di
nh (V): " + std::to_string(report.V) + " So canh (E): " + std::to_string(report.E));
62
63     if (type == AlgorithmType::DIJKSTRA || type == AlgorithmType::BOTH) {
64         auto metrics = measureAlgorithm(startVertex, AlgorithmType::DIJKSTRA);
65         report.metrics.push_back(metrics);
66     }
67
68     if (type == AlgorithmType::BELLMAN_FORD || type == AlgorithmType::BOTH) {
69         auto metrics = measureAlgorithm(startVertex, AlgorithmType::BELLMAN_FORD);
70         report.metrics.push_back(metrics);

```

```

70 }
71
72 if (type == AlgorithmType::BOTH && report.metrics.size() == 2) {
73     const auto& d = report.metrics[0];
74     const auto& b = report.metrics[1];
75
76     const int labelW = 16;
77     const int colW = 20;
78
79     auto utf8Len = [](const std::string& s) {
80         int count = 0;
81         for (unsigned char c : s) {
82             if ((c & 0xC0) != 0x80) {
83                 count++;
84             }
85         }
86         return count;
87     };
88
89     auto fit = [&](const std::string& s, int width) {
90         if (width <= 0) return std::string();
91         int len = utf8Len(s);
92         if (len == width) return s;
93         if (len < width) return s + std::string(width - len, ' ');
94
95         std::string out;
96         out.reserve(s.size());
97         int count = 0;
98         for (size_t i = 0; i < s.size() && count < width; i++) {
99             unsigned char c = static_cast<unsigned char>(s[i]);
100            if ((c & 0xC0) != 0x80) {
101                if (count >= width) break;
102                count++;
103            }
104            out.push_back(s[i]);
105        }
106        int outLen = utf8Len(out);
107        if (outLen < width) {
108            out += std::string(width - outLen, ' ');
109        }
110        return out;
111    };
112
113    auto row = [&](const std::string& label, const std::string& dv, const std::string
& bv) {
114        return std::string("|") + fit(label, labelW) + "|"
115            + fit(dv, colW) + "|" + fit(bv, colW) + "|";
116    };
117
118    std::string border = "+" + std::string(labelW, '-') +
119                    "+" + std::string(colW, '-') +
120                    "+" + std::string(colW, '-') + "+";
121
122    auto fmtStatus = [](bool ok) {
123        return ok ? std::string("Thanh cong") : std::string("That bai");
124    };
125
126    auto fmtComplexity = [](double value, const std::string& form) {
127        return form + "      0(" + std::to_string(static_cast<int>(value)) + ")";

```

```

128     };
129
130     report.logs.push_back(border);
131     report.logs.push_back(row("", "DIJKSTRA", "BELLMAN-FORD"));
132     report.logs.push_back(border);
133     report.logs.push_back(row("Thoi gian chay", std::to_string(d.executionTimeUs) + " us",
134                             std::to_string(b.executionTimeUs) + " us"));
135     report.logs.push_back(row("Bo nho dung", std::to_string(d.memoryUsageBytes) + " bytes",
136                             std::to_string(b.memoryUsageBytes) + " bytes"));
137     report.logs.push_back(row("Do phuc tap",
138                             fmtComplexity(d.complexity, "O(E log V)),
139                             fmtComplexity(b.complexity, "O(V      E)")));
140     report.logs.push_back(row("Trang thai", fmtStatus(d.success), fmtStatus(b.success
141 ))));
141     report.logs.push_back(border);
142 }
143 // else if (type == AlgorithmType::DIJKSTRA && report.metrics.size() == 1) {
144 //     const auto& metrics = report.metrics[0];
145 //     report.logs.push_back("--- THUAT TOAN DIJKSTRA ---");
146 //     report.logs.push_back("Thoi gian chay: " + std::to_string(metrics.
147 executionTimeUs) + " us");
148 //     report.logs.push_back("Bo nho su dung: " + std::to_string(metrics.
149 memoryUsageBytes) + " bytes");
150 //     report.logs.push_back("Do phuc tap: O(E log V)      O(" + std::to_string((int)
151 metrics.complexity) + ")");
152 //     report.logs.push_back("Trang thai: " + std::string(metrics.success ? "Thanh co
153 ng" : "That bai"));
154 //     report.logs.push_back("");
155 // } else if (type == AlgorithmType::BELLMAN_FORD && report.metrics.size() == 1) {
156 //     const auto& metrics = report.metrics[0];
157 //     report.logs.push_back("--- THUAT TOAN BELLMAN-FORD ---");
158 //     report.logs.push_back("Thoi gian chay: " + std::to_string(metrics.
159 executionTimeUs) + " us");
160 //     report.logs.push_back("Bo nho su dung: " + std::to_string(metrics.
161 memoryUsageBytes) + " bytes");
162 //     report.logs.push_back("Do phuc tap: O(V      E)      O(" + std::to_string((int)
163 metrics.complexity) + ")");
164 //     report.logs.push_back("Trang thai: " + std::string(metrics.success ? "Thanh co
165 ng" : "That bai"));
166 //     report.logs.push_back("");
167 // }
168
169 if (type == AlgorithmType::BOTH && report.metrics.size() == 2) {
170     report.logs.push_back("                                         --- SO SANH ---");
171     auto dijkstraTime = report.metrics[0].executionTimeUs;
172     auto bellmanTime = report.metrics[1].executionTimeUs;
173
174     if (bellmanTime > 0 && dijkstraTime > 0) {
175         double ratio = static_cast<double>(bellmanTime) / dijkstraTime;
176         report.logs.push_back("Bellman-Ford cham hon Dijkstra " + std::to_string(
177 ratio) + " lan");
178     } else {
179         report.logs.push_back("Khong du do chenh lech thoi gian de so sanh.");
180     }
181     report.logs.push_back("");
182 }
```

```

175     return report;
176 }

```

## .2.5 Mã nguồn chương trình vẽ giao diện : GUI.cpp

```

1 #include "../lib/GUI.h"
2 #include <iostream>
3 #include <cctype>
4 #include <sstream>
5
6 namespace {
7 const int OUTER_MARGIN = 30;
8 const int HEADER_GAP = 20;
9 const int FRAME_PADDING = 24;
10
11 const int HEADER_LINES = 8;
12 const int MENU_LINES = 8;
13 const int CHOICE_LINES = 2;
14 const int LEFT_INDENT_SPACES = 30;
15
16 const char* MENU_PROMPT = "Nhập lựa chọn (1-6): ";
17 const char* PRESS_ANY_KEY = "Nhấn phím bất kỳ để tiếp tục...";
18
19 int approxCharWidth() {
20     return WINDOW_WIDTH / 120;
21 }
22
23 int approxLineHeight() {
24     return WINDOW_HEIGHT / 40;
25 }
26
27 int headerHeight() {
28     return approxLineHeight() * HEADER_LINES;
29 }
30
31 int menuHeight() {
32     return approxLineHeight() * MENU_LINES;
33 }
34
35 int choiceHeight() {
36     return approxLineHeight() * CHOICE_LINES;
37 }
38
39 int utf8CharCount(const std::string& text) {
40     int count = 0;
41     for (unsigned char c : text) {
42         if ((c & 0xC0) != 0x80) {
43             count++;
44         }
45     }
46     return count;
47 }
48
49 int approxTextWidth(const std::string& text) {
50     return utf8CharCount(text) * approxCharWidth();
51 }

```

```

53 std::vector<std::string> wrapLineToWidth(const std::string& line, int maxWidth) {
54     std::vector<std::string> out;
55     if (maxWidth <= 0) {
56         out.push_back(line);
57         return out;
58     }
59     if (line.empty()) {
60         out.push_back("");
61         return out;
62     }
63
64     size_t pos = line.find_first_not_of(' ');
65     if (pos == std::string::npos) {
66         out.push_back(line);
67         return out;
68     }
69
70     std::string indent = line.substr(0, pos);
71     std::string content = line.substr(pos);
72     std::istringstream iss(content);
73     std::string word;
74     std::string current = indent;
75     bool hasWord = false;
76
77     while (iss >> word) {
78         std::string candidate = hasWord ? (current + " " + word) : (current + word);
79         if (approxTextWidth(candidate) <= maxWidth || !hasWord) {
80             current = candidate;
81             hasWord = true;
82         } else {
83             out.push_back(current);
84             current = indent + word;
85             hasWord = true;
86         }
87     }
88
89     if (hasWord) {
90         out.push_back(current);
91     } else {
92         out.push_back(line);
93     }
94     return out;
95 }
96
97 std::vector<std::string> wrapLinesToWidth(const std::vector<std::string>& lines, int
98     maxWidth) {
99     std::vector<std::string> out;
100    for (const auto& line : lines) {
101        auto parts = wrapLineToWidth(line, maxWidth);
102        out.insert(out.end(), parts.begin(), parts.end());
103    }
104    return out;
105 }
106 void drawCenteredText(int centerX, int y, const std::string& text) {
107     int x = centerX - (approxTextWidth(text) / 2);
108     outtextxy(x, y, (char*)text.c_str());
109 }
110

```

```

111 void drawLeftAlignedText(int leftX, int y, const std::string& text) {
112     outtextxy(leftX, y, (char*)text.c_str());
113 }
114
115 int leftIndentX(int frameLeft) {
116     return frameLeft + 4 + LEFT_INDENT_SPACES * approxCharWidth();
117 }
118
119 int bottomIndentX(int frameLeft) {
120     return leftIndentX(frameLeft);
121 }
122
123 std::string trim(const std::string& s) {
124     size_t start = s.find_first_not_of(' ');
125     if (start == std::string::npos) return "";
126     size_t end = s.find_last_not_of(' ');
127     return s.substr(start, end - start + 1);
128 }
129
130 bool tryParseInt(const std::string& text, int& value) {
131     std::string s = trim(text);
132     if (s.empty()) return false;
133     try {
134         size_t idx = 0;
135         int v = std::stoi(s, &idx);
136         if (idx != s.size()) return false;
137         value = v;
138         return true;
139     } catch (...) {
140         return false;
141     }
142 }
143
144 bool tryParseEdgeLine(const std::string& text, int& u, int& v, int& w) {
145     std::istringstream iss(text);
146     if (!(iss >> u >> v >> w)) {
147         return false;
148     }
149     std::string extra;
150     if (iss >> extra) {
151         return false;
152     }
153     return true;
154 }
155
156 enum class InputFilter { Any, Integer };
157
158 void drawInputField(int x, int y, int maxLen, const std::string& value) {
159     if (maxLen < 1) maxLen = 1;
160     std::string blank(static_cast<size_t>(maxLen), ' ');
161     outtextxy(x, y, (char*)blank.c_str());
162     std::string clipped = value;
163     if ((int)clipped.size() > maxLen) {
164         clipped = clipped.substr(0, static_cast<size_t>(maxLen));
165     }
166     outtextxy(x, y, (char*)clipped.c_str());
167 }
168
169 void clearTextLine(int x, int y, int maxLen) {

```

```

170     if (maxLength < 1) maxLength = 1;
171     std::string blank(static_cast<size_t>(maxLength), ' ');
172     outtextxy(x, y, (char*)blank.c_str());
173 }
174
175 std::string readLineAt(int x, int y, int maxLength, InputFilter filter) {
176     std::string value;
177     setcolor(COLOR_TEXT);
178     drawInputField(x, y, maxLength, value);
179     while (true) {
180         int c = getch();
181         if (c == '\r' || c == '\n') {
182             break;
183         }
184         if (c == 8 || c == 127) {
185             if (!value.empty()) {
186                 value.pop_back();
187             }
188         } else if (c >= 32 && c <= 126) {
189             char ch = static_cast<char>(c);
190             bool accept = true;
191             if (filter == InputFilter::Integer) {
192                 if (std::isdigit(static_cast<unsigned char>(ch))) {
193                     accept = true;
194                 } else if ((ch == '-') || (ch == '+') && value.empty()) {
195                     accept = true;
196                 } else {
197                     accept = false;
198                 }
199             }
200             if (accept && (int)value.size() < maxLength) {
201                 value.push_back(ch);
202             }
203         }
204         drawInputField(x, y, maxLength, value);
205     }
206     return value;
207 }
208
209 void drawHorizontalRule(int leftX, int rightX, int y) {
210     int charW = approxCharWidth();
211     int count = (rightX - leftX) / (charW > 0 ? charW : 1);
212     if (count < 2) count = 2;
213     std::string rule;
214     rule.reserve(static_cast<size_t>(count));
215     rule.push_back('+');
216     if (count > 2) {
217         rule.append(static_cast<size_t>(count - 2), '-');
218     }
219     rule.push_back('+');
220     outtextxy(leftX, y, (char*)rule.c_str());
221 }
222
223 void drawFrameBox(const std::string& title, int left, int top, int right, int bottom,
224                     int& centerX, int& innerLeft, int& innerTop) {
225     setcolor(COLOR_BUTTON);
226     rectangle(left, top, right, bottom);
227     rectangle(left + 4, top + 4, right - 4, bottom - 4);
228 }
```

```

229     centerX = (left + right) / 2;
230     int lineH = approxLineHeight();
231     int titleY = top + 8;
232
233     if (!title.empty()) {
234         setcolor(COLOR_TEXT);
235         drawCenteredText(centerX, titleY, title);
236         innerTop = titleY + lineH;
237     } else {
238         innerTop = top + FRAME_PADDING;
239     }
240
241     innerLeft = leftIndentX(left);
242 }
243
244 void drawHeaderFrame(int screenW) {
245     int extra = 2 * approxCharWidth();
246     int left = OUTER_MARGIN - extra;
247     int right = screenW - OUTER_MARGIN + extra;
248     if (left < 0) left = 0;
249     if (right > screenW) right = screenW;
250     int top = OUTER_MARGIN;
251     int bottom = top + headerHeight();
252
253     int centerX = 0;
254     int innerLeft = 0;
255     int innerTop = 0;
256     drawFrameBox("", left, top, right, bottom, centerX, innerLeft, innerTop);
257
258     int lineH = approxLineHeight();
259     int y = top + 12;
260
261     setcolor(COLOR_TEXT);
262     y += lineH;
263     drawCenteredText(centerX, y, "TRUONG DAI HOC BACH KHOA - DAI HOC DA NANG");
264     y += lineH + 4;
265     // drawCenteredText(centerX, y, "|");
266     // y += lineH + 2;
267
268     setcolor(YELLOW);
269     drawCenteredText(centerX, y, "PBL1 : DO AN LAP TRINH TINH TOAN");
270     y += lineH + 14;
271     const int paddingX = 40;
272
273     setcolor(COLOR_TEXT);
274     const std::string leftText = "Ten SV: Nguyen Huu Rin";
275     const std::string rightText = "GVHD: Nguyen Van Hieu";
276     outtextxy(390, y, (char*)leftText.c_str());
277     outtextxy(right - 7*paddingX - approxTextWidth(rightText), y, (char*)rightText.c_str());
278     y += lineH + 4;
279     drawCenteredText(centerX-25, y, "Huynh Nguyen Hong Nhi");
280 }
281
282 void drawContentFrame(const std::string& title, int screenW, int screenH,
283                         int& left, int& top, int& right, int& bottom, int& centerX, int&
284                         innerLeft, int& innerTop) {
285     int extra = 2 * approxCharWidth();
286     left = OUTER_MARGIN - extra;

```

```

286     right = screenW - OUTER_MARGIN + extra;
287     if (left < 0) left = 0;
288     if (right > screenW) right = screenW;
289     top = OUTER_MARGIN + headerHeight() + HEADER_GAP;
290     bottom = screenH - OUTER_MARGIN;
291     drawFrameBox(title, left, top, right, bottom, centerX, innerLeft, innerTop);
292 }
293
294 // void drawContentFrameCustom(const std::string& title, int screenW, int screenH, int
295 // outerMargin, int headerGap,
296 //                               int& left, int& top, int& right, int& bottom, int& centerX
297 // , int& innerLeft, int& innerTop) {
298 //     left = outerMargin;
299 //     right = screenW - outerMargin;
300 //     top = outerMargin + headerHeight() + headerGap;
301 //     bottom = screenH - outerMargin;
302 //     drawFrameBox(title, left, top, right, bottom, centerX, innerLeft, innerTop);
303 // }
304
305 void drawLogFrame(const std::string& title, int left, int top, int right, int bottom,
306                     int& centerX, int& innerLeft, int& innerTop) {
307     setcolor(COLOR_BUTTON);
308     rectangle(left, top, right, bottom);
309
310     centerX = (left + right) / 2;
311     int lineH = approxLineHeight();
312     if (!title.empty()) {
313         setcolor(COLOR_TEXT);
314         int titleY = top + 2;
315         drawCenteredText(centerX, titleY, title);
316     }
317
318     innerLeft = left + 2;
319     innerTop = top + 2 * lineH;
320 }
321
322 GUI::GUI() : screenWidth(WINDOW_WIDTH), screenHeight(WINDOW_HEIGHT) {
323 }
324
325 GUI::~GUI() {}
326
327 void GUI::drawMenu() {
328     clearScreen();
329     setbkcolor(COLOR_BACKGROUND);
330
331     drawHeaderFrame(WINDOW_WIDTH);
332
333     int left = OUTER_MARGIN;
334     int right = WINDOW_WIDTH - OUTER_MARGIN;
335     int menuTop = OUTER_MARGIN + headerHeight() + HEADER_GAP;
336     int menuBottom = menuTop + menuHeight();
337
338     int centerX = 0;
339     int innerLeft = 0;
340     int innerTop = 0;
341     drawFrameBox("CHUONG TRINH TIM DUONG DI NGAN NHAT", left, menuTop, right, menuBottom,
342                 centerX, innerLeft, innerTop);
343 }
```

```

343     int lineH = approxLineHeight();
344     int y = innerTop;
345
346     setcolor(YELLOW);
347     std :: cout << '\n';
348     int menuTextX = leftIndentX(left);
349     drawLeftAlignedText(menuTextX, y, "[1]. Khoi tao/Nap do thi (tu file)"); y += lineH;
350     drawLeftAlignedText(menuTextX, y, "[2]. Chay thuat toan Dijkstra"); y += lineH;
351     drawLeftAlignedText(menuTextX, y, "[3]. Chay thuat toan Bellman-Ford"); y += lineH;
352     drawLeftAlignedText(menuTextX, y, "[4]. So sanh hieu nang"); y += lineH;
353     drawLeftAlignedText(menuTextX, y, "[5]. Truc quan hoa (Python)"); y += lineH;
354     drawLeftAlignedText(menuTextX, y, "[6]. Thoat"); y += lineH;
355     std :: cout << '\n';
356
357     setcolor(COLOR_BUTTON);
358     drawHorizontalRule(left + 4, right - 4, y);
359
360     int choiceTop = menuBottom + HEADER_GAP;
361     int choiceBottom = choiceTop + choiceHeight();
362     int choiceCenterX = 0;
363     int choiceInnerLeft = 0;
364     int choiceInnerTop = 0;
365     drawFrameBox("", left, choiceTop, right, choiceBottom,
366                  choiceCenterX, choiceInnerLeft, choiceInnerTop);
367
368     int choiceTextY = choiceTop + (choiceHeight() - lineH) / 2;
369     setcolor(COLOR_TEXT);
370     drawLeftAlignedText(choiceInnerLeft, choiceTextY, MENU_PROMPT);
371 }
372
373
374 void GUI::drawComparisonScreen(const std::vector<std::string>& logs) {
375     size_t index = 0;
376     int lineH = approxLineHeight();
377     int maxLinesPerPage = 0;
378     std::vector<std::string> wrappedLogs = logs;
379     const int safetyLines = 3;
380
381     while (true) {
382         clearScreen();
383         setbkcolor(COLOR_BACKGROUND);
384
385         drawHeaderFrame(WINDOW_WIDTH);
386
387         int left, top, right, bottom, centerX, innerLeft, innerTop;
388         drawContentFrame("SO SANH THUAT TOAN", WINDOW_WIDTH, WINDOW_HEIGHT,
389                         left, top, right, bottom, centerX, innerLeft, innerTop);
390
391         if (wrappedLogs.empty()) {
392             wrappedLogs.push_back("");
393         }
394
395         if (maxLinesPerPage == 0) {
396             int yProbe = innerTop + 6;
397             while (yProbe < bottom - lineH * 2) {
398                 maxLinesPerPage++;
399                 yProbe += lineH + 4;
400             }
401             if (maxLinesPerPage < 1) maxLinesPerPage = 1;

```

```

402         if (maxLinesPerPage > safetyLines) {
403             maxLinesPerPage -= safetyLines;
404         }
405     }
406
407     setcolor(LIGHTGREEN);
408     int yPos = innerTop + 6;
409     int count = 0;
410     while (index < wrappedLogs.size() && count < maxLinesPerPage) {
411         outtextxy(innerLeft, yPos, (char*)wrappedLogs[index].c_str());
412         yPos += lineH + 4;
413         index++;
414         count++;
415     }
416
417     bool hasMore = index < wrappedLogs.size();
418     setcolor(LIGHTCYAN);
419     if (hasMore) {
420         drawCenteredText(centerX, bottom - lineH - 4, "Nhan phim bat ky de xem tiep
...");
421         getch();
422         continue;
423     }
424
425     drawCenteredText(centerX, bottom - lineH - 4, "Nhan phim bat ky de quay lai menu
...");
426     break;
427 }
428 }
429
430 void GUI::clearScreen() {
431     cleardevice();
432 }
433
434 int GUI::promptMenuChoice() {
435     while (true) {
436         drawMenu();
437         int lineH = approxLineHeight();
438
439         int left = OUTER_MARGIN;
440         int menuTop = OUTER_MARGIN + headerHeight() + HEADER_GAP;
441         int menuBottom = menuTop + menuHeight();
442         int choiceTop = menuBottom + HEADER_GAP;
443
444         int promptY = choiceTop + (choiceHeight() - lineH) / 2;
445         int inputX = leftIndentX(left) + approxTextWidth(MENU_PROMPT) + approxCharWidth()
;
446
447         std::string input = readLineAt(inputX, promptY, 3, InputFilter::Integer);
448         int value = 0;
449         if (tryParseInt(input, value) && value >= 1 && value <= 6) {
450             return value;
451         }
452
453         setcolor(LIGHTRED);
454         drawLeftAlignedText(left + FRAME_PADDING, promptY + lineH, "Du lieu khong hop le.
Vui long thu lai.");
455         waitForKey();
456     }
}

```

```

457 }
458
459 int GUI::promptChoice(const std::string& title, const std::vector<std::string>& options,
460                         const std::string& prompt, int minValue, int maxValue) {
461     std::string error;
462     while (true) {
463         clearScreen();
464         setbkcolor(COLOR_BACKGROUND);
465         drawHeaderFrame(WINDOW_WIDTH);
466
467         int left, top, right, bottom, centerX, innerLeft, innerTop;
468         drawContentFrame(title, WINDOW_WIDTH, WINDOW_HEIGHT,
469                           left, top, right, bottom, centerX, innerLeft, innerTop);
470
471         int lineH = approxLineHeight();
472         int y = innerTop + 4;
473
474         setcolor(YELLOW);
475         for (const auto& opt : options) {
476             drawLeftAlignedText(innerLeft, y, opt);
477             y += lineH;
478         }
479
480         y += lineH / 2;
481         setcolor(COLOR_TEXT);
482         drawLeftAlignedText(innerLeft, y, prompt);
483
484         int inputX = innerLeft + approxTextWidth(prompt) + approxCharWidth();
485         int inputY = y;
486
487         if (!error.empty()) {
488             setcolor(LIGHTRED);
489             drawLeftAlignedText(bottomIndentX(left), y + lineH, error);
490         }
491
492         std::string input = readLineAt(inputX, inputY, 6, InputFilter::Integer);
493         int value = 0;
494         if (tryParseInt(input, value) && value >= minValue && value <= maxValue) {
495             return value;
496         }
497
498         error = "Du lieu khong hop le. Vui long thu lai.";
499     }
500 }
501
502 int GUI::promptInt(const std::string& title, const std::string& prompt, int minValue, int
503                     maxValue) {
504     std::string error;
505     while (true) {
506         clearScreen();
507         setbkcolor(COLOR_BACKGROUND);
508         drawHeaderFrame(WINDOW_WIDTH);
509
510         int left, top, right, bottom, centerX, innerLeft, innerTop;
511         drawContentFrame(title, WINDOW_WIDTH, WINDOW_HEIGHT,
512                           left, top, right, bottom, centerX, innerLeft, innerTop);
513
514         int lineH = approxLineHeight();
515         int y = innerTop + 6;

```

```

515     setcolor(COLOR_TEXT);
516     drawLeftAlignedText(innerLeft, y, prompt);
517
518     int inputX = innerLeft + approxTextWidth(prompt) + approxCharWidth();
519     int inputY = y;
520
521     if (!error.empty()) {
522         setcolor(LIGHTRED);
523         drawLeftAlignedText(bottomIndentX(left), y + lineH, error);
524     }
525
526     std::string input = readLineAt(inputX, inputY, 8, InputFilter::Integer);
527     int value = 0;
528     if (tryParseInt(input, value) && value >= minValue && value <= maxValue) {
529         return value;
530     }
531
532     error = "Du lieu khong hop le. Vui long thu lai.";
533 }
534
535 }
536
537 std::string GUI::promptLine(const std::string& title, const std::string& prompt,
538                             const std::string& defaultValue) {
539     clearScreen();
540     setbkcolor(COLOR_BACKGROUND);
541     drawHeaderFrame(WINDOW_WIDTH);
542
543     int left, top, right, bottom, centerX, innerLeft, innerTop;
544     drawContentFrame(title, WINDOW_WIDTH, WINDOW_HEIGHT,
545                      left, top, right, bottom, centerX, innerLeft, innerTop);
546
547     int y = innerTop + 6;
548     setcolor(COLOR_TEXT);
549     drawLeftAlignedText(innerLeft, y, prompt);
550
551     int inputX = innerLeft + approxTextWidth(prompt) + approxCharWidth();
552     int inputY = y;
553     int maxLen = (right - innerLeft) / (approxCharWidth() > 0 ? approxCharWidth() : 1) -
554     2;
555     if (maxLen < 8) maxLen = 8;
556
557     std::string input = readLineAt(inputX, inputY, maxLen, InputFilter::Any);
558     if (input.empty()) {
559         return defaultValue;
560     }
561     return input;
562 }
563
564 bool GUI::promptYesNo(const std::string& title, const std::string& prompt) {
565     while (true) {
566         std::string input = promptLine(title, prompt + " (y/n): ");
567         if (input.empty()) {
568             continue;
569         }
570         char c = static_cast<char>(std::tolower(static_cast<unsigned char>(input[0])));
571         if (c == 'y') return true;
572         if (c == 'n') return false;
573         showMessage(title, {"Vui long nhap y hoac n."});

```

```

573     }
574 }
575
576 void GUI::promptStartEnd(const std::string& title, int minValue, int maxValue,
577                           int& startValue, int& endValue) {
578     clearScreen();
579     setbkcolor(COLOR_BACKGROUND);
580     drawHeaderFrame(WINDOW_WIDTH);
581
582     int left, top, right, bottom, centerX, innerLeft, innerTop;
583     drawContentFrame(title, WINDOW_WIDTH, WINDOW_HEIGHT,
584                      left, top, right, bottom, centerX, innerLeft, innerTop);
585
586     int lineH = approxLineHeight();
587     int y = innerTop + 6;
588
589     setcolor(COLOR_TEXT);
590     std::string promptStart = "Dinh bat dau (" + std::to_string(minValue) + ".." + std::
591     to_string(maxValue) + "): ";
592     std::string promptEnd = "Dinh ket thuc (" + std::to_string(minValue) + ".." + std::
593     to_string(maxValue) + "): ";
594
595     int charW = approxCharWidth();
596     if (charW < 1) charW = 1;
597     int errorX = innerLeft;
598     int errorY = bottom - lineH - 6;
599     int errorMaxChars = (right - innerLeft) / charW;
600
601     auto clearError = [&]() {
602         setcolor(COLOR_TEXT);
603         clearTextLine(errorX, errorY, errorMaxChars);
604     };
605
606     auto showError = [&](const std::string& msg) {
607         clearError();
608         setcolor(LIGHTRED);
609         drawLeftAlignedText(errorX, errorY, msg);
610         setcolor(COLOR_TEXT);
611     };
612
613     drawLeftAlignedText(innerLeft, y, promptStart);
614     int inputXStart = innerLeft + approxTextWidth(promptStart) + approxCharWidth();
615     while (true) {
616         std::string input = readLineAt(inputXStart, y, 8, InputFilter::Integer);
617         int value = 0;
618         if (tryParseInt(input, value) && value >= minValue && value <= maxValue) {
619             startValue = value;
620             clearError();
621             break;
622         }
623         showError("Gia tri dinh bat dau khong hop le.");
624
625         y += lineH;
626         drawLeftAlignedText(innerLeft, y, promptEnd);
627         int inputXEnd = innerLeft + approxTextWidth(promptEnd) + approxCharWidth();
628         while (true) {
629             std::string input = readLineAt(inputXEnd, y, 8, InputFilter::Integer);
630             int value = 0;
631             if (tryParseInt(input, value) && value >= minValue && value <= maxValue) {

```

```

630         endValue = value;
631         clearError();
632         break;
633     }
634     showError("Gia tri dinh ket thuc khong hop le.");
635 }
636
637
638 void GUI::promptGraphInput(bool isDirected, int& numVertices, int& numEdges,
639                             std::vector<std::tuple<int, int, int>>& edges) {
640     edges.clear();
641     numVertices = 0;
642     numEdges = 0;
643
644     clearScreen();
645     setbkcolor(COLOR_BACKGROUND);
646     drawHeaderFrame(WINDOW_WIDTH);
647
648     int left, top, right, bottom, centerX, innerLeft, innerTop;
649     drawContentFrame("TAO DO THI", WINDOW_WIDTH, WINDOW_HEIGHT,
650                      left, top, right, bottom, centerX, innerLeft, innerTop);
651
652     int lineH = approxLineHeight();
653     int y = innerTop + 6;
654
655     setcolor(YELLOW);
656     drawLeftAlignedText(innerLeft, y, "Nhap thong tin do thi:");
657     y += lineH + 4;
658
659     setcolor(COLOR_TEXT);
660     int charW = approxCharWidth();
661     if (charW < 1) charW = 1;
662     int errorX = innerLeft;
663     int errorY = bottom - lineH - 6;
664     int errorMaxChars = (right - innerLeft) / charW;
665
666     auto clearError = [&]() {
667         setcolor(COLOR_TEXT);
668         clearTextLine(errorX, errorY, errorMaxChars);
669     };
670     auto showError = [&](const std::string& msg) {
671         clearError();
672         setcolor(LIGHTRED);
673         drawLeftAlignedText(errorX, errorY, msg);
674         setcolor(COLOR_TEXT);
675     };
676
677     std::string promptV = "So dinh (1-100): ";
678     drawLeftAlignedText(innerLeft, y, promptV);
679     int inputXV = innerLeft + approxTextWidth(promptV) + approxCharWidth();
680     while (true) {
681         std::string input = readLineAt(inputXV, y, 6, InputFilter::Integer);
682         int value = 0;
683         if (tryParseInt(input, value) && value >= 1 && value <= 100) {
684             numVertices = value;
685             clearError();
686             break;
687         }
688         showError("So dinh khong hop le (1-100).");

```

```

689 }
690
691     y += lineH;
692
693     int maxEdges = numVertices * numVertices;
694     std::string promptE = "So canh (0-" + std::to_string(maxEdges) + "): ";
695     drawLeftAlignedText(innerLeft, y, promptE);
696     int inputXE = innerLeft + approxTextWidth(promptE) + approxCharWidth();
697
698     while (true) {
699         std::string input = readLineAt(inputXE, y, 8, InputFilter::Integer);
700         int value = 0;
701         if (tryParseInt(input, value) && value >= 0 && value <= maxEdges) {
702             int yEdgesStart = y + lineH + 4;
703             if (!isDirected) {
704                 yEdgesStart += lineH + 4;
705             }
706             int availableLines = (errorY - lineH) - yEdgesStart;
707             int maxEdgeLines = INT_MAX;
708             if (maxEdgeLines < 1) maxEdgeLines = 1;
709             if (value > maxEdgeLines) {
710                 showError("So canh qua nhieu de nhap trong 1 khung. Toi da " + std::
711                           to_string(maxEdgeLines) + ".");
712                 continue;
713             }
714             numEdges = value;
715             clearError();
716             break;
717         }
718         showError("So canh khong hop le.");
719     }
720     y += lineH + 4;
721
722     if (!isDirected) {
723         setcolor(LIGHTRED);
724         drawLeftAlignedText(innerLeft, y, "Vo huong: nhap moi canh 1 lan (u v w).");
725         y += lineH + 4;
726         setcolor(COLOR_TEXT);
727     }
728
729     for (int i = 0; i < numEdges; i++) {
730         std::string prompt = "Canh " + std::to_string(i + 1) + " (u v w): ";
731         drawLeftAlignedText(innerLeft, y, prompt);
732         int inputX = innerLeft + approxTextWidth(prompt) + approxCharWidth();
733         while (true) {
734             std::string input = readLineAt(inputX, y, 24, InputFilter::Any);
735             int u = 0, v = 0, w = 0;
736             if (tryParseEdgeLine(input, u, v, w) &&
737                 u >= 1 && u <= numVertices &&
738                 v >= 1 && v <= numVertices &&
739                 w >= -1000000 && w <= 1000000) {
740                 edges.emplace_back(u - 1, v - 1, w);
741                 clearError();
742                 break;
743             }
744             showError("Canh khong hop le. Dinh dang: u v w, u/v trong [1.." + std::
745                           to_string(numVertices) + "].");
746     }

```

```

746         y += lineH;
747     }
748 }
749
750 void GUI::showAlgorithmLogs(const std::string& title, const std::vector<std::string>&
751   logs) {
752     size_t index = 0;
753     int lineH = approxLineHeight();
754     int lineStep = lineH;
755     std::vector<std::string> wrappedLogs;
756     const int margin = 1;
757     const int left = margin;
758     const int right = WINDOW_WIDTH - margin;
759     const int top = margin;
760     const int bottom = WINDOW_HEIGHT - margin;
761     const int indentSpaces = 25;
762     int indent = indentSpaces * approxCharWidth();
763     int maxAllowedIndent = (right - left) / 2;
764     if (maxAllowedIndent < 0) maxAllowedIndent = 0;
765     if (indent > maxAllowedIndent) indent = maxAllowedIndent;
766
767     int innerLeftBase = left + 2 + indent;
768     int innerTopBase = top + 2 * lineH;
769
770     int maxWidth = right - innerLeftBase - 2;
771     wrappedLogs = wrapLinesToWidth(logs, maxWidth);
772     if (wrappedLogs.empty()) {
773       wrappedLogs.push_back("");
774     }
775
776     int usableBottom = bottom - lineH;
777     int maxLinesPerPage = (usableBottom - innerTopBase) / lineStep + 1;
778     if (maxLinesPerPage < 1) maxLinesPerPage = 1;
779     const int safetyLines = 15;
780     if (maxLinesPerPage > safetyLines) {
781       maxLinesPerPage -= safetyLines;
782     }
783     int totalPages = static_cast<int>((wrappedLogs.size() + maxLinesPerPage - 1) /
784     maxLinesPerPage);
785     if (totalPages < 1) totalPages = 1;
786
787     int page = 0;
788     while (index < wrappedLogs.size()) {
789       page++;
790       clearScreen();
791       setbkcolor(COLOR_BACKGROUND);
792
793       std::string pageTitle = title;
794       if (totalPages > 1) {
795         pageTitle += " (Trang " + std::to_string(page) + "/" + std::to_string(
796         totalPages) + ")";
797       }
798
799       int centerX = 0;
800       int innerLeft = 0;
801       int innerTop = 0;
802       drawLogFrame(pageTitle, left, top, right, bottom, centerX, innerLeft, innerTop);
803       innerLeft = innerLeftBase;
804       innerTop = innerTopBase;

```

```

802     setcolor(LIGHTGREEN);
803     int yPos = innerTop;
804     int count = 0;
805     while (index < wrappedLogs.size() && count < maxLinesPerPage) {
806         outtextxy(innerLeft, yPos, (char*)wrappedLogs[index].c_str());
807         yPos += lineStep;
808         index++;
809         count++;
810     }
811 }
812
813 getch();
814 }
815 }
816
817 void GUI::showMessage(const std::string& title, const std::vector<std::string>& lines) {
818     clearScreen();
819     setbkcolor(COLOR_BACKGROUND);
820     drawHeaderFrame(WINDOW_WIDTH);
821
822     int left, top, right, bottom, centerX, innerLeft, innerTop;
823     drawContentFrame(title, WINDOW_WIDTH, WINDOW_HEIGHT,
824                       left, top, right, bottom, centerX, innerLeft, innerTop);
825
826     int lineHeight = approxLineHeight();
827     int y = innerTop + 4;
828     int messageX = bottomIndentX(left);
829     setcolor(COLOR_TEXT);
830     for (const auto& line : lines) {
831         if (y > bottom - lineHeight * 2) break;
832         drawLeftAlignedText(messageX, y, line);
833         y += lineHeight;
834     }
835
836     setcolor(LIGHTCYAN);
837     drawCenteredText(centerX, bottom - lineHeight - 4, PRESS_ANY_KEY);
838     waitForKey();
839 }
840
841 void GUI::waitForKey() {
842     getch();
843 }
```

## 2.6 Mã nguồn chương trình vẽ khung : graphics.cpp

```

1 #include "../lib/graphics.h"
2 #include <cstdio>
3 #include <iostream>
4 #include <string>
5
6 #ifdef _WIN32
7 #include <windows.h>
8 #endif
9
10 static int g_graph_result = 0;
11 static bool g_ansi_enabled = false;
12 static int g_current_color = WHITE;
13
```

```

14 static const int g_virtual_width = 1200;
15 static const int g_virtual_height = 800;
16 static const int g_console_cols = 120;
17 static const int g_console_rows = 40;
18
19 static void setConsoleUtf8() {
20 #ifdef _WIN32
21     SetConsoleOutputCP(CP_UTF8);
22     SetConsoleCP(CP_UTF8);
23 #endif
24 }
25
26 static void enableAnsi() {
27 #ifdef _WIN32
28     HANDLE hOut = GetStdHandle(STD_OUTPUT_HANDLE);
29     if (hOut == INVALID_HANDLE_VALUE) {
30         g_ansi_enabled = false;
31         return;
32     }
33     DWORD mode = 0;
34     if (!GetConsoleMode(hOut, &mode)) {
35         g_ansi_enabled = false;
36         return;
37     }
38     mode |= ENABLE_VIRTUAL_TERMINAL_PROCESSING;
39     if (!SetConsoleMode(hOut, mode)) {
40         g_ansi_enabled = false;
41         return;
42     }
43 #endif
44     setConsoleUtf8();
45     g_ansi_enabled = true;
46 }
47
48 static int mapX(int x) {
49     if (x < 0) x = 0;
50     if (x > g_virtual_width) x = g_virtual_width;
51     int col = (x * g_console_cols) / g_virtual_width;
52     if (col < 1) col = 1;
53     if (col > g_console_cols) col = g_console_cols;
54     return col;
55 }
56
57 static int mapY(int y) {
58     if (y < 0) y = 0;
59     if (y > g_virtual_height) y = g_virtual_height;
60     int row = (y * g_console_rows) / g_virtual_height;
61     if (row < 1) row = 1;
62     if (row > g_console_rows) row = g_console_rows;
63     return row;
64 }
65
66 static int ansiColorCode(int color) {
67 // thiet lap mau thi ch doi mau gie thi doi o day.
68
69     switch (color) {
70         case BLACK: return 30;
71         case BLUE: return 34;
72         case GREEN: return 32;

```

```

73     case CYAN: return 36;
74     case RED: return 31;
75     case MAGENTA: return 35;
76     case BROWN: return 33;
77     case LIGHTGRAY: return 37;
78     case DARKGRAY: return 90;
79     case LIGHTBLUE: return 94;
80     case LIGHTGREEN: return 92;
81     case LIGHTCYAN: return 96;
82     case LIGHTRED: return 91;
83     case LIGHTMAGENTA: return 95;
84     case YELLOW: return 93;
85     case WHITE: return 97;
86     default: return 37;
87 }
88 }
89
90 static void moveCursor(int row, int col) {
91     if (!g_ansi_enabled) return;
92     std::cout << "\x1b[" << row << ";" << col << "H";
93 }
94
95 static void applyColor() {
96     if (!g_ansi_enabled) return;
97     std::cout << "\x1b[" << ansiColorCode(g_current_color) << "m";
98 }
99
100 static bool writeUtf8(const char* text) {
101     if (!text) return false;
102 #ifdef _WIN32
103     HANDLE hOut = GetStdHandle(STD_OUTPUT_HANDLE);
104     if (hOut == INVALID_HANDLE_VALUE) {
105         return false;
106     }
107     int wlen = MultiByteToWideChar(CP_UTF8, 0, text, -1, nullptr, 0);
108     if (wlen <= 0) {
109         return false;
110     }
111     std::wstring wstr(static_cast<size_t>(wlen), L'\0');
112     MultiByteToWideChar(CP_UTF8, 0, text, -1, &wstr[0], wlen);
113     DWORD written = 0;
114     BOOL ok = WriteConsoleW(hOut, wstr.c_str(), static_cast<DWORD>(wlen - 1), &written,
115                             nullptr);
116     return ok != FALSE;
117 #else
118     std::cout << text;
119     return true;
120 #endif
121 }
122 void initgraph(int *graphdriver, int *graphmode, char *pathtodriver) {
123     (*void)graphdriver; (*void)graphmode; (*void)pathtodriver;
124     g_graph_result = 0; // gr0k
125     enableAnsi();
126     cleardevice();
127 }
128 int graphresult() {
129     return g_graph_result;
130 }
```

```

131 }
132
133 void closegraph() {
134     if (g_ansi_enabled) {
135         std::cout << "\x1b[0m" << std::flush;
136     }
137 }
138
139 void cleardevice() {
140     if (g_ansi_enabled) {
141         std::cout << "\x1b[2J\x1b[H" << std::flush;
142     } else {
143         std::cout << std::string(50, '\n');
144     }
145 }
146
147 void setcolor(int color) {
148     g_current_color = color;
149 }
150
151 void setbkcolor(int color) { (void)color; }
152
153 void outtextxy(int x, int y, char *textstring) {
154     if (!textstring) return;
155     if (g_ansi_enabled) {
156         int row = mapY(y);
157         int col = mapX(x);
158         moveCursor(row, col);
159         applyColor();
160         if (!writeUtf8(textstring)) {
161             std::cout << textstring;
162         }
163         std::cout << std::flush;
164     } else {
165         if (!writeUtf8(textstring)) {
166             std::cout << textstring;
167         }
168         std::cout << std::endl;
169     }
170 }
171
172 void rectangle(int left, int top, int right, int bottom) {
173     if (!g_ansi_enabled) return;
174     int r1 = mapY(top);
175     int r2 = mapY(bottom);
176     int c1 = mapX(left);
177     int c2 = mapX(right);
178     if (r2 <= r1 || c2 <= c1) return;
179
180     applyColor();
181     // canh tren va duoi
182     moveCursor(r1, c1);
183     std::cout << "+" << std::string(c2 - c1 - 1, '=')
184     << "+";
185     moveCursor(r2, c1);
186     std::cout << "+" << std::string(c2 - c1 - 1, '-')
187     << "+";
188     // khung phia ngoai (hai ben)
189     for (int r = r1+1; r < r2; r++) {
190         moveCursor(r, c1);
191         std::cout << "|";
192     }
193 }
```

```

190         std::cout << "|";
191
192         moveCursor(r, c2);
193         std::cout << "|";
194     }
195     std::cout << std::flush;
196 }
197
198 int getch() {
199     int c = std::getchar();
200     return c;
201 }
```

### .3 Mã nguồn các thư viện tự cài đặt

#### .3.1 Mã nguồn thư viện dữ liệu đồ thị : graph.h

```

1 #pragma once
2 #include <string>
3 #include <vector>
4 #include "Global.h"
5
6 struct Edge {
7     int destination;
8     int weight;
9     Edge(int dest, int w) : destination(dest), weight(w) {}
10 };
11
12 class Graph {
13 private:
14     int V;
15     int E;
16     std::vector<std::vector<Edge>> adjList;
17     std::vector<std::string> vertexLabels;
18
19 public:
20     Graph();
21     explicit Graph(int vertices);
22     ~Graph();
23
24     int getVertexCount() const;
25     int getEdgeCount() const;
26     const std::vector<std::vector<Edge>>& getAdjacencyList() const;
27     std::string getVertexLabel(int vertex) const;
28
29     void clear();
30     void addVertex(const std::string& label);
31     void addEdge(int source, int destination, int weight);
32     bool hasEdge(int source, int destination) const;
33     void makeUndirected();
34
35     bool fileExists(const std::string& filename) const;
36     bool readFromFile(const std::string& filename, bool& needCreate);
37     bool saveToFile(const std::string& filename) const;
38     bool exportForPython(const std::string& filename) const;
39     bool exportForPython(const std::vector<int>& path) const;
40     bool exportWithPath(const std::string& filename, const std::vector<int>& path) const;
```

```

41     bool isValid() const;
42     bool hasNegativeWeights() const;
43 };

```

### .3.2 Mã nguồn thư viện thiết lập màu : Colors.h

```

1 #ifndef COLORS_H
2 #define COLORS_H
3
4 #ifdef COLOR_BACKGROUND
5 #undef COLOR_BACKGROUND
6 #endif
7
8
9
10 const int COLOR_BACKGROUND = 0; // BLACK
11 const int COLOR_TEXT = 15; // WHITE
12 const int COLOR_BUTTON = 1; // BLUE
13 const int COLOR_BUTTON_HOVER = 9; // LIGHTBLUE
14 const int COLOR_BUTTON_CLICKED = 11; // LIGHTCYAN
15 const int COLOR_VERTEX = 2; // GREEN
16 const int COLOR_EDGE = 15; // WHITE
17 const int COLOR_SHORTEST_PATH = 4; // RED
18 const int COLOR_NEGATIVE_CYCLE = 12; // LIGHTRED
19
20#endif

```

### .3.3 Mã nguồn thư viện toàn cục : Global.h

```

1 #pragma once
2 #include <string>
3
4 constexpr int WINDOW_WIDTH = 1200;
5 constexpr int WINDOW_HEIGHT = 800;
6
7 enum class AppState {
8     MENU,
9     INPUT,
10    RUNNING,
11    COMPARE,
12    VISUALIZE
13};
14
15 enum class AlgorithmType {
16     DIJKSTRA,
17     BELLMAN_FORD,
18     BOTH
19};
20
21 const std::string DATA_FOLDER = "../data";
22 const std::string TEMP_EXPORT_FILE = "../data/temp.txt";
23 const std::string DEFAULT_GRAPH_FILE = "../data/graph.txt";
24 const std::string DEFAULT_REPORT_FILE = "../data/report.txt";
25 const std::string DEFAULT_FONT_PATH = "assets/arial.ttf";

```

### .3.4 Mã nguồn thư viện đồ họa : GUI.h

```

1 #ifndef GUI_H
2 #define GUI_H
3
4 #include "graphics.h"
5 #include <vector>
6 #include <string>
7 #include <tuple>
8 #include "Global.h"
9 #include "Colors.h"
10
11 class GUI {
12 private:
13     int screenWidth;
14     int screenHeight;
15
16 public:
17     GUI();
18     ~GUI();
19
20     void drawMenu();
21     void drawComparisonScreen(const std::vector<std::string>& logs);
22     void clearScreen();
23
24     int promptMenuChoice();
25     int promptChoice(const std::string& title, const std::vector<std::string>& options,
26                      const std::string& prompt, int minValue, int maxValue);
27     int promptInt(const std::string& title, const std::string& prompt, int minValue, int
28                   maxValue);
29     std::string promptLine(const std::string& title, const std::string& prompt,
30                           const std::string& defaultValue = "");
31     bool promptYesNo(const std::string& title, const std::string& prompt);
32     void promptGraphInput(bool isDirected, int& numVertices, int& numEdges,
33                           std::vector<std::tuple<int, int, int>>& edges);
34     void promptStartEnd(const std::string& title, int minValue, int maxValue,
35                          int& startValue, int& endValue);
36     void showAlgorithmLogs(const std::string& title, const std::vector<std::string>& logs
37 );
38     void showMessage(const std::string& title, const std::vector<std::string>& lines);
39     void waitForKey();
40 };
#endif

```

### .3.5 Mã nguồn thư viện thuật toán : algorithms.h

```

1 #ifndef ALGORITHMS_H
2 #define ALGORITHMS_H
3
4 #include <vector>
5 #include <string>
6 #include <utility>
7 #include "Graph.h"
8
9
10 struct PathResult {
11     bool success;

```

```

12     int startVertex;
13     std::vector<int> distances;
14     std::vector<int> previousVertex;
15     std::vector<int> shortestPath;
16     std::vector<std::string> logs;
17     bool hasNegativeCycle;
18
19     PathResult() : success(false), startVertex(-1), hasNegativeCycle(false) {}
20 };
21
22 class Algorithms {
23 private:
24     const Graph& graph;
25
26     void logStep(std::vector<std::string>& logs, const std::string& message);
27     std::vector<int> reconstructPath(int destination, const std::vector<int>&
28                                     previousVertex) const;
29
30 public:
31     explicit Algorithms(const Graph& g);
32
33     PathResult dijkstra(int start, bool showSteps = false);
34
35     PathResult bellmanFord(int start, bool showSteps = false);
36
37     std::vector<int> getShortestPath(const PathResult& result, int destination) const;
38
39     int getDistance(const PathResult& result, int destination) const;
40 };
41 #endif

```

### 3.6 Mã nguồn thư viện so sánh : comparison.h

```

1 #ifndef COMPARISON_H
2 #define COMPARISON_H
3
4 #include <string>
5 #include <vector>
6 #include <chrono>
7 #include "Algorithms.h"
8 #include "Graph.h"
9
10 struct PerformanceMetrics {
11     std::string algorithmName;
12     long long executionTimeUs;           // tinh bang micro giay
13     long long memoryUsageBytes;          // byte
14     int distancesCalculated;
15     double complexity;                  // do phuc tap
16     bool success;
17
18     PerformanceMetrics() : algorithmName(""), executionTimeUs(0),
19                           memoryUsageBytes(0), distancesCalculated(0),
20                           complexity(0.0), success(false) {}
21 };
22
23 struct ComparisonReport {
24

```

```
25     int startVertex;
26     int V; // so dinh
27     int E; // canh
28     std::vector<PerformanceMetrics> metrics;
29     std::vector<std::string> logs;
30
31     ComparisonReport() : startVertex(-1), V(0), E(0) {}
32 };
33
34 class Comparison {
35 private:
36     const Graph& graph;
37     Algorithms algorithms;
38
39 public:
40     explicit Comparison(const Graph& g);
41
42     ComparisonReport comparePerformance(int startVertex, AlgorithmType type =
43                                         AlgorithmType::BOTH);
44
45     PerformanceMetrics measureAlgorithm(int startVertex, AlgorithmType type);
46 };
47 #endif
```