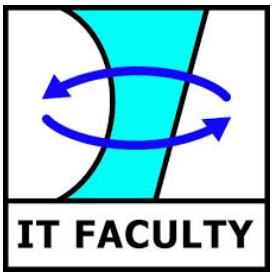
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA CÔNG NGHỆ THÔNG TIN



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**

Đà Nẵng, 06/2026

Mục lục

Danh sách hình vẽ

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.

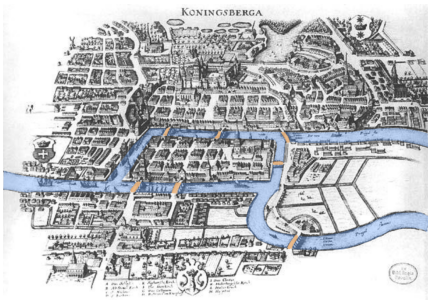
Do thời gian và trình độ chuyên môn còn hạn chế, chúng em nhận thức rằng đồ án không tránh khỏi những thiếu sót . Chúng em xin chân thành cảm ơn sự hướng dẫn tận tâm của thầy **TS. Nguyễn Văn Hiệu** trong suốt quá trình thực hiện đề tài. Nhóm chúng em rất mong nhận được những ý kiến đóng góp quý báu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



Hình 1.1: Hình minh họa lý thuyết đồ thị

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 nới lỏng 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 (nới 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 (nới 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 (nới 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 nới lỏng toàn bộ các cạnh $|V| - 1$ lần. Mỗi lần nới 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 nới 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 **nới 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 nới 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 thưa 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 $v_0, v_1, \ldots, v_k$ 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 $v_0 \to v_1 \to \cdots \to v_0$. 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.

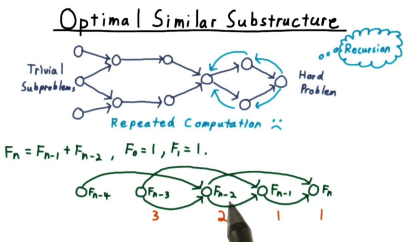
2.2.2 Nguyên lý tối ưu và tính đúng đắn

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 ∞ 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 **nới 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(nới 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:

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

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 ∞ biểu thị đỉnh không thể tới được.
  - (Tùy chọn) Mảng đường đi trước `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
`iostream`	Nhập/xuất console.
`string`	Xử lý chuỗi.
`vector`	Mảng động.
`tuple`	Lưu cạnh (u, v, w).
`limits`	Giá trị `INF`.
`chrono`	Đo thời gian chạy thuật toán.
`filesystem`	Kiểm tra/tạo thư mục, kiểm tra file tồn tại.
`fstream`	Đọc/ghi file.
`algorithm`	Các thao tác tìm kiếm/xử lý trong `vector`.
`queue`	`priority_queue` cho Dijkstra.
`sstream`	Ghép chuỗi/định dạng văn bản (format).
`iomanip`	Căn lề/định dạng khi in bảng.
`cmath`	`log` và các phép tính phục vụ phân tích độ phức tạp.
`cctype`	Kiểm tra ký tự nhập.
`cstdio`	Dùng `std::getchar()` 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 lí.

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).

```
struct Edge {
    int destination;
    int weight;
    Edge(int dest, int w) : destination(dest), weight(w) {}
};
```

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.

```
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`

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

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 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.

```
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`

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

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`

```cpp
struct PerformanceMetrics {
    std::string algorithmName;
    long long executionTimeUs;      // tính bằng micro giây
    long long memoryUsageBytes;     // byte
    int distancesCalculated;
    double complexity;              // độ phức tạp
    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`

```cpp
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`

- `std::vector<std::string> logs` – log kết quả để hiển thị.

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.

Giữ const Graph& graph và Algorithms algorithms.

```cpp
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.

```cpp
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 promptChoice(const std::string& title, const std::vector<std::string>& options, const std::string& prompt, int minValue, int maxValue);
    int promptInt(const std::string& title, const std::string& prompt, int minValue, int maxValue);
    std::string promptLine(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 isDirected, int& numVertices, int& numEdges, std::vector<std::tuple<int, int, int>>& edges);
    void promptStartEnd(const std::string& title, int minValue, int maxValue, int& startValue, int& endValue);
    void showAlgorithmLogs(const std::string& title, const std::vector<std::string>& logs);
    void showMessage(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 nới 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.
- **Nới 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 nới 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 `dist[]` với giá trị ∞ cho tất cả các đỉnh; đặt $\texttt{dist}[s] \leftarrow 0$ cho đỉnh nguồn s. Khởi tạo mảng `pred[]` để lưu đường đi trước, gán $\texttt{pred}[v] \leftarrow$ `undefined`.
2. Khởi tạo hàng đợi ưu tiên Q chứa cặp $(\texttt{dist}[v], v)$ 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. nới lỏng các cạnh kề: với mỗi đỉnh kề v của u có trọng số $w(u, v)$, nếu $\texttt{dist}[u] + w(u, v) < \texttt{dist}[v]$, cập nhật $\texttt{dist}[v] \leftarrow \texttt{dist}[u] + w(u, v)$ và $\texttt{pred}[v] \leftarrow u$; đồng thời thêm $(\texttt{dist}[v], v)$ vào hàng đợi.
5. Lặp lại cho tới khi hàng đợi trống. Kết quả `dist` chứa khoảng cách ngắn nhất từ s đến mọi đỉnh; `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

Độ 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 lỏng 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

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

  nếu xét đồ thị liên thông và thưa thì thường viết gọn là $O(|E| \log |V|)$.
- **Dùng heap Fibonacci:** cải thiện thao tác *decrease-key* để đạt độ phức tạp

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

Nhiều nghiên cứu thực nghiệm cho thấy trên các đồ thị thưa, 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 đồ thị.

Ứng dụng và ghi chú

Thuật toán Dijkstra có nhiều ứng dụng:

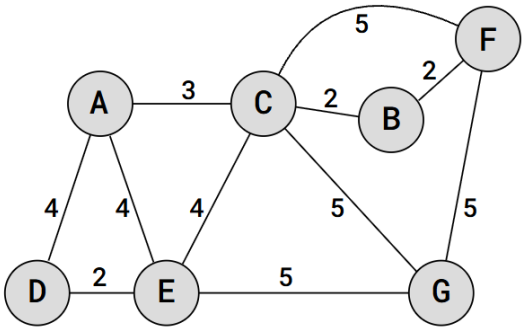
- **Định tuyến mạng:** giao thức OSPF và IS–IS sử dụng Dijkstra để tính toán đường dẫn tối ưu cho gói tin.
- **Hệ thống GPS và chỉ đường:** tìm đường đi ngắn nhất trên bản đồ có trọng số là thời gian hoặc khoảng cách.

- **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 lộ trình 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 $A \to G$. Ta bắt đầu từ đỉnh nguồn D và tìm đường đi ngắn nhất đến các đỉnh còn lại.

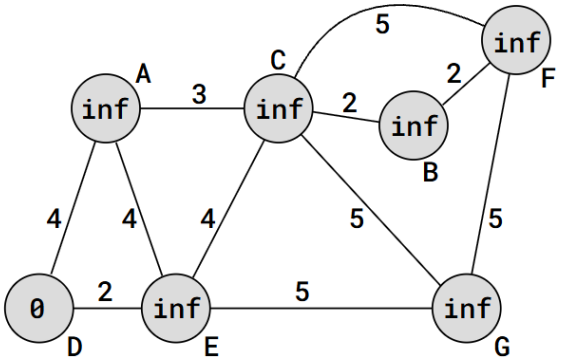


Hình 3.9: Đồ thị minh họa thuật toán Dijkstra

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.

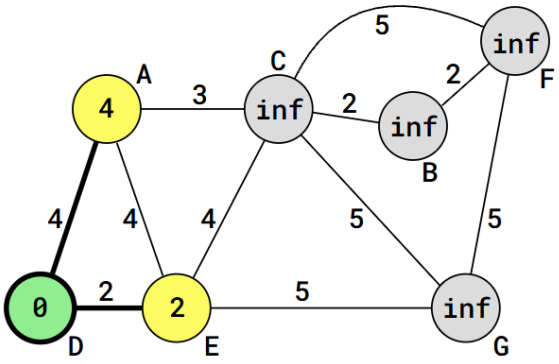
Vì ban đầu khoảng cách từ nguồn đến A và E đều là vô hạn (chưa tìm thấy đường đi), nên khi xét các cạnh từ D:

- Với cạnh $D \to A$ 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 $D \to E$ 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* (nới 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 *nới 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:

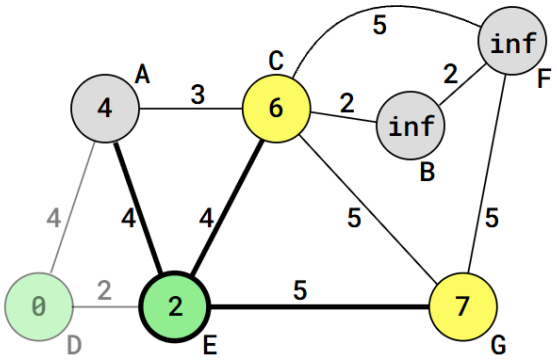
– Trong tất cả các đỉnh chưa thăm, chọn đỉnh có khoảng cách tạm thời dist[·] nhỏ nhất tính từ đỉnh nguồn (ở đây là D).

Ở bước này, sau khi cập nhật:

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

các đỉnh khác vẫn là ∞.

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:

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

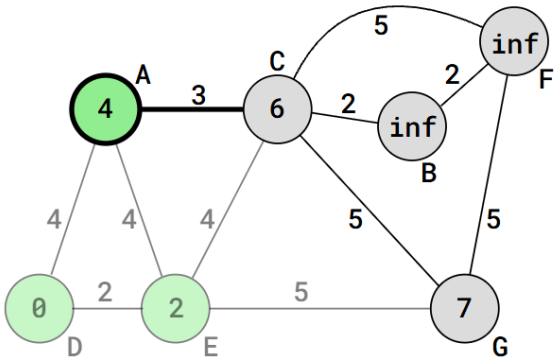
sau đó so sánh với $\text{dist}[v]$ hiện tại.

- **Đỉnh A:** nếu đi theo $D \to E \to A$ 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 $D \to E \to C$ cho
$$\text{alt} = 2 + 4 = 6.$$
Vì $\text{dist}[C]$ đang là ∞, nên cập nhật $\text{dist}[C] = 6$.

- **Đỉnh G:** đường $D \to E \to G$ cho
$$\text{alt} = 2 + 5 = 7.$$
Vì $\text{dist}[G]$ đang là ∞, 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 dist[·] 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.

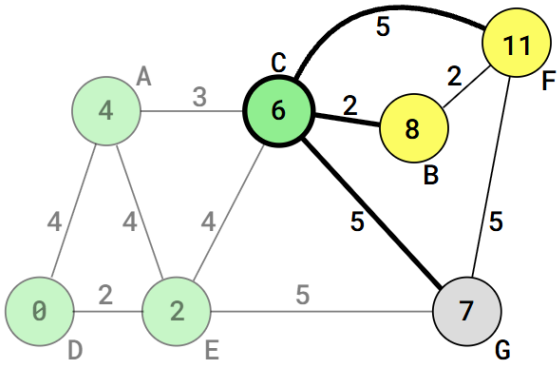
- **Với đỉnh C:** nếu đi theo đường $D \to A \to C$ thì khoảng cách tạm thời mới là

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

  Nhưng trước đó ta đã có $\text{dist}[C] = 6$ (từ đường $D \to E \to C$), 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 dist[·] 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 $D \to \cdots \to C \to F$ là

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

  nên cập nhật $\text{dist}[F] = 11$.

- **Đỉnh B:** khoảng cách mới theo đường $D \to \cdots \to C \to B$ là

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

nên cập nhật $\text{dist}[B] = 8$.

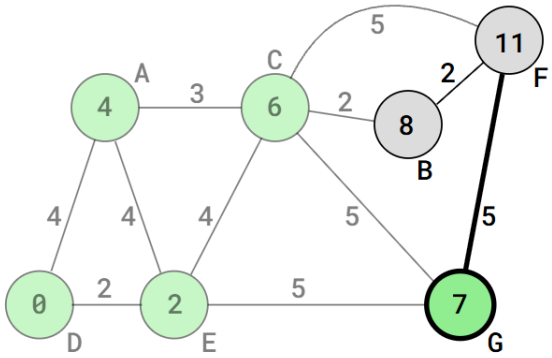
- **Đỉnh G:** nếu đi qua C thì khoảng cách là

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

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 dist[·] 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.

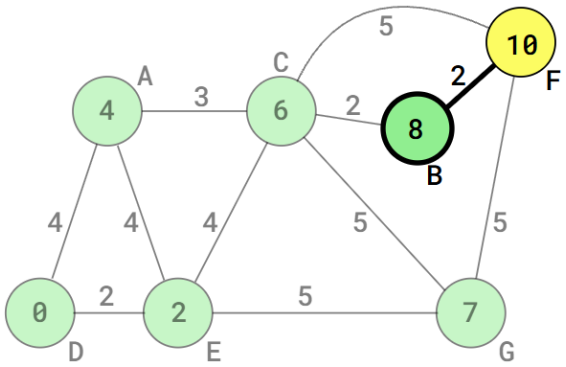
- **Đỉnh F:** hiện đã có $\text{dist}[F] = 11$. Nếu đi theo đường $D \to \cdots \to G \to F$ thì khoảng cách mới là

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

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 dist[·] 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

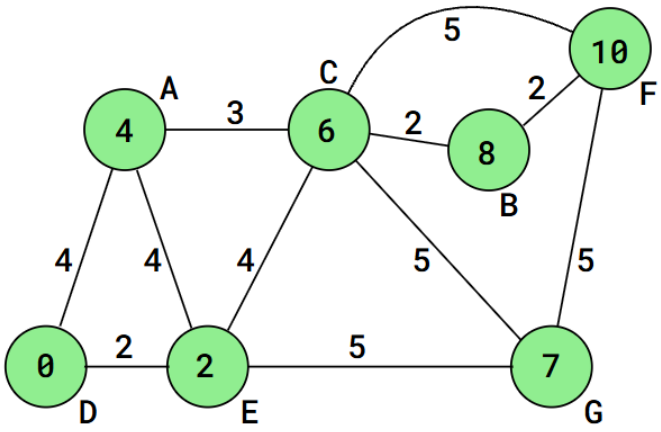
Khi B được chọn làm đỉnh hiện tại (vì $\text{dist}[B] = 8$), thuật toán sẽ xét các đỉnh kề của B mà chưa được thăm và thử cập nhật khoảng cách đến chúng bằng cách đi qua B.

- **Đỉnh F:** khoảng cách mới theo đường $D \to \cdots \to B \to F$ 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 tới 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 ∞.
- 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. **Nới lỏng 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 nới lỏng toàn bộ các cạnh cho mỗi lần lặp.
- **Trường hợp tốt nhất:** khi không cần nới lỏng 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 nới lỏng 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 nới lỏng, thuật toán dừng. Cách làm này dựa trên quan sát rằng phần lớn thao tác nới lỏng 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 (`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 $\texttt{dist}[s] = 0$ cho nguồn s, các đỉnh khác bằng ∞; đặt $\texttt{inQueue}[s] = \texttt{true}$; đưa s vào hàng đợi vòng.
2. **Lặp cho tới khi hàng đợi rỗng:**
   - Lấy đỉnh u ra khỏi đầu hàng đợi, đặt $\texttt{inQueue}[u] = \texttt{false}$.
   - Duyệt tất cả các cạnh (u, v, w). Nếu $\texttt{dist}[u] + w < \texttt{dist}[v]$ thì cập nhật $\texttt{dist}[v]$ và, nếu v chưa có trong hàng đợi, đưa v vào cuối (hoặc đầu nếu dùng kỹ thuật *Small Label First*) và đặt $\texttt{inQueue}[v] = \texttt{true}$.
   - Tăng $\texttt{cnt}[v]$ và nếu $\texttt{cnt}[v] > |V|$ 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 nới lỏng 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 **nới lỏng 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 nới lỏng 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.

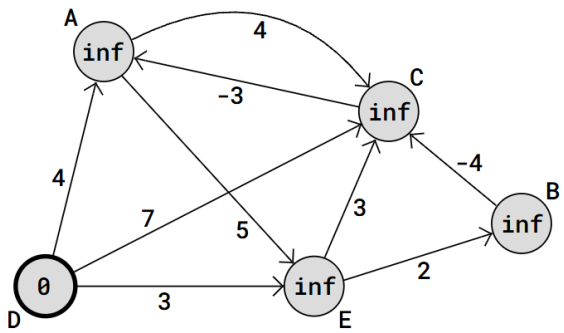
Thuật toán Bellman-Ford là thuật toán tìm đường đi ngắn nhất từ một điểm xuất phát duy nhất . Nó hoạt động hiệu quả ngay cả trong trường hợp có cạnh âm và cũng có khả năng phát hiện chu trình âm. Thuật toán này hoạt động dựa trên nguyên tắc nới lỏng các cạnh.

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 tới nhiều yếu tố như kẹt xe và điều kiện đường sá, nơi trọng số (thời gian/chi phí) có thể thay đổi và đôi 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 kí hiệu bằng các chữ cái in hoa từ $A \to E$. Với đỉnh bắt đầu là D.

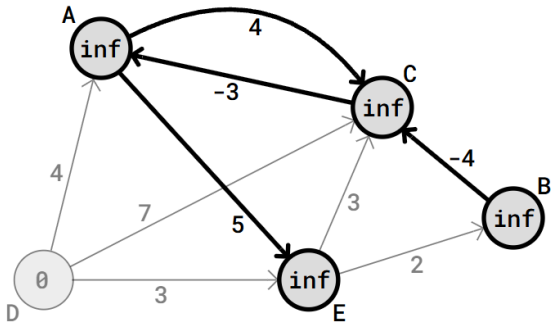


Hình 3.18: Hình minh hóa thuật toán Bellman-Ford

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 $A \to C$, $A \to E$, $B \to C$ **và** $C \to A$. 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 hóa thuật toán Bellman-Ford

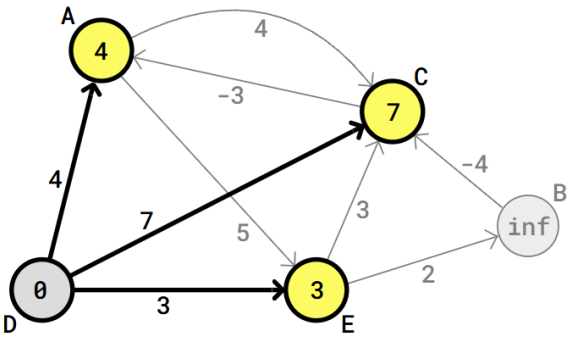
Sau khi đã thử *relax* các cạnh xuất phát từ A, B, C mà vẫn không có cập nhật (vì các đỉnh này đang có dist $= \infty$), thuật toán chuyển sang kiểm tra các cạnh đi ra từ đỉnh D.

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

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

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

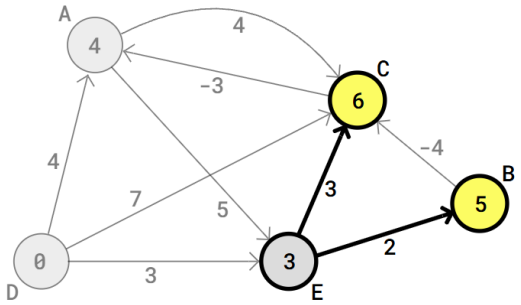
- $D \to A$ (trọng số 4) $\Rightarrow$ $\text{dist}[A] = 4$.
- $D \to E$ (trọng số 3) $\Rightarrow$ $\text{dist}[E] = 3$.
- $D \to C$ (trọng số 7) $\Rightarrow$ $\text{dist}[C] = 7$.



Hình 3.20: Hình minh hóa 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 $E \to B$ (trọng số 2):**
$$\text{dist}[B] \leftarrow \min\big(\text{dist}[B], \text{dist}[E] + 2\big) = \min(\infty, 3 + 2) = 5.$$
$\Rightarrow$ cập nhật trọng số dist[B] = 5.
- **Với cạnh $E \to C$ (trọng số 3):**
$$\text{dist}[C] \leftarrow \min\big(\text{dist}[C], \text{dist}[E] + 3\big) = \min(7, 3 + 3) = 6.$$
$\Rightarrow$ cập nhật trọng số dist[C] = 6



Hình 3.21: Hình minh 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.

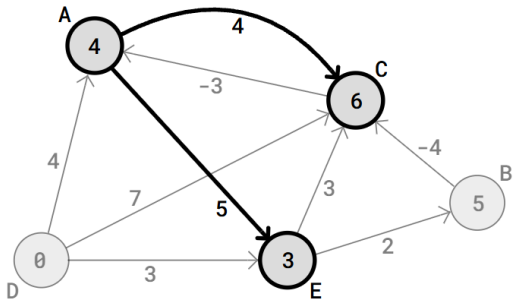
Bước sang lượt kiểm tra thứ hai, thuật toán bắt đầu lại từ đầu và xét các cạnh đi ra từ đỉnh A. Tại thời điểm này ta đang có:

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

Khi *relax* các cạnh từ A:

- **Với cạnh $A \to C$ (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 $A \to E$ (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 $A \to C$ và $A \to E$ 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.



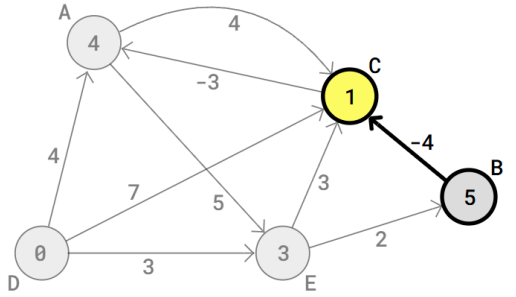
Hình 3.22: Hình minh hóa thuật toán Bellman-Ford

Cạnh tiếp theo được xét là $B \to C$. 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:

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



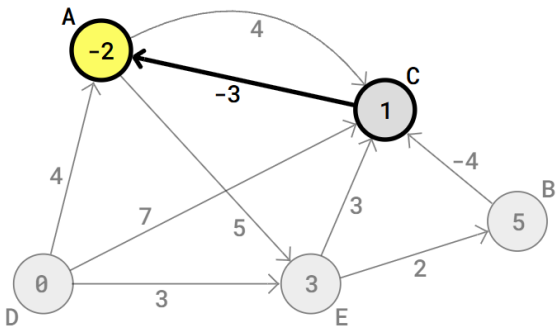
Hình 3.23: Hình minh hóa thuật toán Bellman-Ford

Khi xét cạnh $C \to A$ 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 hóa thuật toán Bellman-Ford

Trong vòng lặp thứ 2, cạnh $C \to A$ đúng là "cú cập nhật cuối cùng" đối với đồ thị này: sau khi dist$[C]$ giảm xuống 1, việc *relax* $C \to A$ với trọng số -3 làm 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 thoạt 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

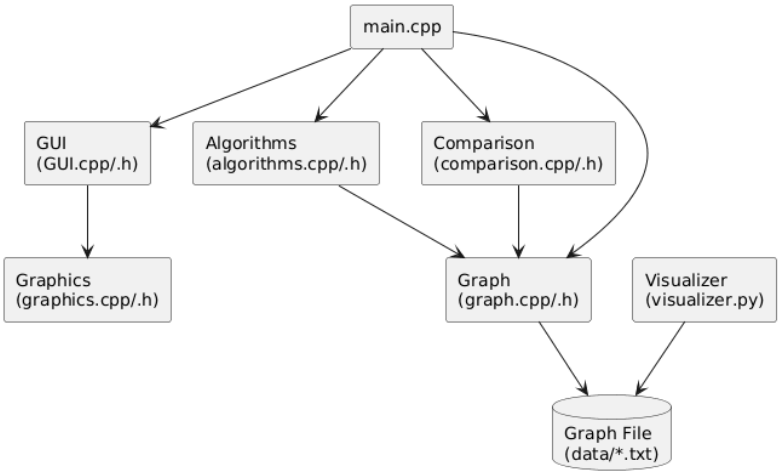
Bảng 3.1: So sánh Dijkstra và Bellman–Ford

Tiêu chí	**Dijkstra**	**Bellman–Ford**
Loại bài toán	SSSP	SSSP
Điều kiện trọng số	$w(e) \geq 0$	Có thể âm; phát hiện chu trình âm
Ý tưởng chính	Tham lam + hàng đợi ưu tiên	Quy hoạch động + nới lỏng toàn bộ cạnh
Độ phức tạp thời gian	$O\big((\lvert V\rvert + \lvert E\rvert)\log \lvert V\rvert\big)$	$O(\lvert V\rvert\lvert E\rvert)$
Bộ nhớ (bổ sung)	$O(\lvert V\rvert)$	$O(\lvert V\rvert)$
Đồ thị thưa / lớn	Rất phù hợp	Thường chậm
Trọng số âm	Không dùng được	Dùng được
Chu trình âm	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 $O\big((V+E)\log V\big)$. 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
`Graph::Graph()`	`Graph.h`, `graph.cpp`	Khởi tạo đồ thị rỗng, tạo thư mục `DATA_FOLDER` nếu chưa có.
`Graph::Graph(int vertices)`	`Graph.h`, `graph.cpp`	Khởi tạo đồ thị với số đỉnh ban đầu, đặt nhãn `V0..V(n-1)`.
`Graph:: Graph()`	`Graph.h`, `graph.cpp`	Giải phóng dữ liệu bằng `clear()`.
`Graph::getVertexCount()`	`Graph.h`, `graph.cpp`	Trả về số đỉnh `V`.
`Graph::getEdgeCount()`	`Graph.h`, `graph.cpp`	Trả về số cạnh `E`.
`Graph::getAdjacencyList()`	`Graph.h`, `graph.cpp`	Trả về danh sách kề (const).
`Graph::getVertexLabel(int)`	`Graph.h`, `graph.cpp`	Trả nhãn đỉnh nếu hợp lệ, ngược lại `"Invalid"`.

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`	Đo 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/đóng 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 thị.
`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

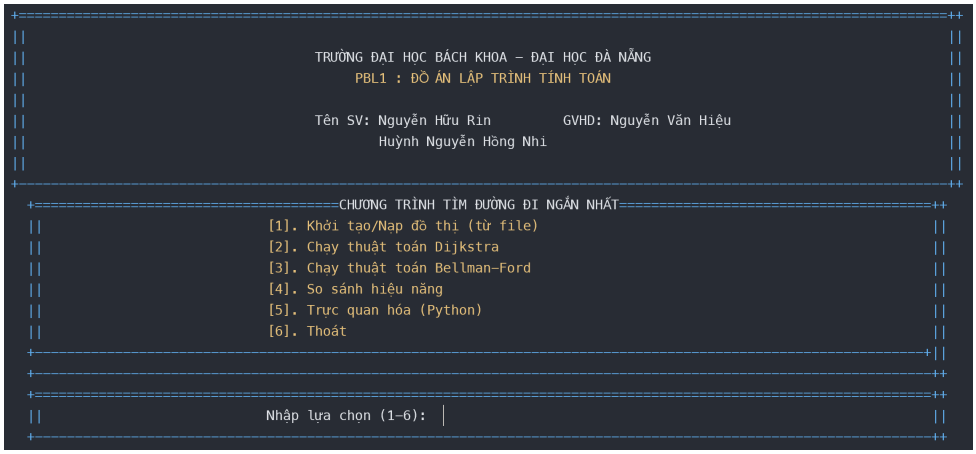
Bài toán được triển khai chủ yếu bằng ngôn ngữ **C++** để đảm bảo hiệu năng cao khi thao tác trên đồ thị lớn. Ngoài thư viện chuẩn, chương trình còn sử dụng một số thư viện bổ trợ:

- `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

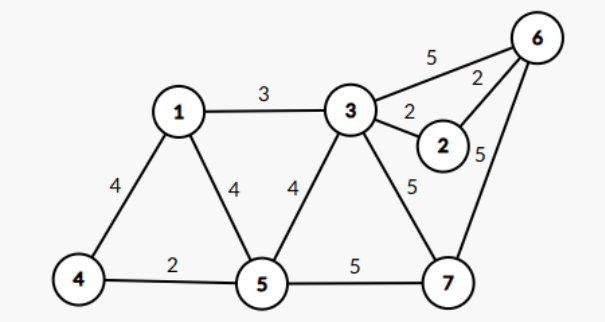
Ví dụ: Cho đồ thị vô hướng, không âm với:

- 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 cạnh

Biểu diễn đồ thị

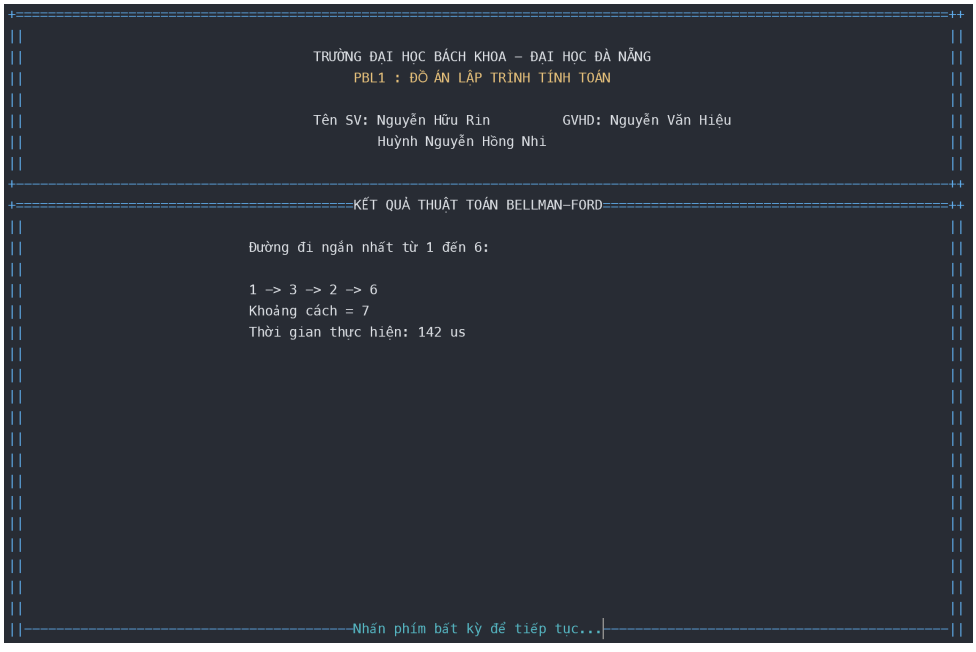


Hình 4.5: Biểu diễn đồ thị

Chức năng Thuật toán Dijkstra

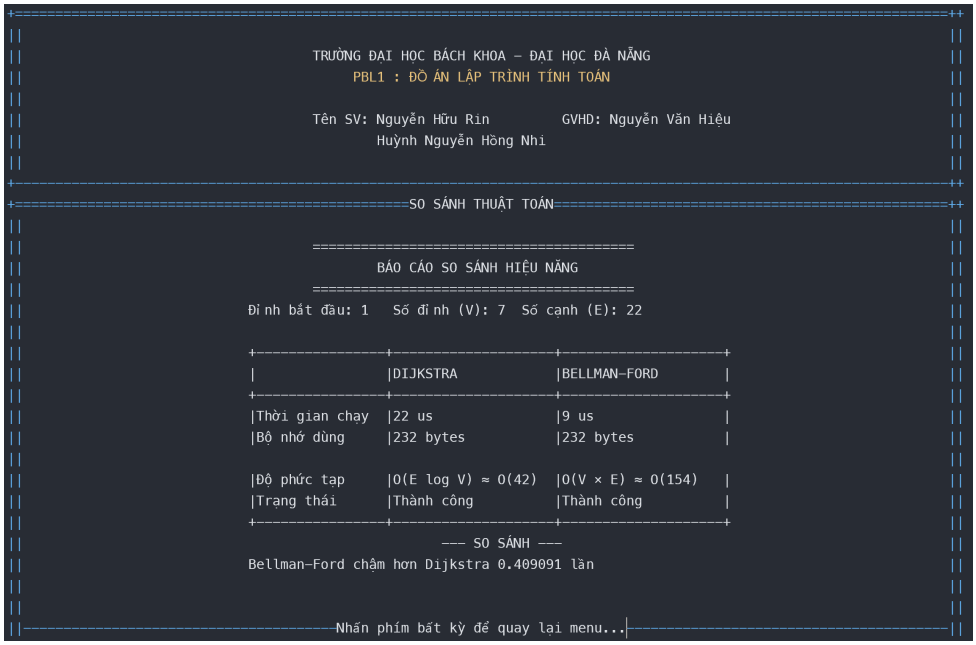
Hình 4.6: Kết quả thuật toán Dijkstra

Chức năng Thuật toán Bellman-Ford



Hình 4.7: Kết quả thuật toán Bellman-Ford

Chức năng So sánh



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

```cpp
#include <iostream>
#include <limits>
#include <string>
#include <vector>
#include <chrono>
#ifdef _WIN32
#include <windows.h>
#include <cwchar>
#endif
#include "../lib/graphics.h"
#include "../lib/Global.h"
#include "../lib/GUI.h"
#include "../lib/graph.h"
#include "../lib/Algorithms.h"
#include "../lib/Comparison.h"

// ==================== GLOBAL VARIABLES ====================
Graph graph;
Algorithms* algorithms = nullptr;
Comparison* comparison = nullptr;
GUI* gui = nullptr;

PathResult lastResult;

#ifdef _WIN32
static void initConsoleUtf8() {
    SetConsoleOutputCP(CP_UTF8);
    SetConsoleCP(CP_UTF8);
    HANDLE hOut = GetStdHandle(STD_OUTPUT_HANDLE);
    if (hOut != INVALID_HANDLE_VALUE) {
        CONSOLE_FONT_INFOEX cfi;
        cfi.cbSize = sizeof(cfi);
        if (GetCurrentConsoleFontEx(hOut, FALSE, &cfi)) {
            cfi.FontFamily = FF_DONTCARE;
            cfi.FontWeight = FW_NORMAL;
            std::wcsncpy(cfi.FaceName, L"Consolas", LF_FACESIZE - 1);
            cfi.FaceName[LF_FACESIZE - 1] = L'\0';
            SetCurrentConsoleFontEx(hOut, FALSE, &cfi);
        }
    }
}
#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, "Nhap lua chon:", 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, "Nhap lua chon:", 1, 3);
83 
84     if (choice == 1) {
85         bool isDirected = promptGraphDirected();
86         std::string filename = gui->promptLine("TAI DO THI",
87             "Nhap duong dan file (mac dinh: " + 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", "Khong tim thay file. Tao
    file moi?");
94                 if (create) {
95                     createGraphFromGui(isDirected);
96                     lastResult = PathResult();
97                     if (graph.saveToFile(filename)) {
98                         gui->showMessage("TAI DO THI", {"Da luu do thi vao: " + filename
    });
99                     } else {
100                        gui->showMessage("TAI DO THI", {"Luu do thi that bai."});
```

```
                    }
                }
            } else {
                gui->showMessage("TAI DO THI", {"Tai file that bai."});
            }
        } else {
            if (!isDirected) {
                graph.makeUndirected();
            }
            lastResult = PathResult();
            gui->showMessage("TAI DO THI", {
                "Da tai do thi: V = " + std::to_string(graph.getVertexCount()) +
                ", E = " + std::to_string(graph.getEdgeCount()) + "."
            });
        }
    } else if (choice == 2) {
        bool isDirected = promptGraphDirected();
        createGraphFromGui(isDirected);

        lastResult = PathResult();
        bool save = gui->promptYesNo("TAO DO THI", "Luu do thi ra file?");
        if (save) {
            std::string filename = gui->promptLine(
                "TAO DO THI",
                "Duong dan file (mac dinh: " + DEFAULT_GRAPH_FILE + "): ",
                DEFAULT_GRAPH_FILE
            );
            if (graph.saveToFile(filename)) {
                gui->showMessage("TAO DO THI", {"Da luu do thi vao " + filename});
            } else {
                gui->showMessage("TAO DO THI", {"Luu do thi that bai."});
            }
        }
    }
}

static void runAlgorithm(AlgorithmType type) {
    if (!graph.isValid()) {
        gui->showMessage("THONG BAO", {"Do thi rong. Vui long tao hoac tai do thi truoc."
    });
        return;
    }

    if (type == AlgorithmType::DIJKSTRA && graph.hasNegativeWeights()) {
        gui->showMessage("THONG BAO", {"Dijkstra khong chay duoc voi canh co trong so am.
    "});
        return;
    }

    int V = graph.getVertexCount();
    int startInput = 1;
    int endInput = 1;
    gui->promptStartEnd("CHON DINH", 1, V, startInput, endInput);
    int start = startInput - 1;
    int end = endInput - 1;

    PathResult result;
    auto startTime = std::chrono::high_resolution_clock::now();
    if (type == AlgorithmType::DIJKSTRA) {
```

```
        result = algorithms->dijkstra(start, true);
    } else {
        result = algorithms->bellmanFord(start, true);
    }
    auto endTime = std::chrono::high_resolution_clock::now();
    auto execUs = std::chrono::duration_cast<std::chrono::microseconds>(endTime -
    startTime).count();

    if (!result.success) {
        std::vector<std::string> lines = {"Thuat toan that bai."};
        if (result.hasNegativeCycle) {
            lines.push_back("Phat hien chu trinh am.");
        }
        gui->showMessage("KET QUA", lines);
        return;
    }

    result.shortestPath = algorithms->getShortestPath(result, end);
    lastResult = result;
    std::string algoTitle = (type == AlgorithmType::DIJKSTRA)
        ? "DIEN BIEN THUAT TOAN DIJKSTRA"
        : "DIEN BIEN THUAT TOAN BELLMAN-FORD";

    std::vector<std::string> displayLogs;
    std::string startLabel = graph.getVertexLabel(start);
    std::string endLabel = graph.getVertexLabel(end);
    int dist = algorithms->getDistance(result, end);
    if (dist >= 0 && dist != INT_MAX) {
        displayLogs.push_back("Duong di ngan nhat tu dinh " + startLabel + " den dinh " +
     endLabel + " la " + std::to_string(dist));
    }
    // else {
    //     displayLogs.push_back("Khong ton tai duong di tu " + startLabel + " den " +
    endLabel);
    // }

    if (result.shortestPath.empty()) {
        displayLogs.push_back("Khong ton tai duong di tu " + startLabel + " den " +
    endLabel);
    } else {
        std::string pathLine = "Duong di: ";
        for (size_t i = 0; i < result.shortestPath.size(); i++) {
            if (i > 0) pathLine += " -> ";
            pathLine += graph.getVertexLabel(result.shortestPath[i]);
        }
        displayLogs.push_back(pathLine);
    }
    displayLogs.push_back("Thoi gian thuc hien: " + std::to_string(execUs) + " us");
    displayLogs.push_back("");
    displayLogs.insert(displayLogs.end(), result.logs.begin(), result.logs.end());

    gui->showAlgorithmLogs(algoTitle, displayLogs);

    std::vector<std::string> lines;
    lines.push_back("Duong di ngan nhat tu " + graph.getVertexLabel(start) +
                    " den " + graph.getVertexLabel(end) + ":");
    if (result.shortestPath.empty()) {
        lines.push_back("Khong tim thay duong di.");
    }
```

```
    else {
        lines.push_back("");
        std::string pathLine;
        for (size_t i = 0; i < result.shortestPath.size(); i++) {
            if (!pathLine.empty()) pathLine += " -> ";
            pathLine += graph.getVertexLabel(result.shortestPath[i]);
        }
        lines.push_back(pathLine);
        int dist = algorithms->getDistance(result, end);
        lines.push_back("Khoang cach = " + std::to_string(dist));
        lines.push_back("Thoi gian thuc hien: " + std::to_string(execUs) + " us");
    }
    if(type == AlgorithmType::DIJKSTRA){
        gui->showMessage("KET QUA THUAT TOAN DIJKSTRA", lines);
    }
    else {
        gui->showMessage("KET QUA THUAT TOAN BELLMAN-FORD", lines);
    }
    // gui->showMessage("KET QUA", lines);
}

static void compareAlgorithms() {
    if (!graph.isValid()) {
        gui->showMessage("THONG BAO", {"Do thi rong. Vui long tao hoac tai do thi truoc."
    });
        return;
    }

    int V = graph.getVertexCount();
    int startInput = gui->promptInt("CHON DINH",
        "Dinh bat dau (1.." + std::to_string(V) + "): ", 1, V);
    int start = startInput - 1;

    auto report = comparison->comparePerformance(start, AlgorithmType::BOTH);
    gui->drawComparisonScreen(report.logs);
    gui->waitForKey();
}

static void exportToPython() {
    if (!graph.isValid()) {
        gui->showMessage("THONG BAO", {"Do thi rong. Vui long tao hoac tai do thi truoc."
    });
        return;
    }

    bool ok = false;
    if (!lastResult.shortestPath.empty()) {
        ok = graph.exportForPython(lastResult.shortestPath);
    } else {
        ok = graph.exportForPython(TEMP_EXPORT_FILE);
    }

    if (ok) {
        gui->showMessage("XUAT PYTHON", {
            "Da xuat ra " + TEMP_EXPORT_FILE,
            "Chay: python visualizer.py"
        });
    } else {
        gui->showMessage("XUAT PYTHON", {"Xuat that bai."});
```

```
    }
}

// ==================== MAIN FUNCTION ====================
int main() {
#ifdef _WIN32
    initConsoleUtf8();
#endif
    int gd = DETECT, gm;
    char bgiPath[] = "";
    initgraph(&gd, &gm, bgiPath);

    if (graphresult() != grOk) {
        std::cerr << "Khoi tao do hoa that bai. Chuyen sang che do console.\n";
    }

    algorithms = new Algorithms(graph);
    comparison = new Comparison(graph);
    gui = new GUI();

    bool running = true;
    while (running) {
        int choice = gui->promptMenuChoice();

        switch (choice) {
            case 1:
                handleGraphInput();
                break;
            case 2:
                runAlgorithm(AlgorithmType::DIJKSTRA);
                break;
            case 3:
                runAlgorithm(AlgorithmType::BELLMAN_FORD);
                break;
            case 4:
                compareAlgorithms();
                break;
            case 5:
                exportToPython();
                break;
            case 6:
                running = false;
                break;
            default:
                break;
        }
    }

    closegraph();
    delete algorithms;
    delete comparison;
    delete gui;

    return 0;
}
```

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

```cpp
#include "../lib/Graph.h"
#include <filesystem>
#include <fstream>

// ==================== CONSTRUCTOR & DESTRUCTOR ====================
Graph::Graph() : V(0), E(0) {
    std::filesystem::create_directories(DATA_FOLDER);
}

Graph::Graph(int vertices) : V(0), E(0) {
    std::filesystem::create_directories(DATA_FOLDER);
    for (int i = 0; i < vertices; i++) {
        addVertex("V" + std::to_string(i));
    }
}

Graph::~Graph() {
    clear();
}

// ==================== GETTERS ====================
int Graph::getVertexCount() const {
    return V;
}

int Graph::getEdgeCount() const {
    return E;
}

const std::vector<std::vector<Edge>>& Graph::getAdjacencyList() const {
    return adjList;
}

std::string Graph::getVertexLabel(int vertex) const {
    if (vertex >= 0 && vertex < V) {
        return vertexLabels[vertex];
    }
    return "Invalid";
}

// ==================== GRAPH MODIFICATION ====================
void Graph::clear() {
    adjList.clear();
    vertexLabels.clear();
    V = 0;
    E = 0;
}

void Graph::addVertex(const std::string& label) {
    adjList.push_back(std::vector<Edge>());
    vertexLabels.push_back(label);
    V++;
}

void Graph::addEdge(int source, int destination, int weight) {
    if (source < 0 || source >= V || destination < 0 || destination >= V) {
        return;
    }
```

```
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
```

```
    for (int i = 0; i < numVertices; i++) {
        std::string label;
        if (!(file >> label)) {
            label = std::to_string(i + 1);
        }
        addVertex(label);
    }

    int numEdges = 0;
    if (!(file >> numEdges)) {
        return false;
    }

    for (int i = 0; i < numEdges; i++) {
        int source = 0, dest = 0, weight = 0;
        if (file >> source >> dest >> weight) {
            addEdge(source - 1, dest - 1, weight);
        }
    }

    return true;
}

bool Graph::saveToFile(const std::string& filename) const {
    std::ofstream file(filename);
    if (!file.is_open()) {
        return false;
    }

    file << V << "\n";
    for (const auto& label : vertexLabels) {
        file << label << " ";
    }
    file << "\n";

    file << E << "\n";
    for (int i = 0; i < V; i++) {
        for (const auto& edge : adjList[i]) {
            file << (i + 1) << " " << (edge.destination + 1) << " " << edge.weight << "\n
";
        }
    }

    return true;
}

bool Graph::exportForPython(const std::string& filename) const {
    std::ofstream file(filename);
    if (!file.is_open()) {
        return false;
    }

    file << V << "\n";
    for (const auto& label : vertexLabels) {
        file << label << " ";
    }
    file << "\n";

    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 
184 bool Graph::exportWithPath(const std::string& filename, const std::vector<int>& path)
    const {
185     std::ofstream file(filename);
186     if (!file.is_open()) {
187         return false;
188     }
189 
190     file << V << "\n";
191     for (const auto& label : vertexLabels) {
192         file << label << " ";
193     }
194     file << "\n";
195 
196     for (int i = 0; i < V; i++) {
197         for (const auto& edge : adjList[i]) {
198             file << (i + 1) << " " << (edge.destination + 1) << " " << edge.weight << "\n
    ";
199         }
200     }
201 
202     file << "\nPATH:\n";
203     for (int v : path) {
204         file << (v + 1) << " ";
205     }
206     file << "\n";
207 
208     return true;
209 }
210 
211 bool Graph::exportForPython(const std::vector<int>& path) const {
212     return exportWithPath(TEMP_EXPORT_FILE, path);
213 }
214 
215 // ==================== VALIDATION ====================
216 bool Graph::isValid() const {
217     return V > 0;
218 }
219 
220 bool Graph::hasNegativeWeights() const {
221     for (int i = 0; i < V; i++) {
222         for (const auto& edge : adjList[i]) {
223             if (edge.weight < 0) {
224                 return true;
225             }
226         }
227     }
228     return false;
229 }
```

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

```cpp
#include "../lib/Algorithms.h"
#include <queue>
#include <limits>
#include <sstream>
#include <iomanip>

namespace {
std::vector<std::string> formatDistanceTable(const Graph& graph, const std::vector<int>& distances) {
    std::vector<std::string> lines;
    if (distances.empty()) return lines;

    const int INF = std::numeric_limits<int>::max();
    std::ostringstream header;
    std::ostringstream values;

    header << "Dinh :";
    values << "D(i) :";

    for (size_t i = 0; i < distances.size(); ++i) {
        const std::string label = graph.getVertexLabel(static_cast<int>(i));
        header << std::setw(6) << label;
        if (distances[i] == INF) {
            values << std::setw(6) << "INF";
        } else {
            values << std::setw(6) << distances[i];
        }
    }

    lines.push_back(header.str());
    lines.push_back(values.str());
    return lines;
}
} // namespace

Algorithms::Algorithms(const Graph& g) : graph(g) {}

void Algorithms::logStep(std::vector<std::string>& logs, const std::string& message) {
    logs.push_back(message);
}

std::vector<int> Algorithms::reconstructPath(int destination, const std::vector<int>& previousVertex) const {
    std::vector<int> path;
    int current = destination;
    while (current != -1) {
        path.insert(path.begin(), current);
        current = previousVertex[current];
    }
    return path;
}


// Dijkstra
PathResult Algorithms::dijkstra(int start, bool showSteps) {
    PathResult result;
    result.startVertex = start;
```

```
    int V = graph.getVertexCount();
    const auto& adjList = graph.getAdjacencyList();

    const int INF = std::numeric_limits<int>::max();
    result.distances.assign(V, INF);
    result.previousVertex.assign(V, -1);
    result.distances[start] = 0;

    std::priority_queue<std::pair<int, int>, std::vector<std::pair<int, int>>,std::
    greater<std::pair<int, int>>> pq;
    pq.push({0, start});

    if (showSteps) {
        logStep(result.logs, "              ======= THUAT TOAN DIJKSTRA =======");
        logStep(result.logs, "Dinh bat dau: " + graph.getVertexLabel(start));
        logStep(result.logs, "Khoi tao khoang cach: tat ca = INF, rieng dinh bat dau = 0"
    );
    }

    std::vector<bool> visited(V, false);
    int iterations = 0;

    while (!pq.empty()) {
        auto [dist, u] = pq.top();
        pq.pop();

        if (visited[u]) continue;
        visited[u] = true;
        iterations++;

        if (showSteps) {
            logStep(result.logs, "");
            logStep(result.logs, "[Lan lap " + std::to_string(iterations) + "]");
            logStep(result.logs, "Xu ly dinh: " + graph.getVertexLabel(u) +
                                 " (khoang cach = " + std::to_string(dist) + ")");
        }

        for (const auto& edge : adjList[u]) {
            int v = edge.destination;
            int weight = edge.weight;

            if (result.distances[u] != INF &&
                result.distances[u] + weight < result.distances[v]) {

                result.distances[v] = result.distances[u] + weight;
                result.previousVertex[v] = u;
                pq.push({result.distances[v], v});

                if (showSteps) {
                    logStep(result.logs, "  Cap nhat: " + graph.getVertexLabel(u) + " ->
    " +
                                         graph.getVertexLabel(v) +
                                         " (khoang cach moi = " + std::to_string(result.
    distances[v]) + ")");
                }
            }
        }

        if (showSteps) {
```

```
111             logStep(result.logs, "Khoang cach sau lan lap " + std::to_string(iterations)
    + ":");
112             auto table = formatDistanceTable(graph, result.distances);
113             for (const auto& line : table) {
114                 logStep(result.logs, line);
115             }
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.getVertexLabel(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;
```

```
                int weight = edge.weight;

                if (result.distances[u] != INF &&
                    result.distances[u] + weight < result.distances[v]) {

                    result.distances[v] = result.distances[u] + weight;
                    result.previousVertex[v] = u;
                    updated = true;

                    if (showSteps) {
                        logStep(result.logs, "  Cap nhat: " + graph.getVertexLabel(u) + "
 -> " +
                                                graph.getVertexLabel(v) +
                                                " (khoang cach moi = " + std::to_string(
result.distances[v]) + ")");
                    }
                }
            }
        }

        if (!updated && showSteps) {
            logStep(result.logs, "  (Khong co cap nhat o luot nay - co the dung som)");
        }

        if (showSteps) {
            logStep(result.logs, "Bang khoang cach sau luot " + std::to_string(i + 1) + "
:");
            auto table = formatDistanceTable(graph, result.distances);
            for (const auto& line : table) {
                logStep(result.logs, line);
            }
        }
    }

    result.hasNegativeCycle = false;
    if (showSteps) {
        logStep(result.logs, "");
        logStep(result.logs, "=== KIEM TRA CHU TRINH AM ===");
    }

    for (int u = 0; u < V; u++) {
        for (const auto& edge : adjList[u]) {
            int v = edge.destination;
            int weight = edge.weight;

            if (result.distances[u] != INF &&
                result.distances[u] + weight < result.distances[v]) {
                result.hasNegativeCycle = true;

                if (showSteps) {
                    logStep(result.logs, "PHAT HIEN CHU TRINH AM!");
                    logStep(result.logs, "Canh: " + graph.getVertexLabel(u) + " -> " +
                                         graph.getVertexLabel(v) +
                                         " (trong so = " + std::to_string(weight) + ")");
                }
            }
        }
    }

```

```
    if (!result.hasNegativeCycle && showSteps) {
        logStep(result.logs, "Khong phat hien chu trinh am.");
    }

    if (showSteps) {
        logStep(result.logs, "");
        logStep(result.logs, "=== KHOANG CACH CUOI ===");
        for (int i = 0; i < V; i++) {
            if (result.distances[i] == INF) {
                logStep(result.logs, graph.getVertexLabel(i) + " = INF (khong toi duoc)")
    ;
            } else {
                logStep(result.logs, graph.getVertexLabel(i) + " = " + std::to_string(
    result.distances[i]));
            }
        }
    }

    result.success = !result.hasNegativeCycle;
    return result;
}

std::vector<int> Algorithms::getShortestPath(const PathResult& result, int destination)
    const {
    if (destination < 0 || destination >= result.previousVertex.size()) {
        return {};
    }
    if (result.distances.empty()) {
        return {};
    }
    if (result.distances[destination] == std::numeric_limits<int>::max()) {
        return {};
    }
    return reconstructPath(destination, result.previousVertex);
}

int Algorithms::getDistance(const PathResult& result, int destination) const {
    if (destination < 0 || destination >= result.distances.size()) {
        return -1;
    }
    return result.distances[destination];
}
```

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

```
#include "../lib/Comparison.h"
#include <cmath>

// ==================== CONSTRUCTOR ====================
Comparison::Comparison(const Graph& g) : graph(g), algorithms(g) {}

// ==================== PERFORMANCE MEASUREMENT ====================
PerformanceMetrics Comparison::measureAlgorithm(int startVertex, AlgorithmType type) {
    PerformanceMetrics metrics;
    int V = graph.getVertexCount();
    int E = graph.getEdgeCount();

    if (type == AlgorithmType::DIJKSTRA) {
```

```
        metrics.algorithmName = "Dijkstra";

        // Check for negative weights
        if (graph.hasNegativeWeights()) {
            metrics.success = false;
            metrics.executionTimeUs = 0;
            return metrics;
        }

        auto start = std::chrono::high_resolution_clock::now();
        PathResult result = algorithms.dijkstra(startVertex, false);
        auto end = std::chrono::high_resolution_clock::now();

        metrics.executionTimeUs = std::chrono::duration_cast<std::chrono::microseconds>(
    end - start).count();
        metrics.distancesCalculated = result.distances.size();
        metrics.memoryUsageBytes = (V * sizeof(int) * 2) + (E * sizeof(Edge));
        metrics.complexity = E * std::log(V);
        metrics.success = result.success;

    } else if (type == AlgorithmType::BELLMAN_FORD) {
        metrics.algorithmName = "Bellman-Ford";

        auto start = std::chrono::high_resolution_clock::now();
        PathResult result = algorithms.bellmanFord(startVertex, false);
        auto end = std::chrono::high_resolution_clock::now();

        metrics.executionTimeUs = std::chrono::duration_cast<std::chrono::microseconds>(
    end - start).count();
        metrics.distancesCalculated = result.distances.size();
        metrics.memoryUsageBytes = (V * sizeof(int) * 2) + (E * sizeof(Edge));
        metrics.complexity = V * E;
        metrics.success = result.success && !result.hasNegativeCycle;
    }

    return metrics;
}

// ==================== COMPARISON REPORT ====================
ComparisonReport Comparison::comparePerformance(int startVertex, AlgorithmType type) {
    ComparisonReport report;
    report.startVertex = startVertex;
    report.V = graph.getVertexCount();
    report.E = graph.getEdgeCount();

    report.logs.push_back("        ========================================");
    report.logs.push_back("                BAO CAO SO SANH HIEU NANG");
    report.logs.push_back("        ========================================");
    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));

    if (type == AlgorithmType::DIJKSTRA || type == AlgorithmType::BOTH) {
        auto metrics = measureAlgorithm(startVertex, AlgorithmType::DIJKSTRA);
        report.metrics.push_back(metrics);
    }

    if (type == AlgorithmType::BELLMAN_FORD || type == AlgorithmType::BOTH) {
        auto metrics = measureAlgorithm(startVertex, AlgorithmType::BELLMAN_FORD);
        report.metrics.push_back(metrics);
```

```
    }

    if (type == AlgorithmType::BOTH && report.metrics.size() == 2) {
        const auto& d = report.metrics[0];
        const auto& b = report.metrics[1];

        const int labelW = 16;
        const int colW = 20;

        auto utf8Len = [](const std::string& s) {
            int count = 0;
            for (unsigned char c : s) {
                if ((c & 0xC0) != 0x80) {
                    count++;
                }
            }
            return count;
        };

        auto fit = [&](const std::string& s, int width) {
            if (width <= 0) return std::string();
            int len = utf8Len(s);
            if (len == width) return s;
            if (len < width) return s + std::string(width - len, ' ');

            std::string out;
            out.reserve(s.size());
            int count = 0;
            for (size_t i = 0; i < s.size() && count < width; i++) {
                unsigned char c = static_cast<unsigned char>(s[i]);
                if ((c & 0xC0) != 0x80) {
                    if (count >= width) break;
                    count++;
                }
                out.push_back(s[i]);
            }
            int outLen = utf8Len(out);
            if (outLen < width) {
                out += std::string(width - outLen, ' ');
            }
            return out;
        };

        auto row = [&](const std::string& label, const std::string& dv, const std::string
    & bv) {
            return std::string("|") + fit(label, labelW) + "|"
                + fit(dv, colW) + "|" + fit(bv, colW) + "|";
        };

        std::string border = "+" + std::string(labelW, '-') +
                             "+" + std::string(colW, '-') +
                             "+" + std::string(colW, '-') + "+";

        auto fmtStatus = [](bool ok) {
            return ok ? std::string("Thanh cong") : std::string("That bai");
        };

        auto fmtComplexity = [](double value, const std::string& form) {
            return form + "     O(" + std::to_string(static_cast<int>(value)) + ")";
```

```
        };

        report.logs.push_back(border);
        report.logs.push_back(row("", "DIJKSTRA", "BELLMAN-FORD"));
        report.logs.push_back(border);
        report.logs.push_back(row("Thoi gian chay", std::to_string(d.executionTimeUs) + "
    us",
                                  std::to_string(b.executionTimeUs) + " us"));
        report.logs.push_back(row("Bo nho dung", std::to_string(d.memoryUsageBytes) + "
    bytes",
                                  std::to_string(b.memoryUsageBytes) + " bytes"));
        report.logs.push_back(row("Do phuc tap",
                                  fmtComplexity(d.complexity, "O(E log V)"),
                                  fmtComplexity(b.complexity, "O(V    E)")));
        report.logs.push_back(row("Trang thai", fmtStatus(d.success), fmtStatus(b.success
    )));
        report.logs.push_back(border);
    }
    // else if (type == AlgorithmType::DIJKSTRA && report.metrics.size() == 1) {
    //     const auto& metrics = report.metrics[0];
    //     report.logs.push_back("--- THUAT TOAN DIJKSTRA ---");
    //     report.logs.push_back("Thoi gian chay: " + std::to_string(metrics.
    executionTimeUs) + " us");
    //     report.logs.push_back("Bo nho su dung: " + std::to_string(metrics.
    memoryUsageBytes) + " bytes");
    //     report.logs.push_back("Do phuc tap: O(E log V)     O(" + std::to_string((int)
    metrics.complexity) + ")");
    //     report.logs.push_back("Trang thai: " + std::string(metrics.success ? "Thanh co
    ng" : "That bai"));
    //     report.logs.push_back("");
    // } else if (type == AlgorithmType::BELLMAN_FORD && report.metrics.size() == 1) {
    //     const auto& metrics = report.metrics[0];
    //     report.logs.push_back("--- THUAT TOAN BELLMAN-FORD ---");
    //     report.logs.push_back("Thoi gian chay: " + std::to_string(metrics.
    executionTimeUs) + " us");
    //     report.logs.push_back("Bo nho su dung: " + std::to_string(metrics.
    memoryUsageBytes) + " bytes");
    //     report.logs.push_back("Do phuc tap: O(V    E)     O(" + std::to_string((int)
    metrics.complexity) + ")");
    //     report.logs.push_back("Trang thai: " + std::string(metrics.success ? "Thanh co
    ng" : "That bai"));
    //     report.logs.push_back("");
    // }

    if (type == AlgorithmType::BOTH && report.metrics.size() == 2) {
        report.logs.push_back("                          --- SO SANH ---");
        auto dijkstraTime = report.metrics[0].executionTimeUs;
        auto bellmanTime = report.metrics[1].executionTimeUs;

        if (bellmanTime > 0 && dijkstraTime > 0) {
            double ratio = static_cast<double>(bellmanTime) / dijkstraTime;
            report.logs.push_back("Bellman-Ford cham hon Dijkstra " + std::to_string(
    ratio) + " lan");
        } else {
            report.logs.push_back("Khong du do chenh lech thoi gian de so sanh.");
        }
        report.logs.push_back("");
    }

```

```
175
176     return report;
177 }
```

.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 = "Nhap lua chon (1-6): ";
17 const char* PRESS_ANY_KEY = "Nhan phim bat ky de tiep tuc...";
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 }
52
```

```
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
      maxWidth) {
98     std::vector<std::string> out;
99     for (const auto& line : lines) {
100         auto parts = wrapLineToWidth(line, maxWidth);
101         out.insert(out.end(), parts.begin(), parts.end());
102     }
103     return out;
104 }
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) {
```

```
    if (maxLen < 1) maxLen = 1;
    std::string blank(static_cast<size_t>(maxLen), ' ');
    outtextxy(x, y, (char*)blank.c_str());
}

std::string readLineAt(int x, int y, int maxLen, InputFilter filter) {
    std::string value;
    setcolor(COLOR_TEXT);
    drawInputField(x, y, maxLen, value);
    while (true) {
        int c = getch();
        if (c == '\r' || c == '\n') {
            break;
        }
        if (c == 8 || c == 127) {
            if (!value.empty()) {
                value.pop_back();
            }
        } else if (c >= 32 && c <= 126) {
            char ch = static_cast<char>(c);
            bool accept = true;
            if (filter == InputFilter::Integer) {
                if (std::isdigit(static_cast<unsigned char>(ch))) {
                    accept = true;
                } else if ((ch == '-' || ch == '+') && value.empty()) {
                    accept = true;
                } else {
                    accept = false;
                }
            }
            if (accept && (int)value.size() < maxLen) {
                value.push_back(ch);
            }
        }
        drawInputField(x, y, maxLen, value);
    }
    return value;
}

void drawHorizontalRule(int leftX, int rightX, int y) {
    int charW = approxCharWidth();
    int count = (rightX - leftX) / (charW > 0 ? charW : 1);
    if (count < 2) count = 2;
    std::string rule;
    rule.reserve(static_cast<size_t>(count));
    rule.push_back('+');
    if (count > 2) {
        rule.append(static_cast<size_t>(count - 2), '-');
    }
    rule.push_back('+');
    outtextxy(leftX, y, (char*)rule.c_str());
}

void drawFrameBox(const std::string& title, int left, int top, int right, int bottom,
                  int& centerX, int& innerLeft, int& innerTop) {
    setcolor(COLOR_BUTTON);
    rectangle(left, top, right, bottom);
    rectangle(left + 4, top + 4, right - 4, bottom - 4);

```

```
    centerX = (left + right) / 2;
    int lineH = approxLineHeight();
    int titleY = top + 8;

    if (!title.empty()) {
        setcolor(COLOR_TEXT);
        drawCenteredText(centerX, titleY, title);
        innerTop = titleY + lineH;
    } else {
        innerTop = top + FRAME_PADDING;
    }

    innerLeft = leftIndentX(left);
}

void drawHeaderFrame(int screenW) {
    int extra = 2 * approxCharWidth();
    int left = OUTER_MARGIN - extra;
    int right = screenW - OUTER_MARGIN + extra;
    if (left < 0) left = 0;
    if (right > screenW) right = screenW;
    int top = OUTER_MARGIN;
    int bottom = top + headerHeight();

    int centerX = 0;
    int innerLeft = 0;
    int innerTop = 0;
    drawFrameBox("", left, top, right, bottom, centerX, innerLeft, innerTop);

    int lineH = approxLineHeight();
    int y = top + 12;

    setcolor(COLOR_TEXT);
    y += lineH;
    drawCenteredText(centerX, y, "TRUONG DAI HOC BACH KHOA - DAI HOC DA NANG");
    y += lineH + 4;
    // drawCenteredText(centerX, y, "|");
    // y += lineH + 2;

    setcolor(YELLOW);
    drawCenteredText(centerX, y, "PBL1 : DO AN LAP TRINH TINH TOAN");
    y += lineH + 14;
    const int paddingX = 40;

    setcolor(COLOR_TEXT);
    const std::string leftText = "Ten SV: Nguyen Huu Rin";
    const std::string rightText = "GVHD: Nguyen Van Hieu";
    outtextxy(390, y, (char*)leftText.c_str());
    outtextxy(right - 7*paddingX - approxTextWidth(rightText), y, (char*)rightText.c_str
    ());
    y += lineH + 4;
    drawCenteredText(centerX-25, y, "Huynh Nguyen Hong Nhi");
}

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

```
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;
```

```
            if (maxLinesPerPage > safetyLines) {
                maxLinesPerPage -= safetyLines;
            }
        }

        setcolor(LIGHTGREEN);
        int yPos = innerTop + 6;
        int count = 0;
        while (index < wrappedLogs.size() && count < maxLinesPerPage) {
            outtextxy(innerLeft, yPos, (char*)wrappedLogs[index].c_str());
            yPos += lineH + 4;
            index++;
            count++;
        }

        bool hasMore = index < wrappedLogs.size();
        setcolor(LIGHTCYAN);
        if (hasMore) {
            drawCenteredText(centerX, bottom - lineH - 4, "Nhan phim bat ky de xem tiep
    ...");
            getch();
            continue;
        }

        drawCenteredText(centerX, bottom - lineH - 4, "Nhan phim bat ky de quay lai menu
    ...");
        break;
    }
}

void GUI::clearScreen() {
    cleardevice();
}

int GUI::promptMenuChoice() {
    while (true) {
        drawMenu();
        int lineH = approxLineHeight();

        int left = OUTER_MARGIN;
        int menuTop = OUTER_MARGIN + headerHeight() + HEADER_GAP;
        int menuBottom = menuTop + menuHeight();
        int choiceTop = menuBottom + HEADER_GAP;

        int promptY = choiceTop + (choiceHeight() - lineH) / 2;
        int inputX = leftIndentX(left) + approxTextWidth(MENU_PROMPT) + approxCharWidth()
    ;

        std::string input = readLineAt(inputX, promptY, 3, InputFilter::Integer);
        int value = 0;
        if (tryParseInt(input, value) && value >= 1 && value <= 6) {
            return value;
        }

        setcolor(LIGHTRED);
        drawLeftAlignedText(left + FRAME_PADDING, promptY + lineH, "Du lieu khong hop le.
     Vui long thu lai.");
        waitForKey();
    }
```

```
}

int GUI::promptChoice(const std::string& title, const std::vector<std::string>& options,
                      const std::string& prompt, int minValue, int maxValue) {
    std::string error;
    while (true) {
        clearScreen();
        setbkcolor(COLOR_BACKGROUND);
        drawHeaderFrame(WINDOW_WIDTH);

        int left, top, right, bottom, centerX, innerLeft, innerTop;
        drawContentFrame(title, WINDOW_WIDTH, WINDOW_HEIGHT,
                         left, top, right, bottom, centerX, innerLeft, innerTop);

        int lineH = approxLineHeight();
        int y = innerTop + 4;

        setcolor(YELLOW);
        for (const auto& opt : options) {
            drawLeftAlignedText(innerLeft, y, opt);
            y += lineH;
        }

        y += lineH / 2;
        setcolor(COLOR_TEXT);
        drawLeftAlignedText(innerLeft, y, prompt);

        int inputX = innerLeft + approxTextWidth(prompt) + approxCharWidth();
        int inputY = y;

        if (!error.empty()) {
        setcolor(LIGHTRED);
        drawLeftAlignedText(bottomIndentX(left), y + lineH, error);
        }

        std::string input = readLineAt(inputX, inputY, 6, InputFilter::Integer);
        int value = 0;
        if (tryParseInt(input, value) && value >= minValue && value <= maxValue) {
            return value;
        }

        error = "Du lieu khong hop le. Vui long thu lai.";
    }
}

int GUI::promptInt(const std::string& title, const std::string& prompt, int minValue, int
     maxValue) {
    std::string error;
    while (true) {
        clearScreen();
        setbkcolor(COLOR_BACKGROUND);
        drawHeaderFrame(WINDOW_WIDTH);

        int left, top, right, bottom, centerX, innerLeft, innerTop;
        drawContentFrame(title, WINDOW_WIDTH, WINDOW_HEIGHT,
                         left, top, right, bottom, centerX, innerLeft, innerTop);

        int lineH = approxLineHeight();
        int y = innerTop + 6;
```

```
515
516         setcolor(COLOR_TEXT);
517         drawLeftAlignedText(innerLeft, y, prompt);
518
519         int inputX = innerLeft + approxTextWidth(prompt) + approxCharWidth();
520         int inputY = y;
521
522         if (!error.empty()) {
523         setcolor(LIGHTRED);
524         drawLeftAlignedText(bottomIndentX(left), y + lineH, error);
525         }
526
527         std::string input = readLineAt(inputX, inputY, 8, InputFilter::Integer);
528         int value = 0;
529         if (tryParseInt(input, value) && value >= minValue && value <= maxValue) {
530             return value;
531         }
532
533         error = "Du lieu khong hop le. Vui long thu lai.";
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) -
    2;
554     if (maxLen < 8) maxLen = 8;
555
556     std::string input = readLineAt(inputX, inputY, maxLen, InputFilter::Any);
557     if (input.empty()) {
558         return defaultValue;
559     }
560     return input;
561 }
562
563 bool GUI::promptYesNo(const std::string& title, const std::string& prompt) {
564     while (true) {
565         std::string input = promptLine(title, prompt + " (y/n): ");
566         if (input.empty()) {
567             continue;
568         }
569         char c = static_cast<char>(std::tolower(static_cast<unsigned char>(input[0])));
570         if (c == 'y') return true;
571         if (c == 'n') return false;
572         showMessage(title, {"Vui long nhap y hoac n."});
```

```
    }
}

void GUI::promptStartEnd(const std::string& title, int minValue, int maxValue,
                         int& startValue, int& endValue) {
    clearScreen();
    setbkcolor(COLOR_BACKGROUND);
    drawHeaderFrame(WINDOW_WIDTH);

    int left, top, right, bottom, centerX, innerLeft, innerTop;
    drawContentFrame(title, WINDOW_WIDTH, WINDOW_HEIGHT,
                     left, top, right, bottom, centerX, innerLeft, innerTop);

    int lineH = approxLineHeight();
    int y = innerTop + 6;

    setcolor(COLOR_TEXT);
    std::string promptStart = "Dinh bat dau (" + std::to_string(minValue) + ".." + std::
    to_string(maxValue) + "): ";
    std::string promptEnd = "Dinh ket thuc (" + std::to_string(minValue) + ".." + std::
    to_string(maxValue) + "): ";

    int charW = approxCharWidth();
    if (charW < 1) charW = 1;
    int errorX = innerLeft;
    int errorY = bottom - lineH - 6;
    int errorMaxChars = (right - innerLeft) / charW;

    auto clearError = [&]() {
        setcolor(COLOR_TEXT);
        clearTextLine(errorX, errorY, errorMaxChars);
    };
    auto showError = [&](const std::string& msg) {
        clearError();
        setcolor(LIGHTRED);
        drawLeftAlignedText(errorX, errorY, msg);
        setcolor(COLOR_TEXT);
    };

    drawLeftAlignedText(innerLeft, y, promptStart);
    int inputXStart = innerLeft + approxTextWidth(promptStart) + approxCharWidth();
    while (true) {
        std::string input = readLineAt(inputXStart, y, 8, InputFilter::Integer);
        int value = 0;
        if (tryParseInt(input, value) && value >= minValue && value <= maxValue) {
            startValue = value;
            clearError();
            break;
        }
        showError("Gia tri dinh bat dau khong hop le.");
    }

    y += lineH;
    drawLeftAlignedText(innerLeft, y, promptEnd);
    int inputXEnd = innerLeft + approxTextWidth(promptEnd) + approxCharWidth();
    while (true) {
        std::string input = readLineAt(inputXEnd, y, 8, InputFilter::Integer);
        int value = 0;
        if (tryParseInt(input, value) && value >= minValue && value <= maxValue) {
```

```
            endValue = value;
            clearError();
            break;
        }
        showError("Gia tri dinh ket thuc khong hop le.");
    }
}

void GUI::promptGraphInput(bool isDirected, int& numVertices, int& numEdges,
                           std::vector<std::tuple<int, int, int>>& edges) {
    edges.clear();
    numVertices = 0;
    numEdges = 0;

    clearScreen();
    setbkcolor(COLOR_BACKGROUND);
    drawHeaderFrame(WINDOW_WIDTH);

    int left, top, right, bottom, centerX, innerLeft, innerTop;
    drawContentFrame("TAO DO THI", WINDOW_WIDTH, WINDOW_HEIGHT,
                     left, top, right, bottom, centerX, innerLeft, innerTop);

    int lineH = approxLineHeight();
    int y = innerTop + 6;

    setcolor(YELLOW);
    drawLeftAlignedText(innerLeft, y, "Nhap thong tin do thi:");
    y += lineH + 4;

    setcolor(COLOR_TEXT);
    int charW = approxCharWidth();
    if (charW < 1) charW = 1;
    int errorX = innerLeft;
    int errorY = bottom - lineH - 6;
    int errorMaxChars = (right - innerLeft) / charW;

    auto clearError = [&]() {
        setcolor(COLOR_TEXT);
        clearTextLine(errorX, errorY, errorMaxChars);
    };
    auto showError = [&](const std::string& msg) {
        clearError();
        setcolor(LIGHTRED);
        drawLeftAlignedText(errorX, errorY, msg);
        setcolor(COLOR_TEXT);
    };

    std::string promptV = "So dinh (1-100): ";
    drawLeftAlignedText(innerLeft, y, promptV);
    int inputXV = innerLeft + approxTextWidth(promptV) + approxCharWidth();
    while (true) {
        std::string input = readLineAt(inputXV, y, 6, InputFilter::Integer);
        int value = 0;
        if (tryParseInt(input, value) && value >= 1 && value <= 100) {
            numVertices = value;
            clearError();
            break;
        }
        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::
    to_string(maxEdgeLines) + ".");
711                 continue;
712             }
713             numEdges = value;
714             clearError();
715             break;
716         }
717         showError("So canh khong hop le.");
718     }
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::
    to_string(numVertices) + "].");
745         }
```

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

```
802
803         setcolor(LIGHTGREEN);
804         int yPos = innerTop;
805         int count = 0;
806         while (index < wrappedLogs.size() && count < maxLinesPerPage) {
807             outtextxy(innerLeft, yPos, (char*)wrappedLogs[index].c_str());
808             yPos += lineStep;
809             index++;
810             count++;
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 lineH = 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 - lineH * 2) break;
832         drawLeftAlignedText(messageX, y, line);
833         y += lineH;
834     }
835
836     setcolor(LIGHTCYAN);
837     drawCenteredText(centerX, bottom - lineH - 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 thich 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;
```

```
        case CYAN: return 36;
        case RED: return 31;
        case MAGENTA: return 35;
        case BROWN: return 33;
        case LIGHTGRAY: return 37;
        case DARKGRAY: return 90;
        case LIGHTBLUE: return 94;
        case LIGHTGREEN: return 92;
        case LIGHTCYAN: return 96;
        case LIGHTRED: return 91;
        case LIGHTMAGENTA: return 95;
        case YELLOW: return 93;
        case WHITE: return 97;
        default: return 37;
    }
}

static void moveCursor(int row, int col) {
    if (!g_ansi_enabled) return;
    std::cout << "\x1b[" << row << ";" << col << "H";
}

static void applyColor() {
    if (!g_ansi_enabled) return;
    std::cout << "\x1b[" << ansiColorCode(g_current_color) << "m";
}

static bool writeUtf8(const char* text) {
    if (!text) return false;
#ifdef _WIN32
    HANDLE hOut = GetStdHandle(STD_OUTPUT_HANDLE);
    if (hOut == INVALID_HANDLE_VALUE) {
        return false;
    }
    int wlen = MultiByteToWideChar(CP_UTF8, 0, text, -1, nullptr, 0);
    if (wlen <= 0) {
        return false;
    }
    std::wstring wstr(static_cast<size_t>(wlen), L'\0');
    MultiByteToWideChar(CP_UTF8, 0, text, -1, &wstr[0], wlen);
    DWORD written = 0;
    BOOL ok = WriteConsoleW(hOut, wstr.c_str(), static_cast<DWORD>(wlen - 1), &written,
    nullptr);
    return ok != FALSE;
#else
    std::cout << text;
    return true;
#endif
}

void initgraph(int *graphdriver, int *graphmode, char *pathtodriver) {
    (void)graphdriver; (void)graphmode; (void)pathtodriver;
    g_graph_result = 0; // grOk
    enableAnsi();
    cleardevice();
}

int graphresult() {
    return g_graph_result;
```

```
}

void closegraph() {
    if (g_ansi_enabled) {
        std::cout << "\x1b[0m" << std::flush;
    }
}

void cleardevice() {
    if (g_ansi_enabled) {
        std::cout << "\x1b[2J\x1b[H" << std::flush;
    } else {
        std::cout << std::string(50, '\n');
    }
}

void setcolor(int color) {
    g_current_color = color;
}

void setbkcolor(int color) { (void)color; }

void outtextxy(int x, int y, char *textstring) {
    if (!textstring) return;
    if (g_ansi_enabled) {
        int row = mapY(y);
        int col = mapX(x);
        moveCursor(row, col);
        applyColor();
        if (!writeUtf8(textstring)) {
            std::cout << textstring;
        }
        std::cout << std::flush;
    } else {
        if (!writeUtf8(textstring)) {
            std::cout << textstring;
        }
        std::cout << std::endl;
    }
}

void rectangle(int left, int top, int right, int bottom) {
    if (!g_ansi_enabled) return;
    int r1 = mapY(top);
    int r2 = mapY(bottom);
    int c1 = mapX(left);
    int c2 = mapX(right);
    if (r2 <= r1 || c2 <= c1) return;

    applyColor();
    // canh tren va duoi
    moveCursor(r1, c1);
    std::cout << "+" << std::string(c2 - c1 - 1, '=') << "+";
    moveCursor(r2, c1);
    std::cout << "+" << std::string(c2 - c1 - 1, '-') << "+";
    //khung phia ngoai (hai ben)
    for (int r = r1+1; r < r2; r++) {
        moveCursor(r, c1);
        std::cout << "|";
```

```
        std::cout << "|";

        moveCursor(r, c2);
        std::cout << "|";
    }
    std::cout << std::flush;
}

int getch() {
    int c = std::getchar();
    return c;
}
```

.3 Mã nguồn các thư viện tự cài đặt

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

```
#pragma once
#include <string>
#include <vector>
#include "Global.h"

struct Edge {
    int destination;
    int weight;
    Edge(int dest, int w) : destination(dest), weight(w) {}
};

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

public:
    Graph();
    explicit Graph(int vertices);
    ~Graph();

    int getVertexCount() const;
    int getEdgeCount() const;
    const std::vector<std::vector<Edge>>& getAdjacencyList() const;
    std::string getVertexLabel(int vertex) const;

    void clear();
    void addVertex(const std::string& label);
    void addEdge(int source, int destination, int weight);
    bool hasEdge(int source, int destination) const;
    void makeUndirected();

    bool fileExists(const std::string& filename) const;
    bool readFromFile(const std::string& filename, bool& needCreate);
    bool saveToFile(const std::string& filename) const;
    bool exportForPython(const std::string& filename) const;
    bool exportForPython(const std::vector<int>& path) const;
    bool exportWithPath(const std::string& filename, const std::vector<int>& path) const;
```

```
41
42      bool isValid() const;
43      bool hasNegativeWeights() const;
44  };
```

.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

```
#ifndef GUI_H
#define GUI_H

#include "graphics.h"
#include <vector>
#include <string>
#include <tuple>
#include "Global.h"
#include "Colors.h"

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 promptChoice(const std::string& title, const std::vector<std::string>& options,
                     const std::string& prompt, int minValue, int maxValue);
    int promptInt(const std::string& title, const std::string& prompt, int minValue, int
    maxValue);
    std::string promptLine(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 isDirected, int& numVertices, int& numEdges,
                          std::vector<std::tuple<int, int, int>>& edges);
    void promptStartEnd(const std::string& title, int minValue, int maxValue,
                        int& startValue, int& endValue);
    void showAlgorithmLogs(const std::string& title, const std::vector<std::string>& logs
    );
    void showMessage(const std::string& title, const std::vector<std::string>& lines);
    void waitForKey();
};

#endif
```

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

```
#ifndef ALGORITHMS_H
#define ALGORITHMS_H

#include <vector>
#include <string>
#include <utility>
#include "Graph.h"


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) {}
};

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;
};

#endif
```

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

```
#ifndef COMPARISON_H
#define COMPARISON_H

#include <string>
#include <vector>
#include <chrono>
#include "Algorithms.h"
#include "Graph.h"

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) {}
};


struct ComparisonReport {
```

```
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 =
        AlgorithmType::BOTH);
43
44      PerformanceMetrics measureAlgorithm(int startVertex, AlgorithmType type);
45  };
46
47  #endif
```