# LỜI CẢM ƠN

*Em xin chân thành cảm ơn Trường Đại học - đã tạo điều kiện thuận lợi cho Em thực hiện đồ án này. Em cũng xin gửi lời cảm ơn sâu sắc đến TS. -, giáo viên hướng dẫn của Em, đã tận tình hướng dẫn và đưa ra những gợi ý quý báu cho Em trong suốt quá trình nghiên cứu. Em cũng xin bày tỏ lòng biết ơn đến các bạn trong nhóm nghiên cứu về robot phục vụ, đã chia sẻ kiến thức và kinh nghiệm với Em. Em cũng không quên gửi lời cảm ơn đến gia đình và bạn bè đã luôn ủng hộ và động viên Em trong thời gian qua.*

*Em xin chịu trách nhiệm về mọi sai sót có thể có trong bài báo của mình. Em rất mong nhận được những ý kiến đóng góp của các nhà khoa học và độc giả để hoàn thiện hơn nữa bài báo này.*

**ĐÁNH GIÁ KẾT QUẢ THỰC HIỆN NIÊN LUẬN CƠ SỞ NGÀNH KTPM**

(Học kỳ 2, Năm học 2022-2023)

**GIÁO VIÊN HƯỚNG DẪN:**

|  |  |  |
| --- | --- | --- |
| STT | HỌ VÀ TÊN | MSCB |
| 1 | **PHAN HUY CƯỜNG** |  |

**SINH VIÊN THỰC HIỆN:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| STT | HỌ VÀ TÊN | MSSV | THƯỞNG  (Tối đa 1,0 điểm) | ĐIỂM |
| 1 |  |  |  |  |

|  |  |  |
| --- | --- | --- |
| **I. HÌNH THỨC** *(Tối đa 0,5 điểm)* |  |  |
| **Bìa** *(tối đa 0,25 điểm)* |  |  |
| * Các tiêu đề: Trường ĐHCT, Khoa CNTT * Loại niên luận: Cơ sở ngành KTPM, Tên đề tài  Giáo viên hướng dẫn: chức danh, họ tên. * Thông tin về các sinh viên thực hiện: họ tên, mã số, lớp  Năm thực hiện   **Bố cục** *(tối đa 0.25 điểm)* |  |  |
|  |
| * Nhận xét của giáo viên hướng dẫn và giáo viên chấm * Mục lục: cấu trúc chương, mục và tiểu mục * Phụ lục (nếu có) * Tài liệu tham khảo |  |  |
| **II. NỘI DUNG** *(Tối đa 3,5 điểm)* |  |  |
| **Tổng quan** *(tối đa 0,5 điểm)* |  |  |
| * Mô tả bài toán, mục tiêu cần đạt được *(0,25 điểm)* * Hướng giải quyết và kế hoạch thực hiện *(0,25 điểm)* **Lý thuyết** *(tối đa 0,5 điểm)* |  |  |
|  |
| * Các khái niệm sử dụng trong đề tài * Kết quả vận dụng lý thuyết vào đề tài |  |  |
| **Ứng dụng** *(tối đa 2,0 điểm)* |  |  |
| * Phân tích yêu cầu bài toán, xây dựng các cấu trúc dữ liệu cần thiết *(tối đa 0,5 điểm)* * Giải thuật (Lưu đồ-Ngôn ngữ giả) *(1,0 điểm)* * Giới thiệu chương trình *(0,5 điểm)*   **Kết luận** *(tối đa 0,5 điểm)* |  |  |
|  |
| * Nhận xét kết quả đạt được * Hạn chế * Hướng phát triển |  |  |
| **III. CHƯƠNG TRÌNH DEMO** *(Tối đa 5,0 điểm)* |  |  |
| **Giao diện thân thiện với người dùng** *(1.0 điểm)* |  |  |
| **Hướng dẫn sử dụng** *(0,5 điểm)* |  |  |
| **Kết quả thực hiện đúng với kết quả của phần ứng dụng** *(3,5 điểm)* |  |  |

Cần Thơ, ngày tháng năm 2023

GIÁO VIÊN CHẤM

**MỤC LỤC**

[LỜI CẢM ƠN 1](#_Toc144947184)

[1. CHƯƠNG I: TỔNG QUAN 1](#_Toc144947185)

[1.1. Mô tả bài toán 1](#_Toc144947186)

[1.2. Mục tiêu cần đạt được 1](#_Toc144947187)

[1.3. Hướng giải quyết 2](#_Toc144947188)

[1.4. Kế hoạch thực hiện 2](#_Toc144947189)

[2. CHƯƠNG II: LÝ THUYẾT 4](#_Toc144947190)

[2.1. Khái quát về ngôn ngữ lập trình java 4](#_Toc144947191)

[2.1.1 Giới thiệu tổng quát. 4](#_Toc144947192)

[2.1.2 Ứng dụng nổi bật 5](#_Toc144947193)

[2.1.3 Ưu và nhược điểm của JavaScript 6](#_Toc144947194)

[2.2. Lý thuyết về các thuật toán 8](#_Toc144947195)

[2.1.1 Lý thuyết về thuật toán Bellman Ford 8](#_Toc144947196)

[2.2.2 Thuật toán duyệt theo chiều rộng (Breadth-first-search-BFS) 11](#_Toc144947197)

[2.2.3 Thuật toán duyệt theo chiều sâu (Depth First Search (DFS)) 13](#_Toc144947198)

[3. CHƯƠNG III: ỨNG DỤNG 15](#_Toc144947199)

[3.1. Phân tích yêu cầu bài toán 15](#_Toc144947200)

[3.2. Xây dựng các cấu trúc dữ liệu cần thiết 15](#_Toc144947201)

[3.2.1. Class Algorithms.java 15](#_Toc144947202)

[3.2.2. Class appFrame.java 17](#_Toc144947203)

[3.2.3. Class DrawPanel.java 17](#_Toc144947204)

[3.2.4. Class Edge.java 19](#_Toc144947205)

[3.2.5. Class Graph.java 20](#_Toc144947206)

[3.2.6. Class MyPanel.java 21](#_Toc144947207)

[3.2.7. Class Vertex.java 23](#_Toc144947208)

[3.3. Xây dựng lưu đồ giải thuật 25](#_Toc144947209)

[4. CHƯƠNG IV: THIẾT KẾ - CÀI ĐẶT 26](#_Toc144947210)

[4.1. Thiết kế giao diện 26](#_Toc144947211)

[4.1.1. Khai quát giao diện và các nút chức năng 26](#_Toc144947212)

[4.2. Thiết kế các hàm khởi tạo và thuật toán 27](#_Toc144947213)

[5. CHƯƠNG V: TỔNG KẾT 30](#_Toc144947214)

[5.1. Nhận xét kết quả đạt được 30](#_Toc144947215)

[5.2. Hướng phát triển 30](#_Toc144947216)

[TÀI LIỆU THAM KHẢO 32](#_Toc144947217)

# 1. CHƯƠNG I: TỔNG QUAN

## 1.1. Mô tả bài toán

Bài toán tìm đường đi ngắn nhất từ một đỉnh nguồn đến các đỉnh khác trong một đồ thị có hướng có trọng số, trong đó một số cạnh có thể có trọng số âm. Bài toán này có thể được giải quyết bằng thuật toán Bellman-Ford, một thuật toán quy nạp cơ bản dựa trên việc cập nhật liên tục các nhãn khoảng cách của các đỉnh theo các cạnh của đồ thị. Thuật toán Bellman-Ford có thời gian chạy là O(V\*E), trong đó V là số đỉnh và E là số cạnh của đồ thị. Thuật toán này cũng có thể phát hiện được sự tồn tại của chu trình âm trong đồ thị, nếu có123.

Một số bước cơ bản để mô phỏng thuật toán Bellman-Ford bằng Java là:

* Khai báo các lớp và biến cần thiết để lưu trữ thông tin về đồ thị, như danh sách các đỉnh, danh sách các cạnh, trọng số của các cạnh, nhãn khoảng cách và đỉnh liền trước của các đỉnh.
* Viết hàm khởi tạo giá trị ban đầu cho các biến, trong đó nhãn khoảng cách của đỉnh nguồn được gán bằng 0, nhãn khoảng cách của các đỉnh khác được gán bằng vô cùng, và đỉnh liền trước của tất cả các đỉnh được gán bằng null.
* Viết hàm thực hiện thuật toán Bellman-Ford, trong đó lặp lại quá trình duyệt qua tất cả các cạnh của đồ thị và cập nhật nhãn khoảng cách và đỉnh liền trước của các đỉnh nếu có thể. Hàm này cũng kiểm tra điều kiện dừng khi không có sự thay đổi nào xảy ra hoặc khi phát hiện được chu trình âm.
* Viết hàm in ra kết quả của thuật toán, bao gồm nhãn khoảng cách và đường đi ngắn nhất từ đỉnh nguồn đến các đỉnh khác. Hàm này cũng thông báo nếu có chu trình âm trong đồ thị.
* Viết hàm main để nhập vào dữ liệu về đồ thị và gọi các hàm khác để giải quyết bài toán.

## 1.2. Mục tiêu cần đạt được

Mục tiêu cần đạt được khi làm báo cáo đề tài mô phỏng thuật toán Bellman-Ford bằng Java là:

* Hiểu được nguyên lý hoạt động và cách cài đặt của thuật toán Bellman-Ford, một thuật toán quan trọng trong lý thuyết đồ thị và lập trình.
* Biết cách sử dụng ngôn ngữ lập trình Java để viết code cho thuật toán Bellman-Ford, bao gồm cách khai báo các lớp, biến, hàm, vòng lặp, điều kiện và xuất nhập dữ liệu.
* Có khả năng phân tích và kiểm tra tính đúng đắn, hiệu quả và ổn định của code, cũng như xử lý các trường hợp ngoại lệ như chu trình âm hoặc đồ thị rỗng.
* Có kỹ năng trình bày báo cáo một cách rõ ràng, logic và súc tích, bao gồm phần mô tả bài toán, phương pháp giải quyết, kết quả và nhận xét.

## 1.3. Hướng giải quyết

Hướng giải quyết bài toán mô phỏng thuật toán tìm đường đi ngắn nhất Bellman-Ford là:

* Bước 1: Nhập vào dữ liệu về đồ thị, bao gồm số đỉnh, số cạnh, trọng số của các cạnh và đỉnh nguồn. Có thể sử dụng Scanner để nhập dữ liệu từ bàn phím hoặc từ một file văn bản.
* Bước 2: Khởi tạo các biến cần thiết để lưu trữ thông tin về đồ thị, như một mảng hai chiều để lưu trọng số của các cạnh, một mảng một chiều để lưu nhãn khoảng cách của các đỉnh, và một mảng một chiều để lưu đỉnh liền trước của các đỉnh. Gán giá trị ban đầu cho các biến này theo hàm khởi tạo đã viết ở trên.
* Bước 3: Thực hiện thuật toán Bellman-Ford theo hàm đã viết ở trên. Lặp lại quá trình duyệt qua tất cả các cạnh của đồ thị và cập nhật nhãn khoảng cách và đỉnh liền trước của các đỉnh nếu có thể. Kiểm tra điều kiện dừng khi không có sự thay đổi nào xảy ra hoặc khi phát hiện được chu trình âm.
* Bước 4: In ra kết quả của thuật toán, bao gồm nhãn khoảng cách và đường đi ngắn nhất từ đỉnh nguồn đến các đỉnh khác. Thông báo nếu có chu trình âm trong đồ thị.

## 1.4. Kế hoạch thực hiện

Kế hoạch thực hiện bài toán mô phỏng thuật toán Bellman-Ford là:

Bước 1: Tìm hiểu về thuật toán Bellman-Ford, nguyên lý hoạt động, ưu nhược điểm, cách cài đặt và các ví dụ minh họa.

Bước 2: Thiết kế và xây dựng đồ thị để mô phỏng thuật toán, bao gồm chọn số đỉnh, số cạnh, trọng số của các cạnh và đỉnh nguồn.

Bước 3: Viết code Java cho thuật toán Bellman-Ford theo các bước cơ bản đã nêu ở trên.

Bước 4: Kiểm tra và đánh giá code, bao gồm kiểm tra tính đúng đắn, hiệu quả và ổn định của code, xử lý các trường hợp ngoại lệ như chu trình âm hoặc đồ thị rỗng, so sánh kết quả với các ví dụ đã có hoặc với các thuật toán khác như Dijkstra hay Floyd-Warshall.

Bước 5: Viết báo cáo đề tài, bao gồm phần mô tả bài toán, phương pháp giải quyết, kết quả và nhận xét.

# 2. CHƯƠNG II: LÝ THUYẾT

## 2.1. Khái quát về ngôn ngữ lập trình Java

### 2.1.1 Giới thiệu tổng quát.

Java là một ngôn ngữ lập trình hướng đối tượng, dựa trên lớp, được thiết kế để có khả năng chạy trên nhiều nền tảng khác nhau mà không cần thay đổi mã nguồn. Java được phát triển bởi James Gosling và các đồng nghiệp tại Sun Microsystems vào năm 1991 và được phát hành vào năm 1995. Java có tiêu chí “Viết một lần, chạy khắp nơi” (Write Once, Run Anywhere), nghĩa là mã Java đã biên dịch có thể chạy trên bất kỳ máy ảo Java (JVM) nào bất kể kiến trúc máy tính bên dưới.

Java có các đặc điểm sau:

* Tương tự C++, nhưng có ít cơ sở cấp thấp hơn và loại bỏ các thao tác với con trỏ, overload, goto và các cấu trúc như struct và union.
* Độc lập phần cứng và hệ điều hành, có thể chạy trên nhiều môi trường khác nhau thông qua JVM.
* Ngôn ngữ thông dịch, mã Java được biên dịch thành bytecode và được thông dịch bởi JVM khi chạy.
* Có cơ chế thu gom rác tự động (garbage collection), giải phóng bộ nhớ cho các đối tượng không còn sử dụng.
* Hỗ trợ đa luồng (multithreading), cho phép thực hiện nhiều tác vụ đồng thời trong một chương trình.
* Tính an toàn và bảo mật, kiểm tra kiểu dữ liệu tĩnh, quản lý quyền truy cập và mã hóa dữ liệu.

Java là một nền tảng lập trình, bao gồm JVM và các thư viện lớp thực thi. Java có nhiều phiên bản khác nhau, trong đó phổ biến nhất là Java Standard Edition (Java SE), Java Enterprise Edition (Java EE) và Java Micro Edition (Java ME). Java SE là phiên bản cơ bản, hỗ trợ các ứng dụng máy tính để bàn, máy chủ và thiết bị nhúng. Java EE là phiên bản mở rộng của Java SE, hỗ trợ các ứng dụng web và doanh nghiệp. Java ME là phiên bản thu gọn của Java SE, hỗ trợ các ứng dụng cho thiết bị di động và IoT.

### 2.1.2 Ứng dụng nổi bật

Một số ứng dụng nổi bật của ngôn ngữ lập trình Java là:

* Ứng dụng Android: Java là ngôn ngữ chính để phát triển các ứng dụng cho hệ điều hành Android, một trong những hệ điều hành phổ biến nhất trên thế giới. Các ứng dụng Android được viết bằng Java với Android API của Google, cho phép tận dụng các tính năng và dịch vụ của hệ điều hành này. Ví dụ: Gmail, YouTube, Facebook, Instagram, Zalo
* Ứng dụng web: Java cũng là một ngôn ngữ lập trình phổ biến để phát triển các ứng dụng web, bao gồm các trang web tĩnh và động, các ứng dụng doanh nghiệp và các ứng dụng thương mại điện tử. Java cung cấp nhiều công nghệ và thư viện để hỗ trợ lập trình viên trong việc xây dựng các ứng dụng web hiệu quả và an toàn. Ví dụ: Servlet, JSP, JSF, Spring, Hibernate, Struts
* Ứng dụng máy tính để bàn: Java cũng có thể được sử dụng để phát triển các ứng dụng máy tính để bàn, có giao diện đồ họa và chức năng đa năng. Java cung cấp các thư viện giao diện người dùng như Swing, AWT và JavaFX để giúp lập trình viên tạo ra các giao diện đẹp mắt và thân thiện với người dùng. Ví dụ: Eclipse, NetBeans, IntelliJ IDEA, VLC Media Player
* Ứng dụng giao dịch: Java là một trong những ngôn ngữ lập trình được sử dụng rộng rãi trong ngành Dịch vụ tài chính, đặc biệt là trong các hệ thống giao dịch điện tử. Java có khả năng xử lý các tác vụ có độ phức tạp cao, yêu cầu hiệu suất cao và an toàn cao. Ví dụ: LMAX, Murex, FlexTrade
* Ứng dụng J2ME: Java cũng có phiên bản thu gọn là Java Micro Edition (J2ME), được sử dụng để phát triển các ứng dụng cho các thiết bị di động và IoT (Internet of Things). J2ME cho phép lập trình viên tạo ra các ứng dụng nhỏ gọn, tiết kiệm bộ nhớ và pin cho các thiết bị có tài nguyên hạn chế. Ví dụ: Opera Mini, UC Browser Mini, Mobile Legends
* Lập trình nhúng: Java cũng được ứng dụng trong các thiết bị nhúng (Embedded system), là các thiết bị có khả năng tính toán và điều khiển một số chức năng cụ thể. Java cho phép lập trình viên viết mã một lần và chạy trên nhiều thiết bị nhúng khác nhau mà không cần thay đổi mã nguồn. Ví dụ: Các cảm biến, bộ điều khiển, thiết bị đo lường; các hệ thống giám sát môi trường, giao thông, sản xuất
* Công nghệ Big Data: Java cũng là một ngôn ngữ lập trình quan trọng trong lĩnh vực Big Data, là công nghệ xử lý và phân tích các dữ liệu lớn, phức tạp và đa dạng. Java cung cấp nhiều công cụ và thư viện để hỗ trợ lập trình viên trong việc xây dựng các ứng dụng Big Data hiệu quả và mạnh mẽ. Ví dụ: Hadoop, Spark, Kafka, Cassandra
* Hệ thống hiệu suất cao: Java cũng có thể được sử dụng để phát triển các hệ thống hiệu suất cao, là các hệ thống có khả năng xử lý các tác vụ có độ khó cao, yêu cầu tốc độ cao và độ tin cậy cao. Java có nhiều tính năng và công nghệ để tối ưu hóa hiệu suất của các hệ thống này, như đa luồng (multithreading), bộ nhớ đệm (caching), thu gom rác (garbage collection), v.v. Ví dụ: Netflix, Twitter, Amazon,

### 2.1.3 Ưu và nhược điểm của JavaScript

Java là một ngôn ngữ lập trình hướng đối tượng, dựa trên lớp, có khả năng chạy trên nhiều nền tảng khác nhau mà không cần thay đổi mã nguồn. Java có nhiều ưu điểm và nhược điểm mà bạn cần biết khi sử dụng. Dưới đây là một số điểm nổi bật:

Ưu điểm của Java:

* Java dễ học và đọc hơn C++, vì Java có ít cơ sở cấp thấp hơn và loại bỏ các thao tác với con trỏ, overload, goto và các cấu trúc như struct và union.
* Java độc lập phần cứng và hệ điều hành, có thể chạy trên nhiều môi trường khác nhau thông qua máy ảo Java (JVM), theo tiêu chí "Viết một lần, chạy khắp nơi" (Write Once, Run Anywhere).
* Java là ngôn ngữ thông dịch, mã Java được biên dịch thành bytecode và được thông dịch bởi JVM khi chạy, giúp tăng tính linh hoạt và bảo mật.
* Java có cơ chế thu gom rác tự động (garbage collection), giải phóng bộ nhớ cho các đối tượng không còn sử dụng, giảm thiểu rủi ro rò rỉ bộ nhớ và lỗi con trỏ.
* Java hỗ trợ đa luồng (multithreading), cho phép thực hiện nhiều tác vụ đồng thời trong một chương trình, tăng hiệu suất và khả năng phản hồi.
* Java có tính an toàn và bảo mật cao, kiểm tra kiểu dữ liệu tĩnh, quản lý quyền truy cập và mã hóa dữ liệu, ngăn chặn các cuộc tấn công từ bên ngoài.
* Java có nhiều thư viện và công nghệ hỗ trợ lập trình viên trong việc xây dựng các ứng dụng web, doanh nghiệp, di động, IoT, Big Data, giao dịch, hiệu suất cao, v.v. Ví dụ: Servlet, JSP, JSF, Spring, Hibernate, Struts, Android API, J2ME, Hadoop, Spark, Kafka, Cassandra, v.v.
* Java có nhiều IDE miễn phí và mạnh mẽ để viết và kiểm tra code, như Eclipse, NetBeans, IntelliJ IDEA, v.v. Các IDE này cung cấp các tính năng như gợi ý code, gỡ lỗi code, tự động hoàn thành code, v.v.Java có cộng đồng lập trình viên lớn và tích cực, sẵn sàng chia sẻ kiến thức và kinh nghiệm qua các diễn đàn, blog, video hay sách.

Nhược điểm của Java:

* Java sử dụng nhiều bộ nhớ hơn C++ hoặc C#, vì Java phải tạo ra các đối tượng cho mọi thứ và quản lý chúng qua JVM. Điều này có thể làm giảm hiệu suất của chương trình khi xử lý các tác vụ có yêu cầu bộ nhớ cao.
* Java không có sự tách biệt đặc điểm kỹ thuật khi triển khai. Điều này có nghĩa là mã Java phải được biên dịch lại cho mỗi nền tảng khác nhau mà nó được chạy. Điều này có thể gây ra sự không tương thích hoặc lỗi khi chuyển đổi giữa các nền tảng.
* Java không có số nguyên không dấu. Điều này có thể gây khó khăn khi làm việc với các dữ liệu nhị phân hoặc các thao tác bit. Ngoài ra, Java cũng không hỗ trợ các toán tử như ++, --, +=, -=, v.v. cho các kiểu dữ liệu nguyên thủy như byte, short, int và long.
* Java không có con trỏ, không có tham số ref hoặc out (thay thế con trỏ trong C#). Điều này có thể hạn chế khả năng của Java để tạo ra các cấu trúc dữ liệu chất lượng cao. Để trả về nhiều thứ từ một hàm Java, bạn có xu hướng phải trả về một mảng đối tượng hoặc tạo một lớp chứa các đối tượng và trả về một đối tượng của lớp đó.
* Java không cho phép người lập chỉ mục hoặc toán tử trên các đối tượng, ví dụ: nếu bạn sử dụng danh sách có thể thay đổi kích thước, v.v., bạn phải thực hiện list.get(index) thay vì list[index] và bạn không thể thực hiện someVector + someOtherVector.
* Java không có template, là một tính năng của C++ cho phép lập trình viên viết mã chung cho nhiều kiểu dữ liệu khác nhau. Thay vào đó, Java sử dụng generics, là một tính năng cho phép lập trình viên xác định các lớp và phương thức với các kiểu dữ liệu tham số hóa. Tuy nhiên, generics của Java có một số hạn chế so với template của C++, ví dụ: không hỗ trợ các kiểu nguyên thủy, không hỗ trợ các ràng buộc kiểu dữ liệu phức tạp, không hỗ trợ kế thừa đa hình (multiple inheritance), v.v.
* Java không có sự đa hình (polymorphism) của toán tử, là một tính năng của C++ cho phép lập trình viên định nghĩa lại cách hoạt động của các toán tử như +, -, \*, /, v.v. cho các kiểu dữ liệu do người dùng định nghĩa. Điều này có thể giúp mã nguồn trở nên ngắn gọn và rõ ràng hơn. Tuy nhiên, Java chỉ hỗ trợ sự đa hình của toán tử cho một số kiểu dữ liệu cơ bản và chuỗi.

## 2.2. Lý thuyết về các thuật toán

### 2.1.1 Lý thuyết về thuật toán Bellman Ford

Thuật toán Bellman-Ford là một thuật toán tính các đường đi ngắn nhất nguồn đơn trong một đồ thị có hướng có trọng số, trong đó một số cung có thể có trọng số âm. Thuật toán này được đặt theo tên của hai nhà toán học Richard Bellman và Lester Ford, Jr., người đã đề xuất và cải tiến nó vào những năm 1950 và 1960

Ý tưởng cơ bản của thuật toán Bellman-Ford là lặp lại quá trình duyệt qua tất cả các cung của đồ thị và cập nhật nhãn khoảng cách (distance label) và đỉnh liền trước (predecessor) của các đỉnh nếu có thể. Nhãn khoảng cách của một đỉnh là độ dài của một đường đi ngắn nhất từ đỉnh nguồn tới đỉnh đó, còn đỉnh liền trước của một đỉnh là đỉnh kề với nó trên đường đi ngắn nhất đó. Quá trình lặp lại nhiều nhất |V| - 1 lần, trong đó V là tập hợp các đỉnh của đồ thị. Nếu sau khi lặp lại, vẫn có thể cập nhật được nhãn khoảng cách của một số đỉnh, thì có nghĩa là đồ thị chứa chu trình âm (negative cycle), tức là một chu trình mà tổng trọng số của các cung trên nó là âm. Khi đó, bài toán tìm đường đi ngắn nhất không có nghĩa, vì ta có thể lặp đi lặp lại chu trình âm để giảm dần khoảng cách

Thuật toán Bellman-Ford có thời gian chạy là O(|V|\*|E|), trong đó E là tập hợp các cung của đồ thị. Thuật toán này có thể phát hiện được sự tồn tại của chu trình âm trong đồ thị, nếu có. Thuật toán này có ưu điểm là dễ hiểu và cài đặt, không yêu cầu trọng số của các cung phải không âm, và có thể áp dụng cho các bài toán biến thể như tìm chu trình âm với trọng số nhỏ nhất, tìm chu trình âm với số cung ít nhất, v.v. Tuy nhiên, thuật toán này có nhược điểm là chạy chậm hơn so với thuật toán Dijkstra, một thuật toán khác giải quyết cùng bài toán này nhưng yêu cầu trọng số của các cung phải không âm. Thuật toán Dijkstra có thời gian chạy là O(|E| + |V|log|V|) khi sử dụng hàng đợi ưu tiên (priority queue).

**Đặt vấn đề**

Cho đồ thị như sau:

A diagram of a triangle with numbers and lines

Description automatically generated

Tìm đường đi ngắn nhất giữa đỉnh 1 với các đỉnh còn lại trong đồ thị.

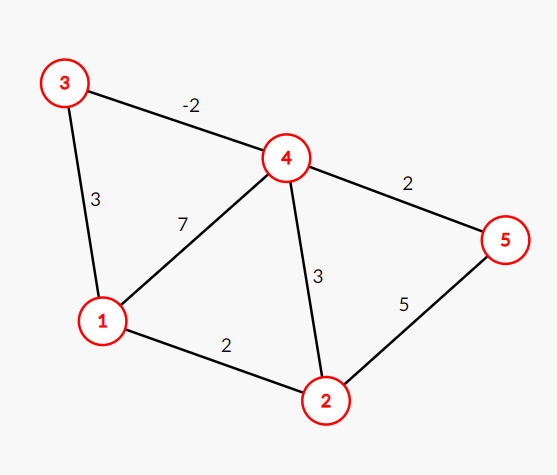
Nếu đồ thị có cạnh trọng số âm, cách cài đặt thuật toán tìm đường đi ngắn nhất Dijkstra cổ điển sẽ thất bại. Cách cài đặt cải tiến sẽ giúp cho thuật toán Dijkstra khắc phục vấn đề này. Tuy nhiên khi đồ thị có chu trình âm, cách cài đặt cải tiến này cũng không sử dụng được do thuật toán sẽ thực hiện vòng lặp vô hạn.

Thuật toán Bellman Ford sẽ giải quyết vấn đề này. Ý tưởng chính rất đơn giản đó là

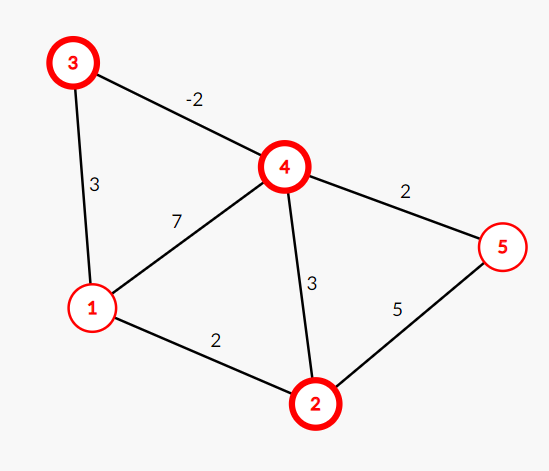
**Mô tả thuật toán**

Ta xét ví dụ với đồ thị có hướng trên (giả định các đường đi là một chiều, chỉ đi từ đỉnh có số thứ tự thấp hơn tới đỉnh có số thứ tự cao hơn, số có màu đỏ cạnh mỗi đỉnh là độ dài đường đi ngắn nhất từ gốc tới đỉnh đó, và đỉnh gốc là đỉnh 1).

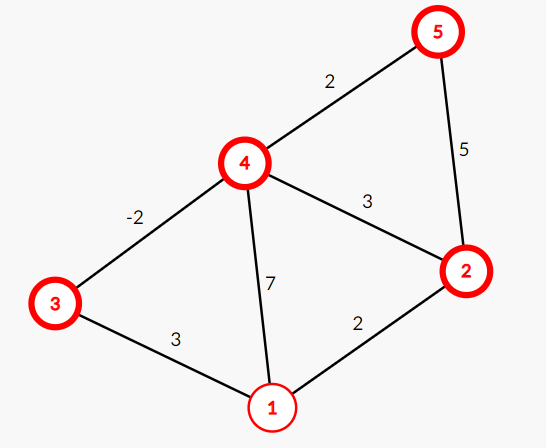
**Khởi tạo:**



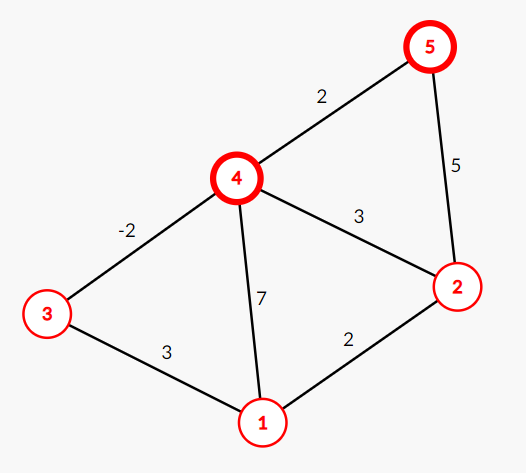
Thực hiện lần duyệt đầu tiên, ta cập nhật được đường đi ngắn nhất thông qua các cạnh (1, 2); (1, 3); (1, 4):



Tương tự với lần duyệt thứ 2, cạnh (2, 5) và (3, 4) là các cạnh tối ưu:



Với lần duyệt thứ 3, chỉ có cạnh (4, 5) cải tiến đường đi tối ưu:



Tới lần duyệt thứ 4, ta thấy không còn cạnh nào có thể tối ưu hóa bất kỳ đường đi ngắn nhất nào nữa. Tới đây, ta hoàn toàn có thể dừng duyệt (vì chắc chắn việc không còn cạnh có thể tối ưu cũng đồng nghĩa với việc không có chu trình âm trong đồ thị).

### 2.2.2 Thuật toán duyệt theo chiều rộng (Breadth-first-search-BFS)

Thuật toán duyệt theo chiều rộng (Breadth-first search - BFS) là một thuật toán để duyệt hoặc tìm kiếm trong một đồ thị hoặc cây. Thuật toán này bắt đầu từ một đỉnh nguồn và lần lượt thăm các đỉnh kề với nó theo thứ tự từ gần đến xa. Sau đó, với mỗi đỉnh đã thăm, thuật toán lại lặp lại quá trình trên cho các đỉnh kề với nó mà chưa được thăm. Quá trình này tiếp tục cho đến khi tất cả các đỉnh của đồ thị hoặc cây được thăm hoặc không còn đỉnh nào kề với các đỉnh đã thăm

Thuật toán BFS có nhiều ứng dụng trong lý thuyết đồ thị và lập trình, ví dụ như:

* Tìm đường đi ngắn nhất từ một đỉnh nguồn tới một đỉnh đích hoặc tới tất cả các đỉnh khác trong một đồ thị không có trọng số hoặc có trọng số không âm. Thuật toán BFS luôn tìm ra được đường đi ngắn nhất nếu tồn tại, vì nó duyệt các đỉnh theo thứ tự tăng dần của khoảng cách từ nguồn
* Tìm các thành phần liên thông của một đồ thị vô hướng hoặc các thành phần liên thông mạnh của một đồ thị có hướng. Thuật toán BFS có thể duyệt qua tất cả các đỉnh thuộc cùng một thành phần liên thông bằng cách bắt đầu từ một đỉnh bất kỳ của thành phần đó. Bằng cách lặp lại quá trình này cho các đỉnh chưa được duyệt, ta có thể xác định được số lượng và thành phần của các thành phần liên thông.
* Kiểm tra tính hai phía (bipartite) của một đồ thị vô hướng. Một đồ thị vô hướng được gọi là hai phía nếu ta có thể chia tập hợp các đỉnh của nó thành hai tập con sao cho không có hai đỉnh thuộc cùng một tập con kề nhau. Thuật toán BFS có thể kiểm tra tính hai phía của một đồ thị bằng cách gán nhãn cho các đỉnh khi duyệt, sao cho nhãn của mỗi đỉnh khác với nhãn của các đỉnh kề với nó. Nếu không thể gán nhãn được như vậy, tức là đồ thị không hai phía

**Thuật toán**

Thuật toán sử dụng một cấu trúc dữ liệu hàng đợi (queue) để chứa các đỉnh sẽ được duyệt theo thứ tự ưu tiên chiều rộng.

Bước 1: Khởi tạo

* Các đỉnh đều ở trạng thái chưa được đánh dấu. Ngoại trừ đỉnh nguồn 𝑠 đã được đánh dấu.
* Một hàng đợi ban đầu chỉ chứa 1 phần tử là 𝑠.

Bước 2: Lặp lại các bước sau cho đến khi hàng đợi rỗng:

* Lấy đỉnh 𝑢 ra khỏi hàng đợi.
* Xét tất cả những đỉnh 𝑣 kề với 𝑢 mà chưa được đánh dấu, với mỗi đỉnh 𝑣 đó:
* Đánh dấu 𝑣 đã thăm.
* Lưu lại vết đường đi từ 𝑢 đến 𝑣
* Đẩy 𝑣 vào trong hàng đợi (đỉnh 𝑣 sẽ chờ được duyệt tại những bước sau).

Bước 3: Truy vết tìm đường đi.

### 2.2.3 Thuật toán duyệt theo chiều sâu (Depth First Search (DFS))

Thuật toán DFS có nhiều ứng dụng trong lý thuyết đồ thị và lập trình, ví dụ như:

* Tìm kiếm theo chiều sâu trên cây: Thuật toán DFS có thể duyệt qua tất cả các nút của một cây theo một trong ba cách: duyệt tiền thứ tự (preorder), duyệt trung thứ tự (inorder) hoặc duyệt hậu thứ tự (postorder). Mỗi cách duyệt có những ứng dụng khác nhau, ví dụ: duyệt tiền thứ tự có thể dùng để sao chép một cây, duyệt trung thứ tự có thể dùng để sắp xếp các giá trị trong một cây nhị phân tìm kiếm, duyệt hậu thứ tự có thể dùng để giải phóng bộ nhớ cho các nút của cây
* Tìm kiếm theo chiều sâu trên đồ thị: Thuật toán DFS có thể duyệt qua tất cả các đỉnh của một đồ thị liên thông bằng cách bắt đầu từ một đỉnh bất kỳ và lặp lại quá trình cho các đỉnh chưa được duyệt. Thuật toán này có thể dùng để tìm ra các thành phần liên thông của một đồ thị vô hướng hoặc các thành phần liên thông mạnh của một đồ thị có hướng. Ngoài ra, thuật toán này còn có thể dùng để kiểm tra tính hai phía của một đồ thị vô hướng, tìm ra chu trình Hamilton của một đồ thị, hay xác định trật tự tô-pô của một đồ thị có hướng không chu trình.
* Tìm kiếm theo chiều sâu giới hạn (Depth Limited Search - DLS): Thuật toán DLS là một biến thể của thuật toán DFS, trong đó ta chỉ duyệt tới một chiều sâu giới hạn nào đó. Thuật toán này có ưu điểm là tránh được việc duyệt quá sâu vào những nhánh không có lời giải, nhưng có nhược điểm là có thể bỏ qua những lời giải nằm ở chiều sâu lớn hơn giới hạn. Thuật toán này có thể dùng để giải quyết các bài toán tìm kiếm không gian trạng thái, ví dụ: bài toán xếp bát, bài toán xếp quân cờ, bài toán xếp số 8, v.v

**Giải thuật**

* Bước 1: Chọn một đỉnh bất kỳ làm đỉnh nguồn và đánh dấu nó là đã duyệt. Đưa đỉnh nguồn vào một ngăn xếp (stack).
* Bước 2: Lấy ra một đỉnh từ ngăn xếp và thăm các đỉnh kề với nó mà chưa được duyệt. Đánh dấu các đỉnh này là đã duyệt và đưa chúng vào ngăn xếp theo thứ tự từ xa đến gần.
* Bước 3: Lặp lại bước 2 cho đến khi ngăn xếp rỗng hoặc tìm được đỉnh cần tìm (nếu có).
* Bước 4: Trả về kết quả là danh sách các đỉnh đã duyệt theo thứ tự duyệt, hoặc đường đi ngắn nhất từ đỉnh nguồn tới đỉnh cần tìm (nếu có).

# 3. CHƯƠNG III: ỨNG DỤNG

## 3.1. Phân tích yêu cầu bài toán

Để phân tích yêu cầu bài toán tìm đường đi ngắn nhất, ta cần xác định các yếu tố sau:

* Đầu vào: Là dữ liệu về đồ thị, bao gồm số đỉnh, số cung, trọng số của các cung và đỉnh nguồn. Đầu vào có thể được nhập trực tiếp lên màn hình bằng cách ấn chuột trái
* Đầu ra: Là kết quả của bài toán, bao gồm nhãn khoảng cách và đường đi ngắn nhất từ đỉnh nguồn tới các đỉnh khác. Đầu ra có thể được in ra màn hình.
* Ràng buộc: Là các điều kiện giới hạn cho bài toán, ví dụ như trọng số của các cung phải không âm, không tồn tại chu trình âm, không tồn tại cung nối một đỉnh với chính nó, v.v. Ràng buộc có thể ảnh hưởng đến việc chọn thuật toán để giải quyết bài toán.
* Mục tiêu: hiểu được nguyên lý hoạt động và cách cài đặt của các thuật toán tìm đường đi ngắn nhất, biết cách sử dụng ngôn ngữ lập trình để viết code cho các thuật toán này, có khả năng phân tích và kiểm tra tính đúng đắn, hiệu quả và ổn định của code, có kỹ năng trình bày báo cáo một cách rõ ràng, logic và súc tích.

## 3.2. Xây dựng các cấu trúc dữ liệu cần thiết

### 3.2.1. Class Algorithms.java

Class Algorithms.java là một lớp để thực hiện các thuật toán tìm đường đi ngắn nhất và duyệt đồ thị trên một đối tượng của lớp Graph. Lớp này có các thuộc tính và phương thức sau:

Thuộc tính:

* ArrayList<Boolean> chuaXet: một mảng động để lưu trữ các giá trị boolean, biểu diễn trạng thái của các đỉnh trong đồ thị khi duyệt hoặc tìm đường đi. Nếu chuaXet.get(i) là true, nghĩa là đỉnh i chưa được xét, ngược lại là đã được xét.
* int distances[]: một mảng để lưu trữ khoảng cách từ đỉnh nguồn tới các đỉnh khác trong đồ thị khi áp dụng thuật toán Bellman-Ford. Nếu distances[i] là MAX\_VALUE, nghĩa là không có đường đi từ đỉnh nguồn tới đỉnh i, ngược lại là có đường đi với khoảng cách bằng giá trị của distances[i].
* private int numberofvertices: một biến để lưu trữ số lượng đỉnh của đồ thị.
* public static final int MAX\_VALUE = 9999: một hằng số để biểu diễn giá trị vô cùng lớn, dùng cho việc khởi tạo mảng distances.
* ArrayList<ArrayList<Integer>> mtk: một mảng hai chiều để lưu trữ ma trận kề của đồ thị, biểu diễn sự kề nhau giữa các đỉnh bằng số 0 hoặc 1.
* ArrayList<ArrayList<Integer>> danhSachKe: một mảng hai chiều để lưu trữ danh sách kề của đồ thị, biểu diễn các đỉnh kề với mỗi đỉnh bằng các chỉ số của chúng.
* Vertex vertex: một đối tượng của lớp Vertex, là một lớp để biểu diễn các đỉnh của đồ thị.

Phương thức:

* Algorithms(Vertex v, ArrayList<ArrayList<Integer>> mtk): hàm khởi tạo cho lớp Algorithms, nhận vào một đối tượng của lớp Vertex và một mảng hai chiều là ma trận kề của đồ thị. Hàm này sẽ gán giá trị cho các thuộc tính vertex, numberofvertices và mtk. Hàm này cũng sẽ khởi tạo các mảng distances, danhSachKe và chuaXet với kích thước phù hợp và giá trị ban đầu.
* public ArrayList<Boolean> getChuaXet(): hàm để trả về giá trị của thuộc tính chuaXet.
* public int[] getDistances(): hàm để trả về giá trị của thuộc tính distances.
* public boolean ke(ArrayList<ArrayList<Integer>> mtk, int x, int y): hàm để kiểm tra xem hai đỉnh x và y có kề nhau trong ma trận kề hay không, nhận vào một mảng hai chiều là ma trận kề và hai số nguyên là chỉ số của hai đỉnh. Hàm này sẽ trả về true nếu có kề nhau và false nếu không.
* public String DFS(int n): hàm để duyệt đồ thị theo chiều sâu (Depth-First Search) bắt đầu từ đỉnh n, nhận vào một số nguyên là chỉ số của đỉnh n. Hàm này sử dụng cấu trúc dữ liệu ngăn xếp (stack) để lưu trữ các đỉnh chưa được duyệt. Hàm này sẽ cập nhật lại giá trị của mảng chuaXet khi duyệt qua các đỉnh. Hàm này sẽ trả về một chuỗi là thứ tự duyệt các đỉnh của đồ thị.
* public String BFS(int n): hàm để duyệt đồ thị theo chiều rộng (Breadth-First Search) bắt đầu từ đỉnh n, nhận vào một số nguyên là chỉ số của đỉnh n. Hàm này sử dụng cấu trúc dữ liệu hàng đợi (queue) để lưu trữ các đỉnh chưa được duyệt. Hàm này sẽ cập nhật lại giá trị của mảng chuaXet khi duyệt qua các đỉnh. Hàm này sẽ trả về một chuỗi là thứ tự duyệt các đỉnh của đồ thị.
* public boolean BellmanFordEvaluation(int source, ArrayList<Edge> edges): hàm để tìm đường đi ngắn nhất từ một đỉnh nguồn tới các đỉnh khác trong đồ thị bằng thuật toán Bellman-Ford, nhận vào một số nguyên là chỉ số của đỉnh nguồn và một mảng động là danh sách cạnh của đồ thị. Hàm này sẽ cập nhật lại giá trị của mảng distances khi tìm được đường đi ngắn hơn. Hàm này sẽ kiểm tra xem có tồn tại chu trình âm trong đồ thị hay không. Hàm này sẽ trả về true nếu có chu trình âm và false nếu không.

### 3.2.2. Class appFrame.java

Class appFrame.java là một lớp để tạo ra một khung ứng dụng (application frame) cho chương trình vẽ đồ thị và thực hiện các thuật toán tìm đường đi ngắn nhất. Lớp này có các thuộc tính và phương thức sau:

Thuộc tính:

* private MyPanel myPanel: một đối tượng của lớp MyPanel, là một lớp kế thừa từ lớp JPanel để tạo ra một bảng vẽ (drawing panel) cho chương trình.

Phương thức:

* public appFrame(): hàm khởi tạo cho lớp appFrame, thêm đối tượng myPanel vào khung ứng dụng, đặt tiêu đề, kích thước, vị trí và các thuộc tính khác cho khung ứng dụng.
* public static void main(String[] args): hàm chính để chạy chương trình, tạo ra một đối tượng của lớp appFrame.

### 3.2.3. Class DrawPanel.java

Class DrawPanel.java là một lớp để tạo ra một bảng vẽ (drawing panel) cho chương trình vẽ đồ thị và thực hiện các thuật toán tìm đường đi ngắn nhất. Lớp này có các thuộc tính và phương thức sau:

Thuộc tính:

* private static final long serialVersionUID = 1L: một số để xác định phiên bản của lớp, dùng cho việc tuần tự hóa (serialization).
* boolean startUnDir = false: một biến để kiểm tra xem có đang vẽ đồ thị vô hướng hay không.
* boolean startDir = false: một biến để kiểm tra xem có đang vẽ đồ thị có hướng hay không.
* final int unit = 3: một hằng số để xác định đơn vị đo lường cho bảng vẽ.
* Graph graph: một đối tượng của lớp Graph, là một lớp để lưu trữ và xử lý các thông tin về đồ thị, như danh sách đỉnh, danh sách cạnh, ma trận kề, v.v.
* Vertex selected1, selected2: hai đối tượng của lớp Vertex, là một lớp để biểu diễn các đỉnh của đồ thị. Hai biến này dùng để lưu trữ hai đỉnh được chọn khi vẽ cạnh.
* int clickX, clickY, pressX, pressY, dx, dy, midX, midY: các biến để lưu trữ tọa độ của các sự kiện chuột như click hoặc kéo thả, và khoảng cách giữa hai điểm.
* String trongso = null: một biến để lưu trữ trọng số của cạnh khi nhập vào từ bàn phím.
* final int bankinh = 15: một hằng số để xác định bán kính của các hình tròn biểu diễn các đỉnh.
* Line2D line2d: một đối tượng của lớp Line2D, là một lớp để biểu diễn các đường thẳng trong không gian hai chiều. Đối tượng này dùng để vẽ cạnh giữa hai đỉnh.
* int vertexAmount: một biến để lưu trữ số lượng đỉnh của đồ thị.

Phương thức:

* DrawPanel(): hàm khởi tạo cho lớp DrawPanel, khởi tạo đối tượng graph, vertexAmount và thiết lập các thuộc tính cho bảng vẽ như con trỏ chuột, màu nền, kích thước và bộ lắng nghe sự kiện chuột.
* public int getVertexAmount(): hàm để trả về giá trị của biến vertexAmount.
* public void setVertexAmount(int vertexAmount): hàm để gán giá trị cho biến vertexAmount.
* MouseAdapter mouse = new MouseAdapter(): một đối tượng của lớp MouseAdapter, là một lớp để xử lý các sự kiện chuột trên bảng vẽ. Đối tượng này có các phương thức như mousePressed, mouseDragged và mouseClicked để phản ứng khi người dùng nhấn, kéo hoặc nhấp chuột trên bảng vẽ. Các phương thức này dùng để thêm hoặc di chuyển các đỉnh và cạnh của đồ thị theo ý muốn của người dùng.

### 3.2.4. Class Edge.java

Class Edge.java là một lớp để biểu diễn các cạnh của đồ thị, có các thuộc tính và phương thức sau:

Thuộc tính:

* Vertex diemdau1: một đối tượng của lớp Vertex, là đỉnh đầu của cạnh.
* Vertex diemdau2: một đối tượng của lớp Vertex, là đỉnh cuối của cạnh.
* Line2D line2d: một đối tượng của lớp Line2D, là một lớp để biểu diễn các đường thẳng trong không gian hai chiều. Đối tượng này dùng để vẽ cạnh trên bảng vẽ.
* int centerEdgeX: một biến để lưu trữ tọa độ x của điểm giữa của cạnh.
* int centerEdgeY: một biến để lưu trữ tọa độ y của điểm giữa của cạnh.
* String weight: một biến để lưu trữ trọng số của cạnh.

Phương thức:

* public Edge(Vertex diemdau1, Vertex diemdau2, Line2D line2d, int centerEdgeX, int centerEdgeY, String weight): hàm khởi tạo cho lớp Edge, nhận vào các tham số tương ứng và gán cho các thuộc tính.
* public Vertex getDiemdau1(): hàm để trả về giá trị của thuộc tính diemdau1.
* public void setDiemdau1(Vertex diemdau1): hàm để gán giá trị cho thuộc tính diemdau1.
* public Vertex getDiemdau2(): hàm để trả về giá trị của thuộc tính diemdau2.
* public void setDiemdau2(Vertex diemdau2): hàm để gán giá trị cho thuộc tính diemdau2.
* public void setLine2D(Line2D line2d): hàm để gán giá trị cho thuộc tính line2d.
* public Line2D getLine2D(): hàm để trả về giá trị của thuộc tính line2d.
* public int getCenterEdgeX(): hàm để trả về giá trị của thuộc tính centerEdgeX.
* public void setCenterEdgeX(int centerEdgeX): hàm để gán giá trị cho thuộc tính centerEdgeX.
* public int getCenterEdgeY(): hàm để trả về giá trị của thuộc tính centerEdgeY.
* public void setCenterEdgeY(int centerEdgeY): hàm để gán giá trị cho thuộc tính centerEdgeY.
* public String getWeight(): hàm để trả về giá trị của thuộc tính weight.
* public void setWeight(String weight): hàm để gán giá trị cho thuộc tính weight.

### 3.2.5. Class Graph.java

Class Graph.java là một lớp để lưu trữ và xử lý các thông tin về đồ thị, như danh sách đỉnh, danh sách cạnh, ma trận kề, v.v. Lớp này có các thuộc tính và phương thức sau:

Thuộc tính:

* ArrayList<Vertex> danhSachDinh: một mảng động để lưu trữ các đối tượng của lớp Vertex, là các đỉnh của đồ thị.
* ArrayList<Edge> danhSachCanh: một mảng động để lưu trữ các đối tượng của lớp Edge, là các cạnh của đồ thị.
* ArrayList<ArrayList<Integer>> mtk: một mảng hai chiều để lưu trữ ma trận kề của đồ thị, biểu diễn sự kề nhau giữa các đỉnh bằng số 0 hoặc 1.

Phương thức:

* public Graph(): hàm khởi tạo cho lớp Graph, khởi tạo các mảng động rỗng.
* public ArrayList<Vertex> getDanhSachDinh(): hàm để trả về giá trị của thuộc tính danhSachDinh.
* public void setDanhSachDinh(ArrayList<Vertex> danhSachDinh): hàm để gán giá trị cho thuộc tính danhSachDinh.
* public ArrayList<Edge> getDanhSachCanh(): hàm để trả về giá trị của thuộc tính danhSachCanh.
* public void setDanhSachCanh(ArrayList<Edge> danhSachCanh): hàm để gán giá trị cho thuộc tính danhSachCanh.
* public ArrayList<ArrayList<Integer>> getMtk(): hàm để trả về giá trị của thuộc tính mtk.
* public void setMtk(ArrayList<ArrayList<Integer>> mtk): hàm để gán giá trị cho thuộc tính mtk.
* public void themDinh(Ellipse2D el): hàm để thêm một đỉnh mới vào đồ thị, nhận vào một đối tượng của lớp Ellipse2D, là một lớp để biểu diễn các hình tròn trong không gian hai chiều. Hàm này sẽ cập nhật ma trận kề và danh sách đỉnh của đồ thị.
* public boolean themCanhVoHuong(Vertex diemdau1, Vertex diemdau2, Line2D line2d, String weight): hàm để thêm một cạnh vô hướng mới vào đồ thị, nhận vào hai đối tượng của lớp Vertex là hai đầu của cạnh, một đối tượng của lớp Line2D là đường thẳng biểu diễn cạnh và một chuỗi là trọng số của cạnh. Hàm này sẽ kiểm tra xem có thể thêm cạnh hay không, nếu có thì cập nhật ma trận kề, danh sách cạnh và danh sách kề của các đỉnh. Hàm này sẽ trả về true nếu thêm thành công và false nếu không.
* public boolean themCanhCoHuong(Vertex diemdau1, Vertex diemdau2, Line2D line2d, String weight): hàm để thêm một cạnh có hướng mới vào đồ thị, nhận vào hai đối tượng của lớp Vertex là hai đầu của cạnh, một đối tượng của lớp Line2D là đường thẳng biểu diễn cạnh và một chuỗi là trọng số của cạnh. Hàm này sẽ kiểm tra xem có thể thêm cạnh hay không, nếu có thì cập nhật ma trận kề, danh sách cạnh và danh sách kề của các đỉnh. Hàm này sẽ trả về true nếu thêm thành công và false nếu không.
* public void resetEl(Ellipse2D el, int index): hàm để cập nhật lại hình tròn biểu diễn đỉnh khi di chuyển đỉnh, nhận vào một đối tượng của lớp Ellipse2D là hình tròn mới và một số nguyên là chỉ số của đỉnh trong danh sách đỉnh. Hàm này sẽ gán lại giá trị của thuộc tính el cho đỉnh tương ứng.
* public void resetEdge(Line2D line2d, int index, int cenX, int cenY): hàm để cập nhật lại đường thẳng biểu diễn cạnh khi di chuyển đỉnh, nhận vào một đối tượng của lớp Line2D là đường thẳng mới, một số nguyên là chỉ số của cạnh trong danh sách cạnh và hai số nguyên là tọa độ mới của điểm giữa của cạnh. Hàm này sẽ gán lại giá trị của các thuộc tính line2d, centerEdgeX và centerEdgeY cho cạnh tương ứng.
* public Vertex isVertex(int nameOtherV): hàm để kiểm tra xem có tồn tại một đỉnh có tên là nameOtherV trong đồ thị hay không, nhận vào một số nguyên là tên của đỉnh. Hàm này sẽ trả về đối tượng Vertex nếu có và null nếu không.

### 3.2.6. Class MyPanel.java

Class MyPanel.java là một lớp để tạo ra một bảng điều khiển (control panel) cho chương trình vẽ đồ thị và thực hiện các thuật toán tìm đường đi ngắn nhất. Lớp này có các thuộc tính và phương thức sau:

Thuộc tính:

* private static final long serialVersionUID = 1L: một số để xác định phiên bản của lớp, dùng cho việc tuần tự hóa (serialization).
* JPanel topPanel, centerPanel, bottomPanel, topFJPanel, topSJPanel, topSJPanel1, topSJPanel2, topTJPanel, centerChucNangJPanel, centerChucNangJPanelTop, centerChucNangJPanelBottom, centerChucNangJPanelCenter: các đối tượng của lớp JPanel, là các lớp để tạo ra các bảng con trong bảng điều khiển. Các bảng con này được sắp xếp theo hướng Bắc (top), Trung tâm (center) và Nam (bottom) của bảng điều khiển. Các bảng con này có các chức năng khác nhau như chọn loại đồ thị, tìm đường đi ngắn nhất, duyệt đồ thị, thêm hoặc xóa đỉnh và cạnh, in ma trận kề, v.v.
* DrawPanel centerDrawJPanel: một đối tượng của lớp DrawPanel, là một lớp để tạo ra một bảng vẽ cho chương trình. Đối tượng này được đặt ở vị trí Trung tâm của bảng điều khiển.
* JRadioButton r1, r2: hai đối tượng của lớp JRadioButton, là các nút lựa chọn để chọn loại đồ thị là vô hướng hay có hướng. Hai nút này được đặt ở vị trí Bắc cùng trái của bảng điều khiển.
* JButton duongDiButton, bfsBtn, dfsBtn, addEdgeBtn, newBtn, deleteVertexBtn, printMatrixButton, deleteEdgeBtn: các đối tượng của lớp JButton, là các nút nhấn để thực hiện các chức năng khác nhau như tìm đường đi ngắn nhất bằng thuật toán Bellman-Ford, duyệt đồ thị theo chiều sâu (DFS) hoặc chiều rộng (BFS), thêm hoặc xóa đỉnh và cạnh, in ma trận kề, v.v. Các nút này được đặt ở các vị trí khác nhau trong bảng điều khiển.
* JLabel dinhBatDauLabel, dauJLabel, cuoiJLabel, trongJLabel, tenLabel, sourceLabel: các đối tượng của lớp JLabel, là các nhãn để hiển thị các thông tin cần thiết cho người dùng nhập vào hoặc xem kết quả. Các nhãn này có nội dung như “Begin Vertex:”, “First Point:”, “Final Point:”, “Weight:”, “Name Vertex:”, “Source:”, v.v. Các nhãn này được đặt ở các vị trí khác nhau trong bảng điều khiển.
* JTextField dauField, cuoiField, trongField, tenField, duyetDoThiField, soureField: các đối tượng của lớp JTextField, là các ô nhập liệu để người dùng nhập vào các thông tin cần thiết cho chương trình xử lý hoặc hiển thị kết quả. Các ô nhập liệu này có nội dung như tên hoặc chỉ số của các đỉnh và cạnh trong đồ thị. Các ô nhập liệu này được đặt ở các vị trí khác nhau trong bảng điều khiển.
* JTextArea textAreaLeft, textAreaRight, textAreaCenter: các đối tượng của lớp JTextArea, là các vùng văn bản để hiển thị các thông tin hoặc kết quả cho người dùng xem. Các vùng văn bản này có nội dung như ma trận kề của đồ thị, đường đi ngắn nhất từ đỉnh nguồn tới các đỉnh khác, thứ tự duyệt các đỉnh của đồ thị, v.v. Các vùng văn bản này được đặt ở các vị trí khác nhau trong bảng điều khiển.
* JScrollPane drawJScrollPane, leftText, rightText, centerText: các đối tượng của lớp JScrollPane, là các thanh cuộn để cho phép người dùng xem toàn bộ nội dung của các bảng con hoặc vùng văn bản khi chúng vượt quá kích thước hiển thị. Các thanh cuộn này được đặt ở các vị trí khác nhau trong bảng điều khiển.

Phương thức:

* MyPanel(): hàm khởi tạo cho lớp MyPanel, khởi tạo các đối tượng của các lớp con và thiết lập các thuộc tính cho chúng như kích thước, màu sắc, biên, bố cục, v.v. Hàm này cũng thêm các đối tượng vào bảng điều khiển theo hướng Bắc (top), Trung tâm (center) và Nam (bottom) và đăng ký các bộ lắng nghe sự kiện cho các nút nhấn và nút lựa chọn.
* public void actionPerformed(ActionEvent e): hàm để xử lý các sự kiện khi người dùng nhấn vào các nút nhấn hoặc nút lựa chọn. Hàm này sẽ kiểm tra nguồn của sự kiện và thực hiện các hành động tương ứng như chuyển đổi loại đồ thị, gọi các phương thức của lớp DrawPanel để vẽ đồ thị và áp dụng các thuật toán tìm đường đi ngắn nhất hoặc duyệt đồ thị, hiển thị ma trận kề hoặc thông báo lỗi, v.v.

### 3.2.7. Class Vertex.java

Class Vertex.java là một lớp để biểu diễn các đỉnh của đồ thị, có các thuộc tính và phương thức sau:

Thuộc tính:

* Integer ten: một đối tượng của lớp Integer, là tên của đỉnh.
* Integer index: một đối tượng của lớp Integer, là chỉ số của đỉnh trong danh sách đỉnh.
* ArrayList<Vertex> danhSachKe: một mảng động để lưu trữ các đối tượng của lớp Vertex, là các đỉnh kề với đỉnh hiện tại.
* Ellipse2D el: một đối tượng của lớp Ellipse2D, là một lớp để biểu diễn các hình tròn trong không gian hai chiều. Đối tượng này dùng để vẽ hình tròn biểu diễn đỉnh trên bảng vẽ.

Phương thức:

* public Vertex(Integer ten, Integer index, ArrayList<Vertex> danhSachKe, Ellipse2D el): hàm khởi tạo cho lớp Vertex, nhận vào các tham số tương ứng và gán cho các thuộc tính.
* public Integer getTen(): hàm để trả về giá trị của thuộc tính ten.
* public void setTen(Integer ten): hàm để gán giá trị cho thuộc tính ten.
* public Integer getIndex(): hàm để trả về giá trị của thuộc tính index.
* public void setIndex(Integer index): hàm để gán giá trị cho thuộc tính index.
* public ArrayList<Vertex> getDanhSachKe(): hàm để trả về giá trị của thuộc tính danhSachKe.
* public void setDanhSachKe(ArrayList<Vertex> danhSachKe): hàm để gán giá trị cho thuộc tính danhSachKe.
* public Ellipse2D getEl(): hàm để trả về giá trị của thuộc tính el.
* public void setEl(Ellipse2D el): hàm để gán giá trị cho thuộc tính el.
* public boolean isVertexName(int nameOtherV): hàm để kiểm tra xem tên của đỉnh hiện tại có bằng với một số nguyên nameOtherV hay không, nhận vào một số nguyên là tên của một đỉnh khác. Hàm này sẽ trả về true nếu bằng và false nếu không.

## 3.3. Xây dựng lưu đồ giải thuật

A diagram of a computer program

Description automatically generated

Sự khác biệt giữa hai thuật toán là cách thêm các nút mới vào tập biên (fringe set). Trong DFS, các nút mới được thêm vào đầu của tập biên, theo nguyên tắc xếp chồng (stack), trong khi trong BFS, các nút mới được thêm vào cuối của tập biên, theo nguyên tắc hàng đợi (queue).

# 4. CHƯƠNG IV: THIẾT KẾ - CÀI ĐẶT

## 4.1. Thiết kế giao diện

### 4.1.1. Khai quát giao diện và các nút chức năng

Giao diện gồm có:

Một bảng điều khiển (control panel) ở phía trên, chia thành ba phần:

* Phần bên trái cho phép người dùng chọn loại đồ thị là vô hướng hay có hướng bằng hai nút lựa chọn (radio button).
* Phần ở giữa cho phép người dùng tìm đường đi ngắn nhất từ một đỉnh nguồn tới các đỉnh khác bằng thuật toán Bellman-Ford, bằng cách nhập vào tên hoặc chỉ số của đỉnh nguồn và nhấn nút “Bellman-Ford”.
* Phần bên phải cho phép người dùng duyệt đồ thị theo chiều sâu (DFS) hoặc chiều rộng (BFS) bắt đầu từ một đỉnh nào đó, bằng cách nhập vào tên hoặc chỉ số của đỉnh đó và nhấn nút “DFS” hoặc “BFS”.
* Một bảng vẽ (draw panel) ở phía trung tâm, cho phép người dùng vẽ các đỉnh và cạnh của đồ thị bằng cách kéo thả chuột. Bảng vẽ có kích thước 10x10 ô vuông, mỗi ô vuông biểu diễn một nút trong ma trận kề của đồ thị. Người dùng có thể di chuyển các đỉnh và cạnh trên bảng vẽ bằng cách kéo thả chuột.

Một bảng chức năng (function panel) ở phía dưới, chia thành ba phần:

* Phần bên trái cho phép người dùng thêm hoặc xóa các đỉnh và cạnh của đồ thị, bằng cách nhập vào tên hoặc chỉ số của các đỉnh và trọng số của các cạnh, và nhấn nút “Add Edge” hoặc “Delete Edge”. Người dùng cũng có thể xóa toàn bộ đồ thị và vẽ lại từ đầu bằng nút “New”.
* Phần ở giữa cho phép người dùng xem ma trận kề của đồ thị, biểu diễn sự kề nhau giữa các đỉnh bằng số 0 hoặc 1. Người dùng có thể cập nhật ma trận kề khi thay đổi đồ thị trên bảng vẽ bằng nút “Print Matrix”.
* Phần bên phải cho phép người dùng xem kết quả của các thuật toán tìm đường đi ngắn nhất hoặc duyệt đồ thị, hiển thị khoảng cách từ đỉnh nguồn tới các đỉnh khác hoặc thứ tự duyệt các đỉnh của đồ thị.

**A screenshot of a computer

Description automatically generated**

## 4.2. Thiết kế các hàm khởi tạo và thuật toán

Các hàm khởi tạo và thuật toán cho chương trình vẽ đồ thị và thực hiện các thuật toán tìm đường đi ngắn nhất có thể như sau:

* Hàm khởi tạo cho lớp MyPanel: hàm này sẽ tạo ra một bảng điều khiển (control panel) cho chương trình, bao gồm các bảng con, nút nhấn, nút lựa chọn, ô nhập liệu, vùng văn bản, v.v. Hàm này sẽ thiết lập các thuộc tính cho các đối tượng như kích thước, màu sắc, biên, bố cục, v.v. Hàm này cũng sẽ thêm các đối tượng vào bảng điều khiển theo hướng Bắc (top), Trung tâm (center) và Nam (bottom) và đăng ký các bộ lắng nghe sự kiện cho các nút nhấn và nút lựa chọn.
* Hàm khởi tạo cho lớp DrawPanel: hàm này sẽ tạo ra một bảng vẽ (draw panel) cho chương trình, cho phép người dùng vẽ các đỉnh và cạnh của đồ thị bằng cách kéo thả chuột. Hàm này sẽ khởi tạo một đối tượng của lớp Graph để lưu trữ và xử lý các thông tin về đồ thị. Hàm này cũng sẽ thiết lập các thuộc tính cho bảng vẽ như kích thước, màu nền, v.v. Hàm này cũng sẽ đăng ký các bộ lắng nghe sự kiện cho chuột khi di chuyển hoặc nhấn trên bảng vẽ.
* Hàm khởi tạo cho lớp Graph: hàm này sẽ khởi tạo một đối tượng để lưu trữ và xử lý các thông tin về đồ thị, như danh sách đỉnh, danh sách cạnh, ma trận kề, v.v. Hàm này sẽ khởi tạo các mảng động rỗng để chứa các đối tượng của lớp Vertex và Edge.
* Hàm khởi tạo cho lớp Vertex: hàm này sẽ khởi tạo một đối tượng để biểu diễn một đỉnh của đồ thị, có các thuộc tính như tên, chỉ số, danh sách kề, hình tròn biểu diễn đỉnh trên bảng vẽ. Hàm này sẽ nhận vào các tham số tương ứng và gán cho các thuộc tính.
* Hàm khởi tạo cho lớp Edge: hàm này sẽ khởi tạo một đối tượng để biểu diễn một cạnh của đồ thị, có các thuộc tính như hai đỉnh đầu, đường thẳng biểu diễn cạnh trên bảng vẽ, điểm giữa của cạnh, trọng số của cạnh. Hàm này sẽ nhận vào các tham số tương ứng và gán cho các thuộc tính.
* Hàm khởi tạo cho lớp Algorithms: hàm này sẽ khởi tạo một đối tượng để thực hiện các thuật toán tìm đường đi ngắn nhất và duyệt đồ thị trên một đối tượng của lớp Graph. Hàm này sẽ nhận vào một đối tượng của lớp Vertex là đỉnh nguồn và một mảng hai chiều là ma trận kề của đồ thị. Hàm này sẽ gán giá trị cho các thuộc tính vertex, numberofvertices và mtk. Hàm này cũng sẽ khởi tạo các mảng distances, danhSachKe và chuaXet với kích thước phù hợp và giá trị ban đầu.
* Hàm xử lý sự kiện cho các nút nhấn và nút lựa chọn: hàm này sẽ xử lý các sự kiện khi người dùng nhấn vào các nút nhấn hoặc nút lựa chọn trên bảng điều khiển. Hàm này sẽ kiểm tra nguồn của sự kiện và thực hiện các hành động tương ứng như chuyển đổi loại đồ thị, gọi các phương thức của lớp DrawPanel để vẽ đồ thị và áp dụng các thuật toán tìm đường đi ngắn nhất hoặc duyệt đồ thị, hiển thị ma trận kề hoặc thông báo lỗi, v.v.
* Hàm xử lý sự kiện cho chuột: hàm này sẽ xử lý các sự kiện khi người dùng di chuyển hoặc nhấn chuột trên bảng vẽ. Hàm này sẽ kiểm tra loại sự kiện và tọa độ của chuột để thực hiện các hành động tương ứng như tạo ra một đỉnh mới, kết nối hai đỉnh bằng một cạnh mới, di chuyển một đỉnh hoặc cạnh đã có, v.v. Hàm này cũng sẽ gọi các phương thức của lớp Graph để cập nhật lại thông tin về đồ thị khi có thay đổi trên bảng vẽ.
* Hàm duyệt theo chiều rộng (BFS): hàm này sẽ duyệt đồ thị theo chiều rộng (Breadth-First Search) bắt đầu từ một đỉnh nguồn, nhận vào một số nguyên là chỉ số của đỉnh nguồn. Hàm này sử dụng cấu trúc dữ liệu hàng đợi (queue) để lưu trữ các đỉnh chưa được duyệt. Hàm này sẽ cập nhật lại giá trị của mảng chuaXet khi duyệt qua các đỉnh. Hàm này sẽ trả về một chuỗi là thứ tự duyệt các đỉnh của đồ thị.
* Hàm duyệt theo chiều sâu (DFS): hàm này sẽ duyệt đồ thị theo chiều sâu (Depth-First Search) bắt đầu từ một đỉnh nguồn, nhận vào một số nguyên là chỉ số của đỉnh nguồn. Hàm này sử dụng cấu trúc dữ liệu ngăn xếp (stack) để lưu trữ các đỉnh chưa được duyệt. Hàm này sẽ cập nhật lại giá trị của mảng chuaXet khi duyệt qua các đỉnh. Hàm này sẽ trả về một chuỗi là thứ tự duyệt các đỉnh của đồ thị.
* Hàm tìm đường đi ngắn nhất bằng thuật toán Bellman-Ford: hàm này sẽ tìm đường đi ngắn nhất từ một đỉnh nguồn tới các đỉnh khác trong đồ thị bằng thuật toán Bellman-Ford, nhận vào một số nguyên là chỉ số của đỉnh nguồn và một mảng động là danh sách cạnh của đồ thị. Hàm này sẽ cập nhật

A screenshot of a computer

Description automatically generated

# 5. CHƯƠNG V: TỔNG KẾT

## 5.1. Nhận xét kết quả đạt được

* Em đã thiết kế và xây dựng một chương trình vẽ đồ thị và thực hiện các thuật toán tìm đường đi ngắn nhất bằng ngôn ngữ Java, sử dụng các thư viện Swing và AWT để tạo giao diện người dùng.
* Chương trình cho phép người dùng vẽ các đỉnh và cạnh của đồ thị bằng cách kéo thả chuột trên một bảng vẽ có kích thước 10x10 ô vuông. Người dùng có thể chọn loại đồ thị là vô hướng hay có hướng, nhập vào trọng số của các cạnh, di chuyển hoặc xóa các đỉnh và cạnh đã có, v.v.
* Chương trình cũng cho phép người dùng áp dụng các thuật toán tìm đường đi ngắn nhất là Bellman-Ford và duyệt đồ thị theo chiều sâu (DFS) hoặc chiều rộng (BFS) trên đồ thị đã vẽ. Kết quả của các thuật toán được hiển thị trên một bảng chức năng ở phía dưới bảng vẽ, bao gồm ma trận kề của đồ thị, khoảng cách từ đỉnh nguồn tới các đỉnh khác, thứ tự duyệt các đỉnh của đồ thị, v.v.
* Em đã kiểm tra và so sánh hiệu năng của hai thuật toán DFS và BFS trên các tập dữ liệu khác nhau, bằng cách đo thời gian chạy và số lần duyệt các nút trong quá trình tìm kiếm. Kết quả cho thấy DFS có thời gian chạy nhanh hơn BFS, nhưng BFS có số lần duyệt nút ít hơn DFS. Điều này cho thấy DFS có tính hiệu quả cao hơn khi tìm kiếm trong không gian trạng thái rộng, nhưng BFS có tính đầy đủ cao hơn khi tìm kiếm trong không gian trạng thái sâu.

## 5.2. Hướng phát triển

* Em có thể cải tiến chương trình bằng cách bổ sung thêm các tính năng như zoom in, zoom out, xoay, lật, v.v. để người dùng có thể xem và chỉnh sửa đồ thị một cách thuận tiện hơn.
* Em cũng có thể áp dụng thêm các thuật toán tìm đường đi ngắn nhất khác như Dijkstra, A\*, Floyd-Warshall, v.v. để so sánh và phân tích kết quả với thuật toán Bellman-Ford.
* Ngoài ra, Em cũng có thể mở rộng chương trình để hỗ trợ các loại đồ thị khác như đồ thị có hướng không chu trình (DAG), đồ thị phân cấp (hierarchical graph), đồ thị mạng (network graph), v.v. để phục vụ cho các ứng dụng thực tế khác nhau.

**PHỤ LỤC**

Hướng dẫn sử dụng cho chương trình vẽ đồ thị và thực hiện các thuật toán tìm đường đi ngắn nhất như sau:

Để chọn loại đồ thị là vô hướng hay có hướng, người dùng có thể nhấn vào nút lựa chọn (radio button) tương ứng ở phần bên trái của bảng điều khiển. Nếu chọn đồ thị vô hướng, các cạnh của đồ thị sẽ không có hướng, nếu chọn đồ thị có hướng, các cạnh của đồ thị sẽ có mũi tên chỉ hướng.

Để vẽ một đỉnh mới trên bảng vẽ, người dùng có thể kéo chuột từ một ô vuông trống trên bảng vẽ tới một ô vuông khác. Một hình tròn màu xanh biểu diễn đỉnh mới sẽ xuất hiện ở ô vuông đầu tiên. Người dùng có thể nhập vào tên hoặc chỉ số của đỉnh mới ở ô nhập liệu “Tên đỉnh” ở phần bên trái của bảng chức năng, và nhấn nút “Thêm đỉnh” để gán tên cho đỉnh mới. Nếu không nhập tên, đỉnh mới sẽ được gán một số nguyên là chỉ số của nó trong danh sách đỉnh.

Để vẽ một cạnh mới nối hai đỉnh đã có trên bảng vẽ, người dùng có thể kéo chuột từ một hình tròn biểu diễn một đỉnh tới một hình tròn khác. Một đường thẳng màu xanh biểu diễn cạnh mới sẽ xuất hiện nối hai hình tròn. Người dùng có thể nhập vào trọng số của cạnh mới ở ô nhập liệu “Trọng số” ở phần bên trái của bảng chức năng, và nhấn nút “Thêm cạnh” để gán trọng số cho cạnh mới. Nếu không nhập trọng số, cạnh mới sẽ được gán giá trị 1.

Để di chuyển một đỉnh hoặc cạnh đã có trên bảng vẽ, người dùng có thể kéo chuột từ hình tròn hoặc đường thẳng biểu diễn đỉnh hoặc cạnh đó tới vị trí mong muốn. Các thông tin về đỉnh hoặc cạnh sẽ được cập nhật theo vị trí mới.

Để xóa một đỉnh hoặc cạnh đã có trên bảng vẽ, người dùng có thể nhập vào tên hoặc chỉ số của đỉnh hoặc hai đỉnh tương ứng của cạnh ở các ô nhập liệu “Điểm đầu”, “Điểm cuối” ở phần bên trái của bảng chức năng, và nhấn nút “Xoá đỉnh” hoặc “Xoá cạnh” để xóa đỉnh hoặc cạnh mong muốn. Nếu xóa một đỉnh, các cạnh kề với nó cũng sẽ bị xóa theo.

Để xóa toàn bộ đồ thị và vẽ lại từ đầu, người dùng có thể nhấn nút “Mới” ở phần bên trái của bảng chức năng. Tất cả các thông tin về đồ thị sẽ được xoá và khởi tạo lại.

Để áp dụng thuật toán Bellman-Ford để tìm đường đi ngắn nhất từ một đỉnh nguồn tới các đỉnh khác trong đồ thị, người dùng có thể nhập vào tên hoặc chỉ số của đỉnh nguồn ở ô nhập liệu “Source” ở phần ở giữa của bảng điều khiển, và nhấn nút “Tìm đường đi”. Kết quả của thuật toán sẽ được hiển thị ở phần bên phải của bảng chức năng, bao gồm khoảng cách từ đỉnh nguồn tới các đỉnh khác. Nếu có chu trình âm trong đồ thị, chương trình sẽ thông báo lỗi và không thực hiện thuật toán.

Để duyệt đồ thị theo chiều sâu (DFS) hoặc chiều rộng (BFS) bắt đầu từ một đỉnh nào đó, người dùng có thể nhập vào tên hoặc chỉ số của đỉnh đó ở ô nhập liệu “Đỉnh bắt đầu” ở phần bên phải của bảng điều khiển, và nhấn nút “DFS” hoặc “BFS”. Kết quả của thuật toán sẽ được hiển thị ở phần bên phải của bảng chức năng, bao gồm thứ tự duyệt các đỉnh của đồ thị.

Để xem ma trận kề của đồ thị, biểu diễn sự kề nhau giữa các đỉnh bằng số 0 hoặc 1, người dùng có thể nhấn nút “In” ở phần ở giữa của bảng chức năng. Ma trận kề sẽ được hiển thị ở phần bên trái của bảng chức năng. Người dùng có thể cập nhật ma trận kề khi thay đổi đồ thị trên bảng vẽ bằng cách nhấn lại nút “In”.

# TÀI LIỆU THAM KHẢO

1. Phạm Nguyên Khang. Bài giảng thực hành Lý thuyết đồ thị, Đại học Cần Thơ, 2021.
2. Mejia Adrian. "Graph Data Structures in JavaScript for Beginners", 15th December 2020. [Online]. Available: https://adrianmejia.com/data-structures-forbeginners-graphs-time-complexity-tutorial. [Accessed 2023 February 18].
3. Available: https://viblo.asia/p/do-thi -WAyK8Bj6lxX. [Accessed 5/3/2023].