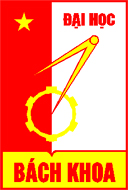
TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

──────── \* ───────



**BÀI TẬP LỚN**

HỌC PHẦN: LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

*(Mã học phần: IT3103)*

*Đề tài số 9:*

**MINH HỌA THUẬT TOÁN**

**ĐỊNH TUYẾN GÓI TIN TỔNG QUAN**

**Sinh viên thực hiện: NHÓM 24**

**Đinh Xuân Thanh 20236054**

**Lê Minh Thành 20225764**

**Nguyễn Công Thành 20215290**

**Nguyễn Đình Thành 20225670**

**Nguyễn Đình Phúc Thành 20225671**

**Lớp: 151965**

**Giảng viên hướng dẫn: TS. Nguyễn Thị Thu Trang**

***Hà Nội, tháng 12 năm 2024***

**PHÂN CÔNG THÀNH VIÊN TRONG NHÓM**

| **STT** | **Họ và tên** | **MSSV** | **Email** | **Công việc** | **Mức độ hoàn thành** | **Ghi chú** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | Đinh Xuân Thanh | 20236054 | thanh.dx236054@sis.hust.edu.vn | - Phân tích yêu cầu, thiết kế các biểu đồ Use Case, UML  - Code phụ | Tốt |  |
| 2 | Lê Minh Thành | 20225764 | thanh.lm225764@sis.hust.edu.vn | - Trưởng nhóm  - Code chính  - Thiết kế giao diện người dùng | Tốt |  |
| 3 | Nguyễn Công Thành | 20215290 | thanh.nc215290@sis.hust.edu.vn | - Phân tích yêu cầu  - Viết báo cáo  - Kiểm thử phần mềm | Tốt |  |
| 4 | Nguyễn Đình Thành | 20225670 | thanh.nd225670@sis.hust.edu.vn | - Code phụ  - Kiểm thử phần mềm | Tốt |  |
| 4 | Nguyễn Đình Phúc Thành | 20225671 | thanh.ndp225671@sis.hust.edu.vn | - Code phụ  - Kiểm thử phần mềm | Tốt |  |

# MỤC LỤC

[MỤC LỤC 3](#_heading=)

[LỜI NÓI ĐẦU 3](#_heading=)

[**CHƯƠNG 1. KHẢO SÁT, ĐẶC TẢ YÊU CẦU BÀI TOÁN 4**](#_heading=h.2et92p0)

[1.1. Mô tả yêu cầu bài toán 5](#_heading=)

[1.2. Biểu đồ Use Case 7](#_heading=h.1t3h5sf)

[1.2.1. Biểu đồ phân rã mức 1 7](#_heading=)

[1.2.2. Biểu đồ phân rã mức 2 8](#_heading=)

[1.2.3. Đặc tả Use Case 8](#_heading=)

[**CHƯƠNG 2. PHÂN TÍCH THIẾT KẾ BÀI TOÁN 9**](#_heading=h.3rdcrjn)

[2.1. Hệ thống các package 10](#_heading=)

[2.2. Thiết kế chi tiết lớp 10](#_heading=h.1ksv4uv)

[2.2.1. Package Model 11](#_heading=h.44sinio)

[2.2.2. Package Activity 12](#_heading=)

[2.2.3. Package Adapter 15](#_heading=h.3j2qqm3)

[2.2.4. Package Fragment 18](#_heading=)

[2.2.5. Package Dialog 20](#_heading=h.2xcytpi)

[**CHƯƠNG 3: CÔNG NGHỆ VÀ THUẬT TOÁN SỬ DỤNG 23**](#_heading=h.3whwml4)

[3.1. Ngôn ngữ lập trình và các thư viện được sử dụng 24](#_heading=h.ehm66n8jecse)

[3.2. Thuật toán và các kiến thức sử dụng 24](#_heading=h.syhwmmv7atk)

[Nguyên tắc Lập trình Hướng Đối Tượng (OOP) 25](#_heading=h.9xdu0ylwo0ex)

[CHƯƠNG 4: XÂY DỰNG CHƯƠNG TRÌNH MINH HOẠ 27](#_heading=)

[4.1. Kết quả của chương trình minh hoạ 27](#_heading=)

[4.2. Giao diện chương trình 27](#_heading=h.3o7alnk)

[4.2.1. Giao diện khởi động 27](#_heading=)

[4.2.2. Giao diện Show Graph 28](#_heading=h.ihv636)

[4.2.3. Giao diện Get Path 32](#_heading=)

[4.2.3. Giao diện Algorithm 33](#_heading=h.i7dab83r1rfj)

[4.3. Kiểm thử các chức năng đã thực hiện 34](#_heading=)

[**KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 34**](#_heading=h.vx1227)

[**TÀI LIỆU THAM KHẢO 35**](#_heading=h.3fwokq0)

# LỜI NÓI ĐẦU

Trong thế giới ngày càng kết nối của chúng ta, mạng máy tính đóng một vai trò trung tâm trong hầu hết mọi khía cạnh của đời sống hàng ngày và hoạt động kinh doanh. Để dữ liệu có thể di chuyển nhanh chóng và hiệu quả qua các mạng lớn, các thuật toán định tuyến chính xác và hiệu quả là cần thiết. Dự án này tập trung vào việc minh họa và phân tích ba thuật toán định tuyến chính: Dijkstra, Bellman-Ford và Flooding, với mục đích cung cấp một cái nhìn sâu sắc về cách thức hoạt động và hiệu quả của chúng trong các tình huống khác nhau.

Nhóm chúng em đã thiết kế một ứng dụng để mô phỏng và trực quan hóa các thuật toán này. Qua đó, người dùng có thể dễ dàng nhận thấy sự khác biệt về hiệu quả định tuyến và hiểu rõ hơn về tầm quan trọng của việc lựa chọn thuật toán phù hợp cho nhu cầu cụ thể của hệ thống mạng. Báo cáo này sẽ cung cấp một cái nhìn tổng quan về cách thức thiết kế và triển khai ứng dụng, cũng như phân tích chi tiết về từng thuật toán được sử dụng.

Chương trình của nhóm được thiết kế theo phương pháp Hướng đối tượng và được viết bằng ngôn ngữ lập trình Java - một ngôn ngữ có thể chạy được ở các nền tảng khác nhau thông qua một môi trường thực thi với điều kiện có môi trường thực thi thích hợp hỗ trợ nền tảng đó dẫn đến việc cài đặt chương trình và thực thi chương trình được dễ dàng hơn cho người sử dụng.

Mặc dù đã cố gắng hoàn thiện sản phẩm nhưng không thể tránh khỏi những thiếu hụt và sai sót về kiến thức và trong kiểm thử. Chúng em mong muốn nhận được những nhận xét chi tiết đến từ cô và các bạn để tiếp tục hoàn thiện hơn nữa. Cuối cùng, nhóm chúng em xin được gửi lời cảm ơn đến cô Nguyễn Thị Thu Trang đã hướng dẫn bọn em trong suốt quá trình hoàn thiện bài tập lớn. Xin trân trọng cảm ơn cô.

# CHƯƠNG 1. KHẢO SÁT, ĐẶC TẢ YÊU CẦU BÀI TOÁN

## Mô tả yêu cầu bài toán

**Minh họa Thuật toán Định tuyến Gói tin** **Tổng quan:** Các thuật toán định tuyến là thiết yếu cho việc truyền tải dữ liệu hiệu quả qua các mạng. Ứng dụng này sẽ trình bày các thuật toán định tuyến phổ biến như thuật toán Dijkstra, thuật toán Bellman-Ford, và Flooding, giúp người dùng hiểu về chức năng và sự khác biệt của chúng. Mục tiêu là để minh họa cách các quyết định định tuyến được thực hiện và ảnh hưởng của chúng đến hiệu suất mạng.

**Kiến thức Cơ bản:**

1. Cơ bản về lý thuyết đồ thị: nút (bộ định tuyến), cạnh (kết nối) và trọng số (chi phí).
2. Hiểu biết về các khái niệm định tuyến như đường đi ngắn nhất, bảng định tuyến và truyền tải gói tin.
3. Các thuật toán định tuyến chính:
   * Thuật toán Dijkstra: Tìm đường đi ngắn nhất trong đồ thị có trọng số.
   * Thuật toán Bellman-Ford: Xử lý đồ thị có trọng số cạnh âm.
   * Flooding: Truyền gói tin đến tất cả các nút hàng xóm mà không tối ưu hóa.

**Đặc tả:**

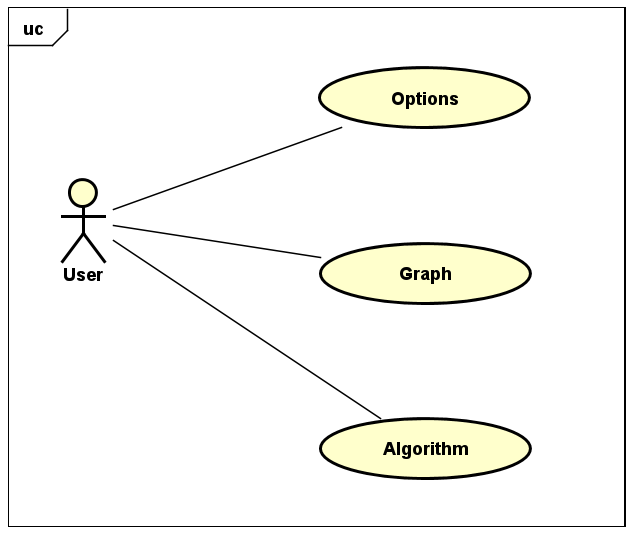
* **Giao diện người dùng (GUI):** Bạn có thể tự do thiết kế GUI của mình. Tuy nhiên, vì mục tiêu chính của dự án này là tập trung vào thiết kế ứng dụng hướng đối tượng, nên không nên ưu tiên giao diện. Bạn có thể tham khảo các ứng dụng mô phỏng mạng như Cisco Packet Tracer hoặc GNS3 để lấy ý tưởng về cách biểu diễn các top mạng.
  + Demo Cisco Packet Tracer:<https://youtu.be/6x9F0Io4bUM?t=322>
* **Thiết kế:** Chức năng chính bao gồm:
  + **Menu Chính:**
    - Tùy chọn để chọn một thuật toán định tuyến: Dijkstra, Bellman-Ford, hoặc Flooding.
    - Nút Trợ giúp mô tả ngắn gọn từng thuật toán.
    - Nút Thoát giúp thoát ứng dụng sau khi xác nhận.
  + **Màn hình Mô phỏng:**
    - Người dùng có thể minh họa top mạng với các nút (bộ định tuyến, switch, máy tính) và cạnh (kết nối).
    - Điều khiển để thêm/bỏ bộ định tuyến và liên kết.
    - Hiển thị bảng định tuyến và di chuyển gói tin thời gian thực.
    - Nút để mô phỏng và minh họa thuật toán định tuyến đã chọn.
  + **Menu Trợ giúp:**
    - Cung cấp mô tả ngắn gọn về từng thuật toán và cơ chế của nó.
  + **Tùy chọn Thoát:**
    - Nhắc người dùng xác nhận trước khi thoát.

**Tóm tắt Kiến thức:**

* **Biểu diễn Đồ thị trong Mạng:**
  + Một mạng có thể được biểu diễn như một đồ thị, nơi bộ định tuyến hoạt động như các nút và liên kết (kết nối vật lý hoặc lô-gic) là các cạnh. Mỗi liên kết có một trọng số hoặc chi phí có thể đại diện cho khoảng cách, thời gian, hoặc tốc độ truyền dữ liệu.
* **Thuật toán Dijkstra:**
  + Thuật toán Dijkstra được sử dụng để tìm đường đi ngắn nhất từ một nút bắt đầu (bộ định tuyến) đến tất cả các nút khác trong đồ thị có trọng số. Nó sử dụng cách tiếp cận tham lam, liên tục chọn nút có khoảng cách tạm thời nhỏ nhất và cập nhật các đường đi tương ứng.
* **Thuật toán Bellman-Ford:**
  + Bellman-Ford là một thuật toán lập trình động tìm đường đi ngắn nhất từ một nút nguồn đến tất cả các nút khác, ngay cả khi có trọng số cạnh âm. Nó hoạt động bằng cách thư giãn các cạnh liên tục, có thể phát hiện các chu kỳ trọng số âm trong đồ thị.
* **Thuật toán Flooding:**
  + Flooding là một kỹ thuật định tuyến đơn giản, nơi các gói dữ liệu được gửi đến tất cả các bộ định tuyến lân cận cho đến khi chúng đạt đến điểm đến của mình. Nó được sử dụng trong các tình huống mà các tuyến đường không được biết hoặc không đáng tin cậy nhưng có thể gây ra lưu lượng mạng cao và kém hiệu quả.

## Biểu đồ Use Case

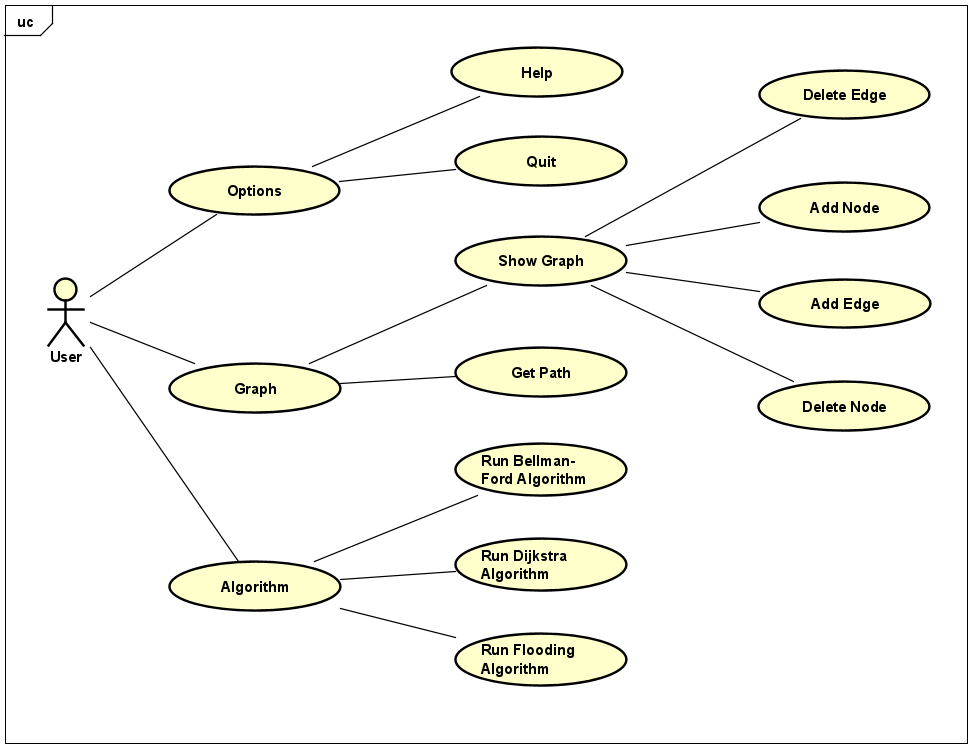
### Biểu đồ phân rã mức 1

****

Chương trình của nhóm có 3 use case chính là :

1. Options (Tùy chọn)
2. Graph (Vẽ đồ thị)
3. Algorithm (Chọn và chạy thuật toán)

### Biểu đồ phân rã mức 2



### Đặc tả Use Case

* + - 1. Danh sách Actor

| Tên Actor | Ý Nghĩa |
| --- | --- |
| User | Người dùng chương trình |

* + - 1. Danh sách Use Case

| Tên Use Case | Ý nghĩa |
| --- | --- |
| Graph | Đưa người dùng đến các chức năng chính của chương trình là Vẽ đồ thị và Tìm các path. |
| Algorithm | Đưa người dùng đến các chức năng Chạy các thuật toán Dijkstra, Bellman-Ford và Flooding từ đồ thị đã vẽ. |
| Options | Đưa người dùng đến 2 chức năng: Help (Trợ giúp) và Quit (Thoát ra). |

* + - 1. Đặc tả Use Case

1. Use case Graph

Tác nhân : User

| STT | Thực Hiện | Hành động |
| --- | --- | --- |
| 1 | System | Hiển thị 2 nút: Show Graph và Get Path |
| 2 | User | Người dùng chọn chức năng Show Graph |
| 3 | System | Hiển thị màn hình Vẽ Graph với các chức năng Add Node, Delete Node, Add Edge, Delete Edge |
| 4 | User | Sử dụng các chức năng trên để vẽ graph |
| 5 | User | Sau khi vẽ xong, người dùng nhấn nút Get Path |
| 6 | System | Hiển thị màn hình Get Path |
| 7 | User | Nhập Node, sau đó nhấn Button Get Path |
| 8 | System | Hiển thị Path theo thuật toán Dijkstra |

1. Use case Algorithm

Tác nhân: User

| STT | Thực hiện | Hành động |
| --- | --- | --- |
| 1 | System | Hiển thị 3 nút: Dijkstra Algorithm, Bellman-Ford Algorithm và Flooding Algorithm |
| 2 | User | * Chọn chức năng mà mình cần thực hiện * Điền các Node cần thiết để tìm Path * Nhấn nút PATH để chạy chương trình |
| 3 | System | Hiển thị Path thỏa mãn theo thuật toán tương ứng |

1. Use case Options

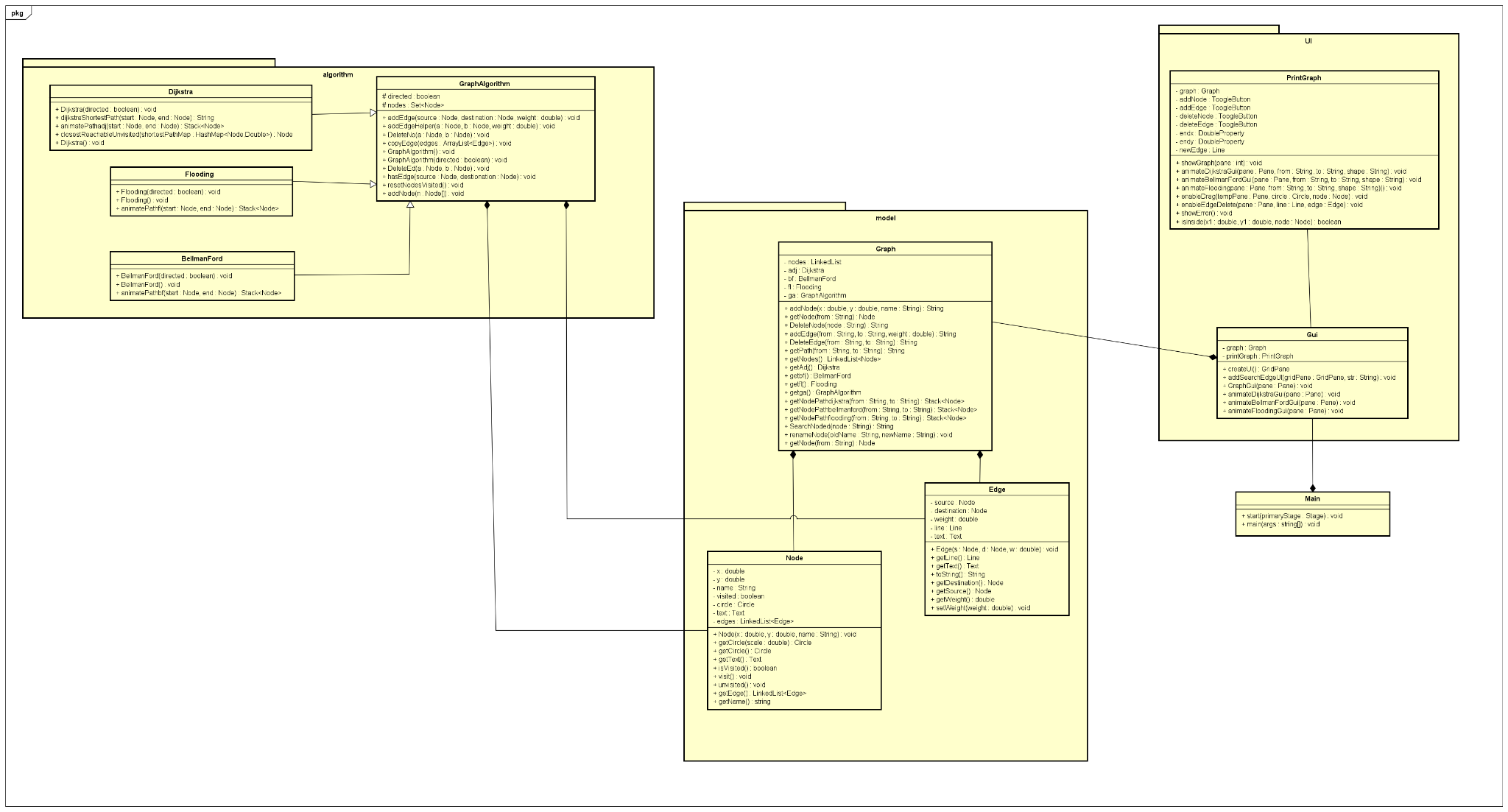
Tác nhân: User

| STT | Thực hiện | Hành động |
| --- | --- | --- |
| 1 | System | Hiển thị 2 nút: Help và Quit |
| 2 | User | Chọn chức năng cần thiết:   * Help khi cần trợ giúp * Quit khi cần thoát ra |

# 

# CHƯƠNG 2. PHÂN TÍCH THIẾT KẾ BÀI TOÁN

## Hệ thống các package



**Package model**

* Mục đích:  
   Gói này chứa các lớp mô hình chính của chương trình, gồm Graph, Node, và Edge. Đây là tầng cơ bản, chịu trách nhiệm quản lý và cung cấp dữ liệu của đồ thị.
* Vai trò:
  + Xây dựng và quản lý cấu trúc đồ thị, bao gồm các đỉnh (node) và cạnh (edge).
  + Cung cấp các phương thức để thêm, xóa, sửa đổi và truy xuất dữ liệu đồ thị.
  + Là nền tảng để thực hiện các thuật toán và hiển thị giao diện.

**Package algorithm**

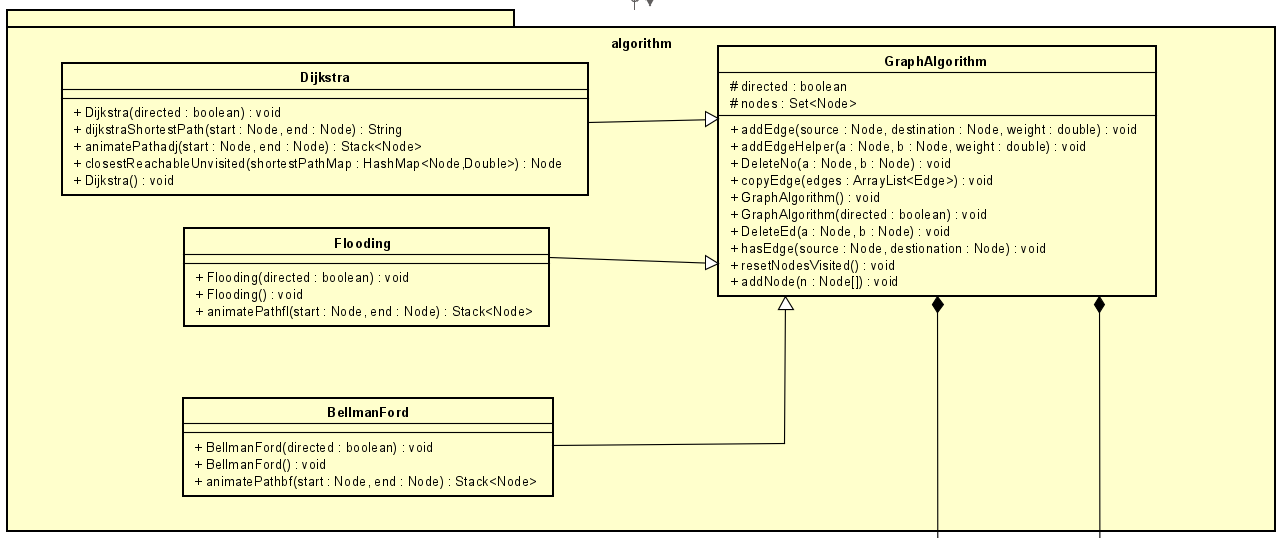
* Mục đích:  
   Gói này chứa các thuật toán đồ thị cơ bản và các lớp hỗ trợ thực hiện các thao tác với dữ liệu đồ thị.
* Vai trò:
  + Cung cấp lớp cơ sở chung (GraphAlgorithm) quản lý các thông tin cơ bản như đỉnh và cạnh.
  + Chứa các thuật toán đồ thị như Dijkstra, Bellman-Ford và Flooding, giúp tìm kiếm đường đi ngắn nhất hoặc xử lý các bài toán liên quan đến đồ thị.
  + Cung cấp các phương thức hỗ trợ thực hiện và mô phỏng các thuật toán.

**Package UI**

* Mục đích:  
   Gói này tập trung vào việc xử lý giao diện người dùng, cung cấp khả năng tương tác trực quan với đồ thị.
* Vai trò:
  + Quản lý giao diện đồ họa, như hiển thị các nút và cạnh trên màn hình.
  + Kết nối giữa người dùng và các dữ liệu đồ thị, cho phép người dùng thêm/xóa/sửa đồ thị hoặc chạy các thuật toán.
  + Tích hợp và hiển thị các hiệu ứng động (animation) cho các thuật toán, giúp trực quan hóa kết quả.

## Thiết kế chi tiết lớp

### Package algorithm



Gói này chứa các lớp và phương pháp dùng để giải quyết các bài toán thuật toán trên đồ thị.

**Lớp GraphAlgorithm**

* Đây là lớp cơ sở trừu tượng (base class) dùng chung cho các thuật toán đồ thị.
* Thuộc tính:
  + directed: boolean: Xác định đồ thị có hướng hay không.
  + nodes: Set<Node>: Tập hợp các nút (nodes) trong đồ thị.
* Phương thức:
  + addEdge(Node source, Node destination, double weight): Thêm cạnh nối giữa hai nút.
  + addEdgeHelper(Node a, Node b, double weight): Hỗ trợ thêm cạnh.
  + DeleteNo(Node a, Node b): Xóa một nút khỏi đồ thị.
  + copyEdge(ArrayList<Edge> edges): Tạo bản sao danh sách các cạnh.
  + hasEdge(Node source, Node destination): Kiểm tra xem có cạnh nối giữa hai nút không.
  + resetNodesVisited(): Đặt lại trạng thái thăm của các nút.
  + addNode(Node[] n): Thêm một hoặc nhiều nút vào đồ thị.

Lớp Dijkstra

* Kế thừa từ lớp GraphAlgorithm.
* Là thuật toán tìm đường ngắn nhất từ một nút đến tất cả các nút khác trong đồ thị.
* Phương thức:
  + dijkstraShortestPath(Node start, Node end): String: Tìm đường đi ngắn nhất giữa hai nút.
  + animatePathdj(Node start, Node end): Stack<Node>: Tạo hoạt ảnh cho đường đi ngắn nhất.
  + closestReachableUnvisited(HashMap<Node, Double> shortestPathMap): Node: Tìm nút gần nhất chưa được thăm.

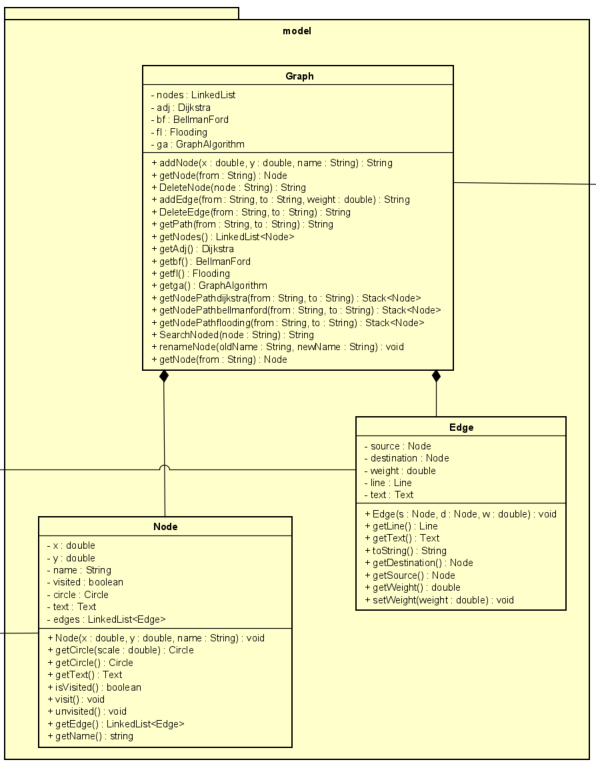
**Lớp Flooding**

* Kế thừa từ lớp GraphAlgorithm.
* Là thuật toán lan truyền (flooding) trong đồ thị.
* Phương thức:
  + animatePathfl(Node start, Node end): Stack<Node>: Tạo hoạt ảnh cho quá trình lan truyền giữa hai nút.

**Lớp BellmanFord**

* Kế thừa từ lớp GraphAlgorithm.
* Là thuật toán Bellman-Ford để tìm đường ngắn nhất trong đồ thị, kể cả khi đồ thị có cạnh trọng số âm.
* Phương thức:
  + animatePathbf(Node start, Node end): Stack<Node>: Tạo hoạt ảnh cho đường đi ngắn nhất.

### Package model



**Lớp Graph**

* Vị trí: Lớp trung tâm trong gói, đại diện cho đồ thị và chứa các thuật toán, nút, cạnh để thao tác.
* Thuộc tính:
  + nodes: LinkedList<Node>: Danh sách các nút trong đồ thị.
  + adj: Dijkstra: Đối tượng cho thuật toán Dijkstra.
  + bf: BellmanFord: Đối tượng cho thuật toán Bellman-Ford.
  + fl: Flooding: Đối tượng cho thuật toán Flooding.
  + ga: GraphAlgorithm: Lớp cha trừu tượng đại diện cho các thuật toán đồ thị.
* Phương thức:
  + addNode(x: double, y: double, name: String): String: Thêm một nút mới vào đồ thị.
  + getNode(from: String): Node: Lấy một nút cụ thể từ đồ thị theo tên.
  + DeleteNode(node: String): String: Xóa một nút khỏi đồ thị.
  + addEdge(from: String, to: String, weight: double): String: Thêm một cạnh nối giữa hai nút.
  + DeleteEdge(from: String, to: String): String: Xóa cạnh nối giữa hai nút.
  + getPath(from: String, to: String): String: Tìm đường đi giữa hai nút.
  + getNodes(): LinkedList<Node>: Lấy danh sách tất cả các nút.
  + getAdj(): Dijkstra: Lấy đối tượng thuật toán Dijkstra.
  + getBf(): BellmanFord: Lấy đối tượng thuật toán Bellman-Ford.
  + getFl(): Flooding: Lấy đối tượng thuật toán Flooding.
  + getGa(): GraphAlgorithm: Lấy đối tượng thuật toán chung.
  + getNodePathDijkstra(from: String, to: String): Stack<Node>: Lấy đường đi từ Dijkstra.
  + getNodePathBellman(from: String, to: String): Stack<Node>: Lấy đường đi từ Bellman-Ford.
  + getNodePathFlooding(from: String, to: String): Stack<Node>: Lấy đường đi từ thuật toán Flooding.
  + SearchNode(node: String): String: Tìm kiếm một nút trong đồ thị.
  + renameNode(oldName: String, newName: String): void: Đổi tên một nút trong đồ thị.
  + getNode(from: String): Node: Trả về đối tượng nút cụ thể.

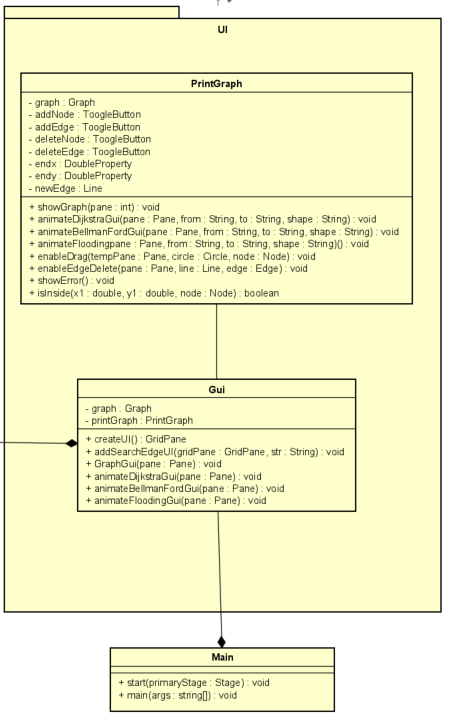
**Lớp Node**

* Vị trí: Đại diện cho các đỉnh (nút) trong đồ thị.
* Thuộc tính:
  + x: double, y: double: Tọa độ của nút.
  + name: String: Tên của nút.
  + visited: boolean: Trạng thái đã thăm hay chưa.
  + circle: Circle: Hình tròn đại diện cho nút trong giao diện đồ họa.
  + text: Text: Văn bản hiển thị bên cạnh nút.
  + edges: LinkedList<Edge>: Danh sách các cạnh kết nối từ nút này.
* Phương thức:
  + Node(x: double, y: double, name: String): void: Hàm khởi tạo để tạo một nút mới.
  + getCircle(): Circle: Lấy đối tượng đồ họa hình tròn.
  + getText(): Text: Lấy văn bản liên quan đến nút.
  + isVisited(): boolean: Kiểm tra trạng thái đã thăm.
  + visit(): void: Đánh dấu nút là đã thăm.
  + unvisit(): void: Đặt lại trạng thái chưa thăm.
  + getEdge(): LinkedList<Edge>: Lấy danh sách các cạnh kết nối từ nút.
  + getName(): String: Lấy tên của nút.

**Lớp Edge**

* Vị trí: Đại diện cho các cạnh nối giữa hai nút trong đồ thị.
* Thuộc tính:
  + source: Node: Nút nguồn của cạnh.
  + destination: Node: Nút đích của cạnh.
  + weight: double: Trọng số của cạnh.
  + line: Line: Đường nối đại diện cho cạnh trên giao diện đồ họa.
  + text: Text: Văn bản hiển thị trọng số của cạnh.
* Phương thức:
  + Edge(source: Node, destination: Node, weight: double): void: Hàm khởi tạo một cạnh.
  + getLine(): Line: Lấy đối tượng đồ họa đường nối.
  + getText(): Text: Lấy văn bản hiển thị trọng số.
  + toString(): String: Trả về chuỗi biểu diễn cạnh.
  + getDestination(): Node: Lấy nút đích của cạnh.
  + getSource(): Node: Lấy nút nguồn của cạnh.
  + getWeight(): double: Lấy trọng số của cạnh.
  + setWeight(weight: double): void: Đặt trọng số mới cho cạnh.

### Package UI



**Lớp PrintGraph**

* Vị trí: Đại diện cho giao diện hiển thị và thao tác với đồ thị trong chương trình.
* Thuộc tính:
  + graph: Graph: Đối tượng đồ thị để hiển thị và thao tác.
  + addNode: ToggleButton: Nút chuyển đổi để thêm nút vào đồ thị.
  + addEdge: ToggleButton: Nút chuyển đổi để thêm cạnh vào đồ thị.
  + deleteNode: ToggleButton: Nút chuyển đổi để xóa nút khỏi đồ thị.
  + deleteEdge: ToggleButton: Nút chuyển đổi để xóa cạnh khỏi đồ thị.
  + endx: DoubleProperty, endy: DoubleProperty: Tọa độ kết thúc của một thao tác đồ họa (chẳng hạn thêm cạnh).
  + newEdge: Line: Đối tượng đồ họa đại diện cho một cạnh mới.
* Phương thức:
  + showGraph(pane: int): void: Hiển thị đồ thị trong một giao diện cụ thể.
  + animateDijkstraGui(pane: Pane, from: String, to: String, shape: String): void: Thực hiện và hiển thị hoạt ảnh thuật toán Dijkstra trên giao diện.
  + animateBellmanFordGui(pane: Pane, from: String, to: String, shape: String): void: Thực hiện và hiển thị hoạt ảnh thuật toán Bellman-Ford trên giao diện.
  + animateFlooding(pane: Pane, from: String, to: String, shape: String): void: Thực hiện và hiển thị hoạt ảnh thuật toán Flooding trên giao diện.
  + enableDrag(tempPane: Pane, circle: Circle, node: Node): void: Cho phép kéo thả các nút trên giao diện.
  + enableEdgeDelete(pane: Pane, line: Line, edge: Edge): void: Cho phép xóa cạnh trực tiếp từ giao diện.
  + showError(): void: Hiển thị lỗi nếu có thao tác sai.
  + isInside(x1: double, y1: double, node: Node): boolean: Kiểm tra xem một tọa độ cụ thể có nằm trong vùng nút hay không.

**Lớp Gui**

* Vị trí: Quản lý giao diện chính của chương trình.
* Thuộc tính:
  + graph: Graph: Đối tượng đồ thị được sử dụng trong chương trình.
  + printGraph: PrintGraph: Đối tượng hỗ trợ hiển thị và thao tác đồ thị.
* Phương thức:
  + createUI(): GridPane: Tạo giao diện người dùng chính dưới dạng một GridPane.
  + addSearchEdgeUI(gridPane: GridPane, str: String): void: Thêm giao diện tìm kiếm hoặc thêm cạnh vào đồ thị.
  + GraphGui(pane: Pane): void: Khởi tạo giao diện chính của đồ thị.
  + animateDijkstraGui(pane: Pane): void: Thực hiện và hiển thị hoạt ảnh thuật toán Dijkstra trong giao diện chính.
  + animateBellmanFordGui(pane: Pane): void: Thực hiện và hiển thị hoạt ảnh thuật toán Bellman-Ford trong giao diện chính.
  + animateFloodingGui(pane: Pane): void: Thực hiện và hiển thị hoạt ảnh thuật toán Flooding trong giao diện chính.

# CÔNG NGHỆ VÀ THUẬT TOÁN SỬ DỤNG

## 3.1. Ngôn ngữ lập trình và các thư viện được sử dụng

**Ngôn ngữ Lập trình:**

* **Java:** Đây là ngôn ngữ lập trình chính được sử dụng trong dự án của bạn. Java là lựa chọn phổ biến cho các ứng dụng do khả năng chạy đa nền tảng, mạnh mẽ và hỗ trợ lập trình hướng đối tượng.

**Thư viện JavaFX:**

* **JavaFX:** Được sử dụng để xây dựng giao diện người dùng đồ họa (GUI). Các thành phần như Scene, Stage, Button, Pane, và Menu được sử dụng để tạo cửa sổ và các điều khiển tương tác, cung cấp một phương thức hiệu quả và trực quan để hiển thị dữ liệu và chức năng của ứng dụng.

**Các Thư viện và Package:**

* **application.UI:** Chứa các lớp như Gui và PrintGraph để xử lý các yếu tố giao diện người dùng và đồ họa của ứng dụng.
* **application.model:** Bao gồm các lớp Node và Edge, dùng để xây dựng và quản lý cấu trúc dữ liệu đồ thị cơ bản.
* **application.algorithm:** Chứa các lớp Dijkstra, Bellman-Ford, và Flooding, chịu trách nhiệm thực hiện các thuật toán định tuyến và các tính toán liên quan.

## 3.2. Thuật toán và các kiến thức sử dụng

**Các Thuật toán Đồ thị:**

* **Dijkstra:** Tìm đường đi ngắn nhất trong đồ thị có trọng số từ một nút đến tất cả các nút khác. Sử dụng một hàng đợi ưu tiên để lựa chọn nút có khoảng cách tạm thời ngắn nhất​.
* **Bellman-Ford:** Hỗ trợ các đồ thị có trọng số cạnh âm và có thể phát hiện chu trình trọng số âm trong đồ thị. Thuật toán này làm giảm dần khoảng cách ước lượng từ nút nguồn đến các nút khác thông qua việc thư giãn các cạnh​.
* **Flooding:** Sử dụng phương pháp phát sóng, gửi gói tin tới tất cả hàng xóm mà không dựa vào bảng định tuyến, thích hợp cho các mạng không có cấu trúc rõ ràng hoặc khi cần đảm bảo sự lan tỏa thông tin đến toàn bộ mạng​.

**Kiến thức Cơ bản và Cài đặt:**

* **Lý thuyết đồ thị và Cấu trúc Dữ liệu:** Sử dụng kiến thức cơ bản về đồ thị như nút, cạnh, trọng số cạnh, và các kỹ thuật như thăm nút, duyệt đồ thị.
* **Cấu trúc dữ liệu Java:** Dùng các cấu trúc dữ liệu như ArrayList, LinkedList, Stack, Queue, và HashMap để lưu trữ và quản lý dữ liệu trong ứng dụng.
* **Kiến trúc Phần mềm:** Thiết kế ứng dụng theo mô hình MVC (Model-View-Controller), phân tách rõ ràng giữa xử lý nghiệp vụ (model và controller) và hiển thị (view), giúp dễ dàng quản lý và mở rộng ứng dụng.

### 

### Nguyên tắc Lập trình Hướng Đối Tượng (OOP)

**1. Đóng gói (Encapsulation):**

* **Đóng gói dữ liệu:** Trong dự án của bạn, đóng gói được thể hiện qua việc mỗi lớp như Node, Edge, và các thuật toán (Dijkstra, BellmanFord, Flooding) đều giữ các thuộc tính và phương thức liên quan đến chúng trong cùng một đơn vị. Ví dụ, lớp Node chứa dữ liệu về vị trí, tên, và danh sách các cạnh liên kết, cùng với các phương thức để quản lý trạng thái của nút như visit() hoặc unvisited()​.
* **Bảo vệ trạng thái đối tượng:** Việc truy cập vào các thuộc tính của đối tượng được kiểm soát chặt chẽ thông qua các phương thức setter và getter, giúp tránh truy cập trực tiếp vào các trường dữ liệu một cách không an toàn.

**2. Kế thừa (Inheritance):**

* **Tái sử dụng và mở rộng mã nguồn:** Các lớp thuật toán (Dijkstra, BellmanFord, Flooding) kế thừa từ lớp GraphAlgorithm, tái sử dụng các thuộc tính và phương thức chung như quản lý danh sách các nút và cạnh. Điều này cho phép mở rộng chức năng mà không cần viết lại mã nguồn từ đầu​.
* **Đa hình thông qua kế thừa:** Các lớp thuật toán có thể được tham chiếu và sử dụng thông qua loại dữ liệu GraphAlgorithm, cho phép thay đổi thuật toán một cách linh hoạt trong runtime.

**3. Đa hình (Polymorphism):**

* **Thay đổi hành vi đối tượng:** Đa hình cho phép các đối tượng từ các lớp con (Dijkstra, BellmanFord, Flooding) có hành vi khác nhau khi thực hiện cùng một phương thức kế thừa từ lớp cha (GraphAlgorithm). Điều này được thể hiện rõ ràng khi gọi phương thức animatePath, mỗi thuật toán sẽ có cách hiển thị đường đi riêng​​.
* **Đa hình trong xử lý sự kiện:** Phương thức xử lý sự kiện trong GUI có thể gọi các phương thức cụ thể của thuật toán dựa trên đối tượng thuật toán đang được chọn, điều này chứng minh khả năng linh hoạt và mở rộng của đa hình trong thực tế​.

**4. Trừu tượng (Abstraction):**

* **Che giấu phức tạp:** Lớp GraphAlgorithm và các lớp kế thừa từ nó che giấu các chi tiết thực hiện cụ thể của thuật toán định tuyến, chỉ cung cấp một giao diện trừu tượng để khám phá đường đi trong đồ thị. Người dùng của lớp này không cần quan tâm đến cách thực hiện bên trong, chỉ cần biết các phương thức có sẵn để sử dụng​.

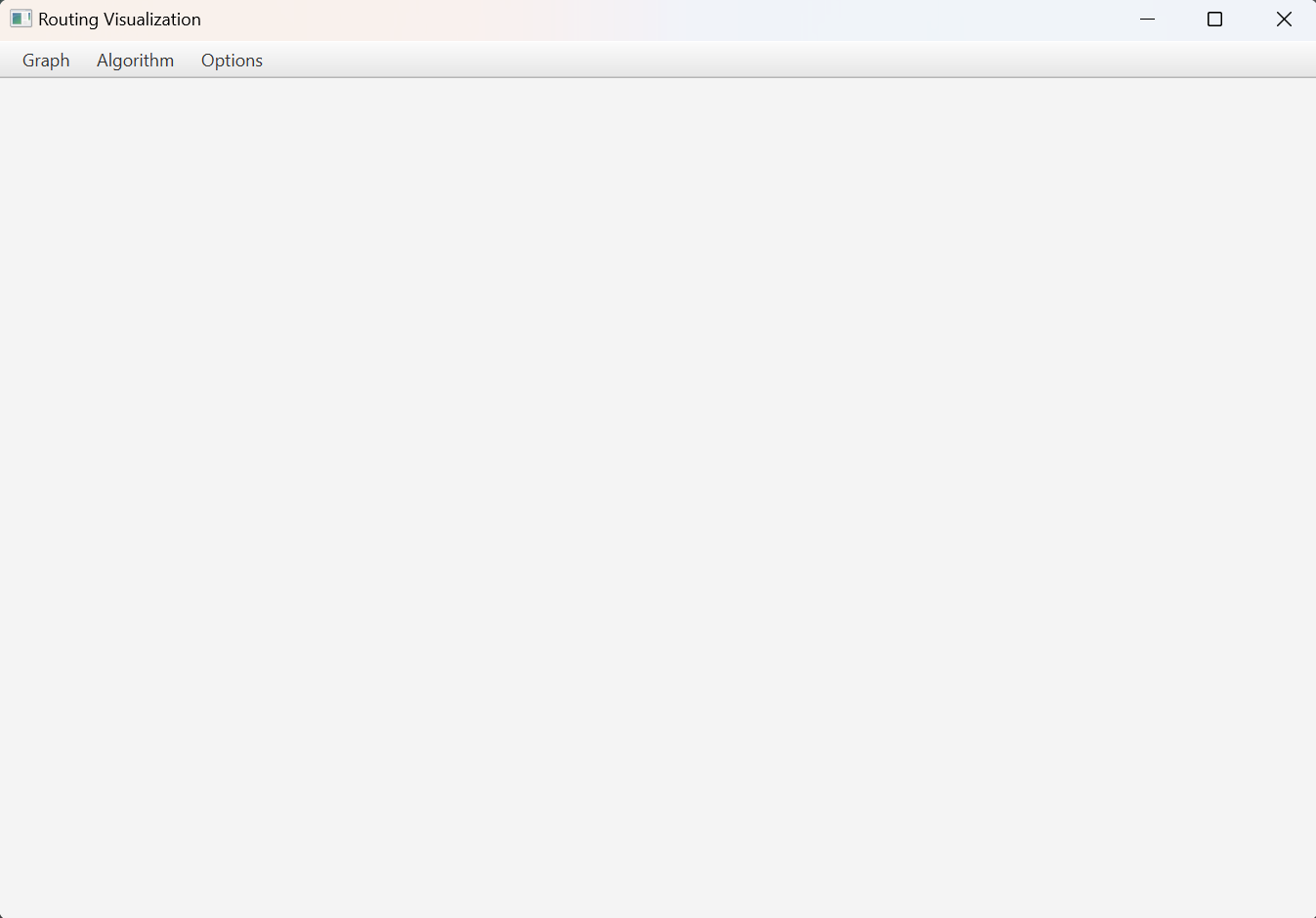
# CHƯƠNG 4: XÂY DỰNG CHƯƠNG TRÌNH MINH HOẠ

## 4.1. Kết quả của chương trình minh hoạ

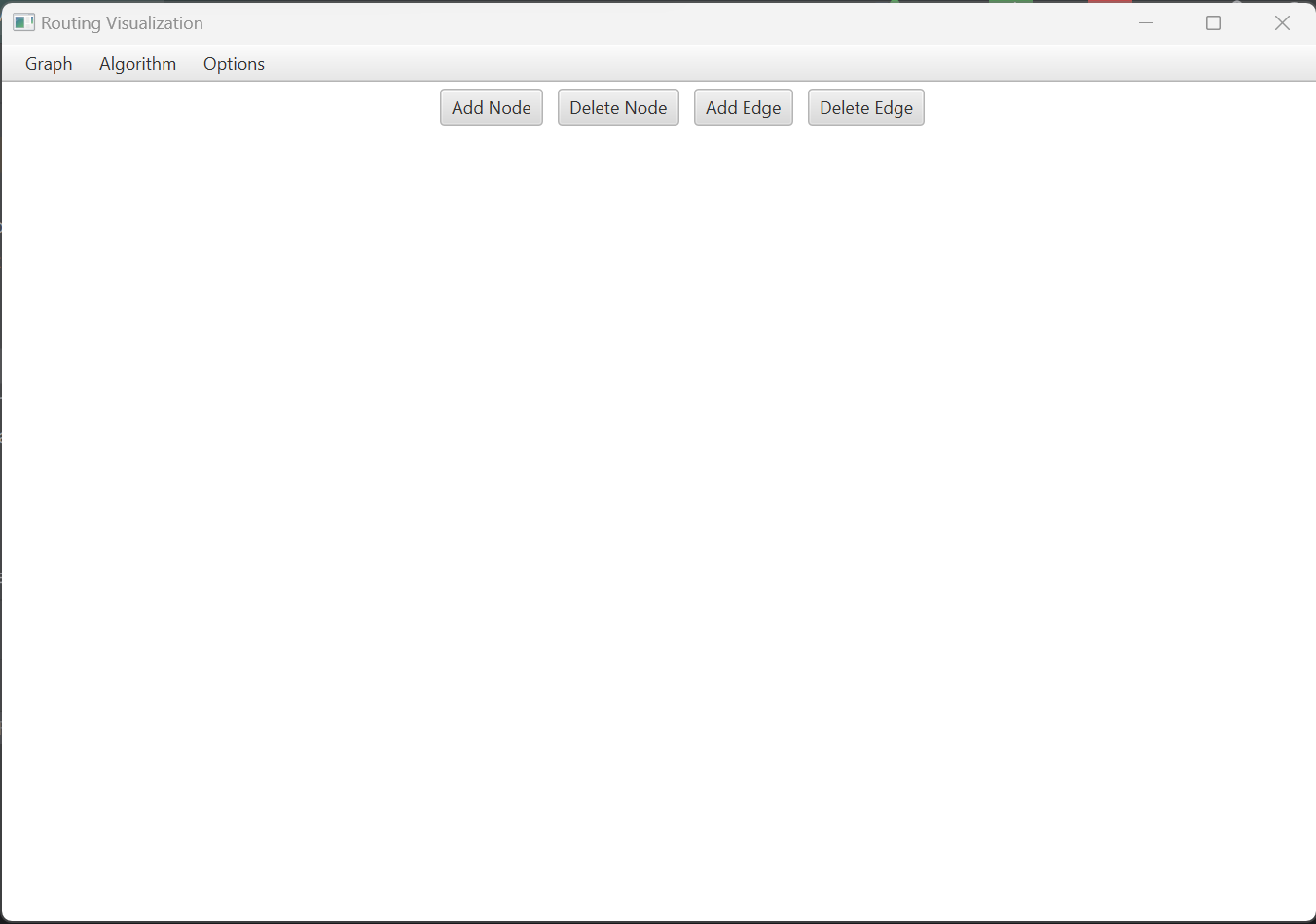
Chương trình được xây dựng về ban đầu đã đáp ứng đầy đủ tất cả các yêu cầu ban đầu của đề tài về chức năng và giao diện.

## 4.2. Giao diện chương trình

### 4.2.1. Giao diện khởi động

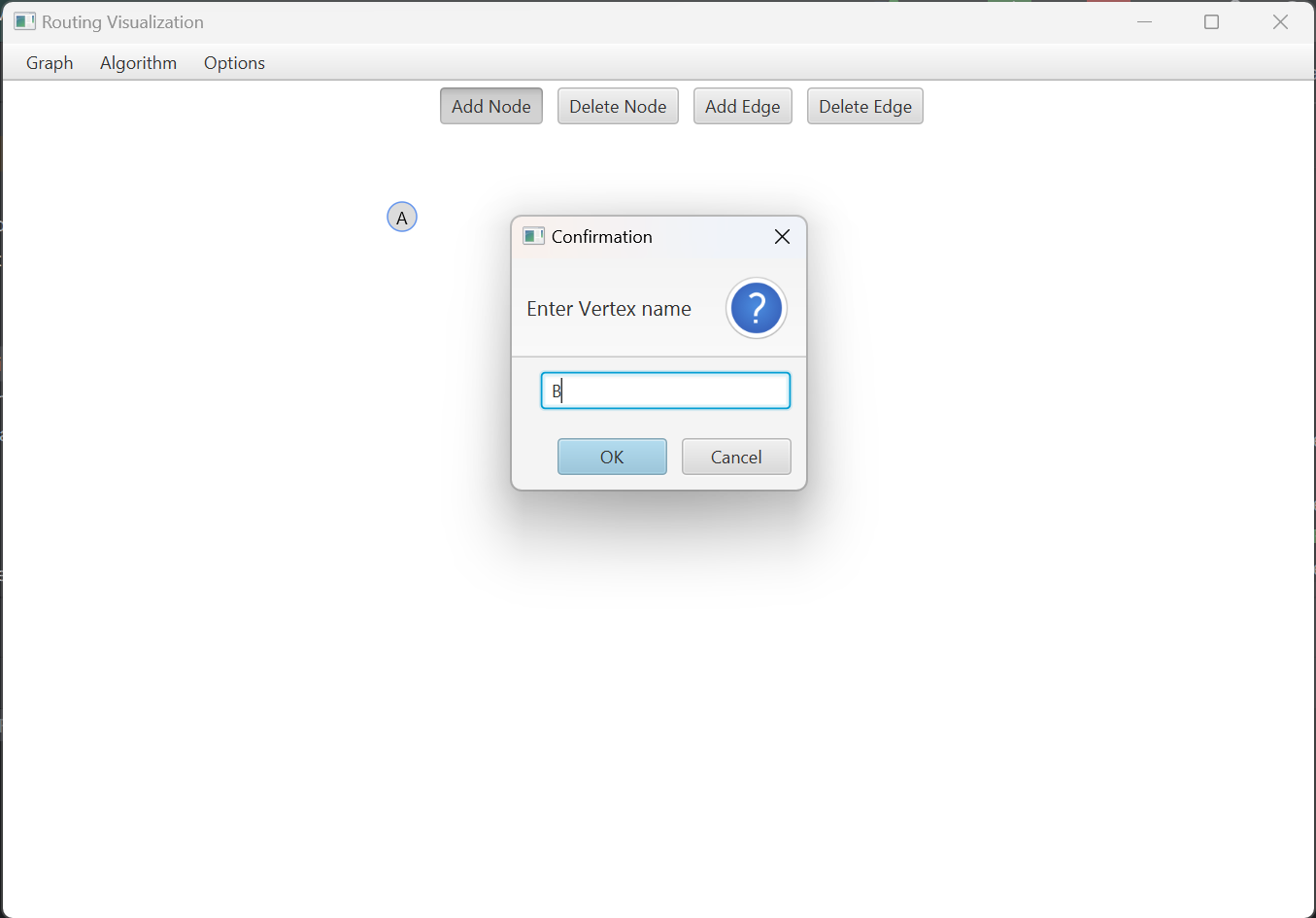


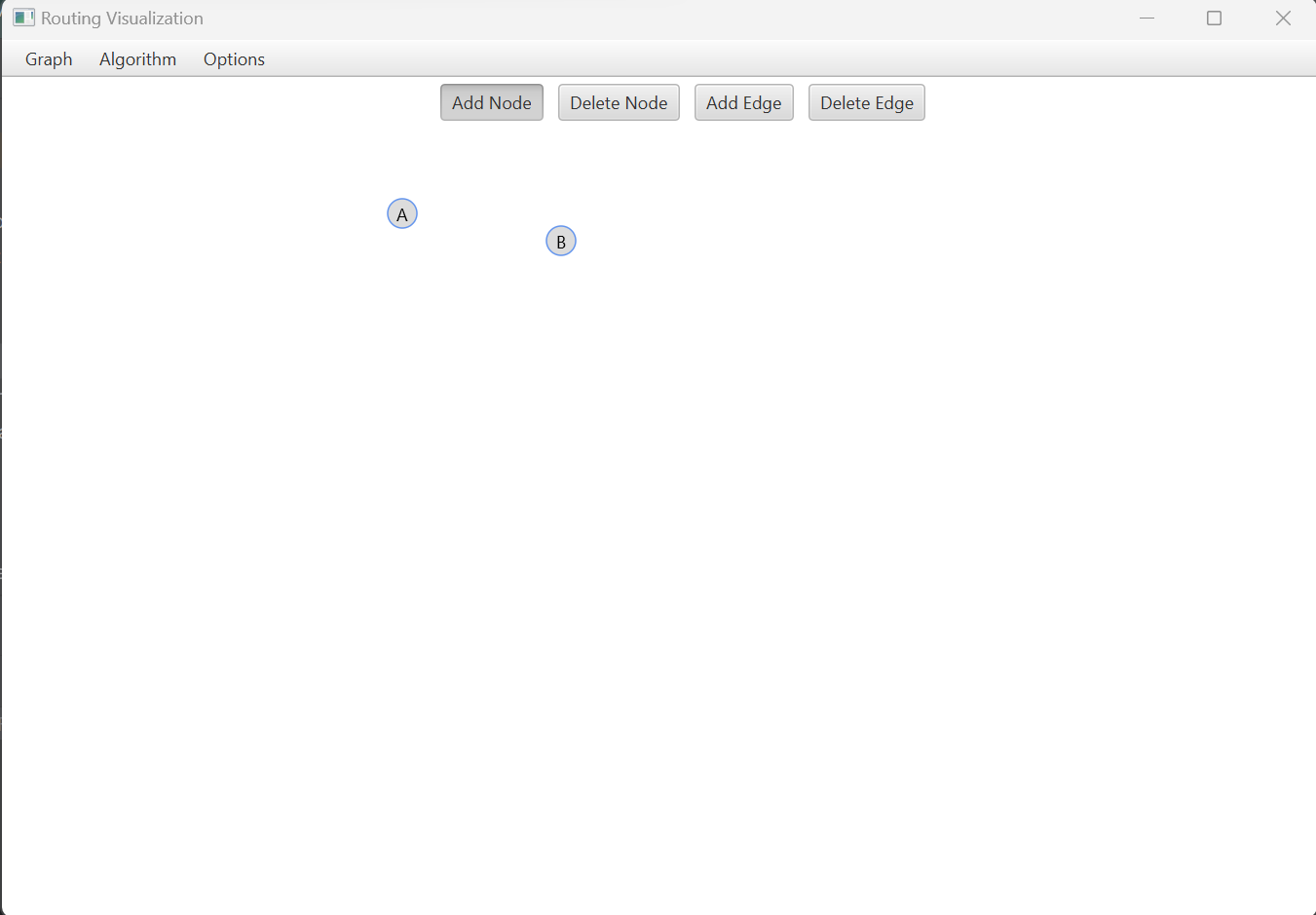
### 4.2.2. Giao diện Show Graph



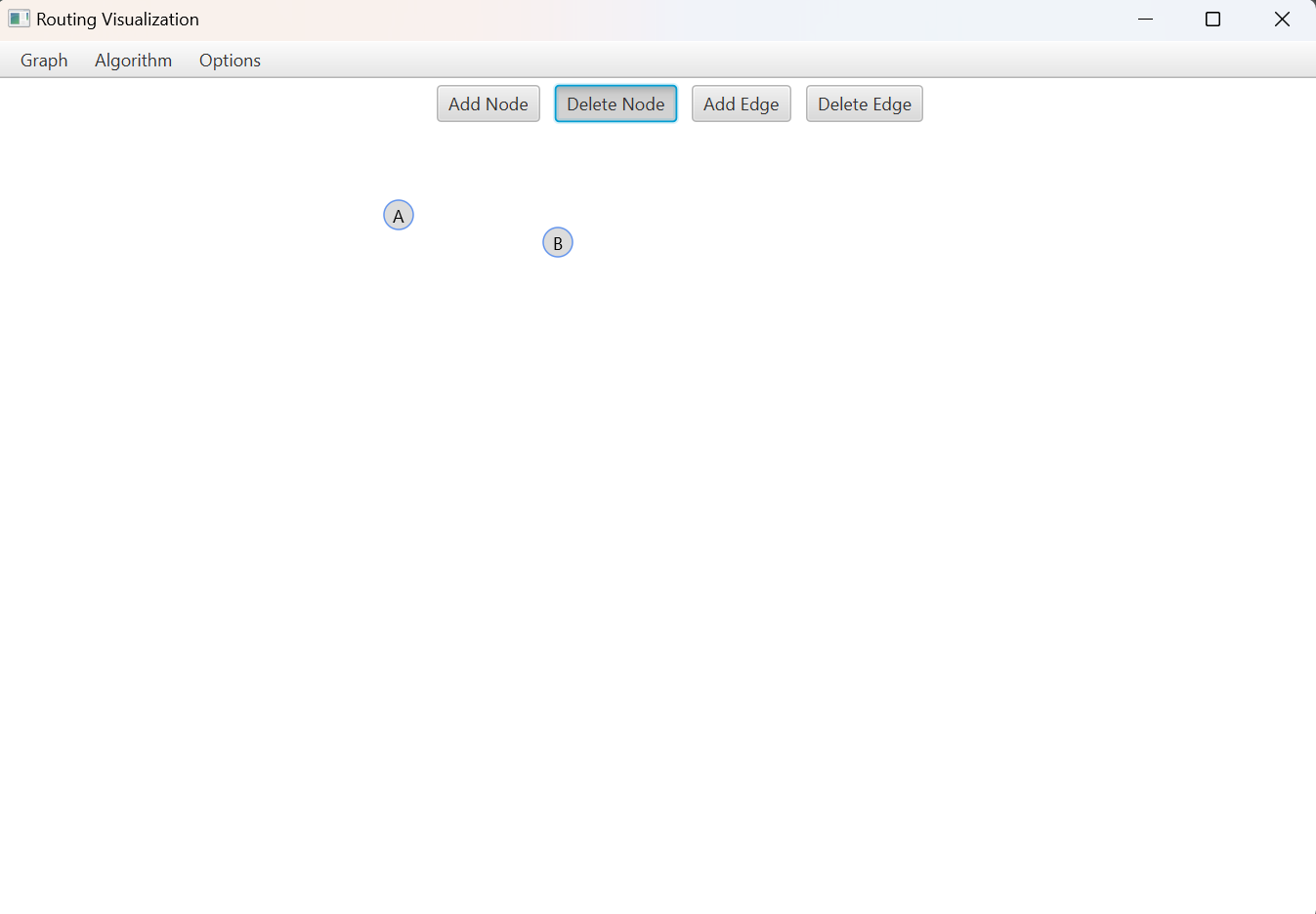
Các chức năng tương tác có trong giao diện:

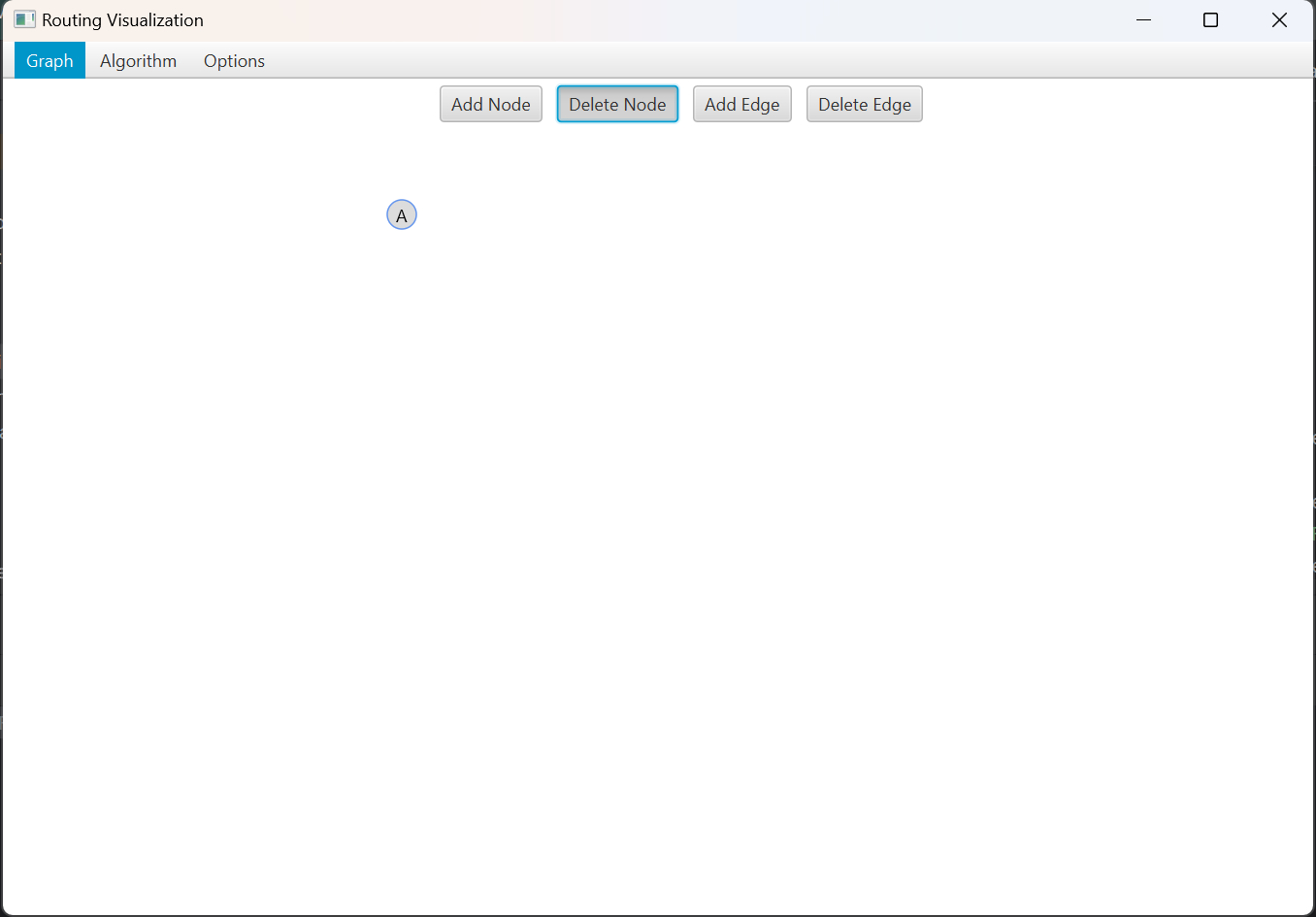
1. Thao tác Thêm 1 Node vào đồ thị: Nhấn nút “Add Node” trên màn hình, click vào vị trí muốn tạo Node, xuất hiện Dialog, điền tên Node, nhấn OK, Node mới sẽ được tạo.



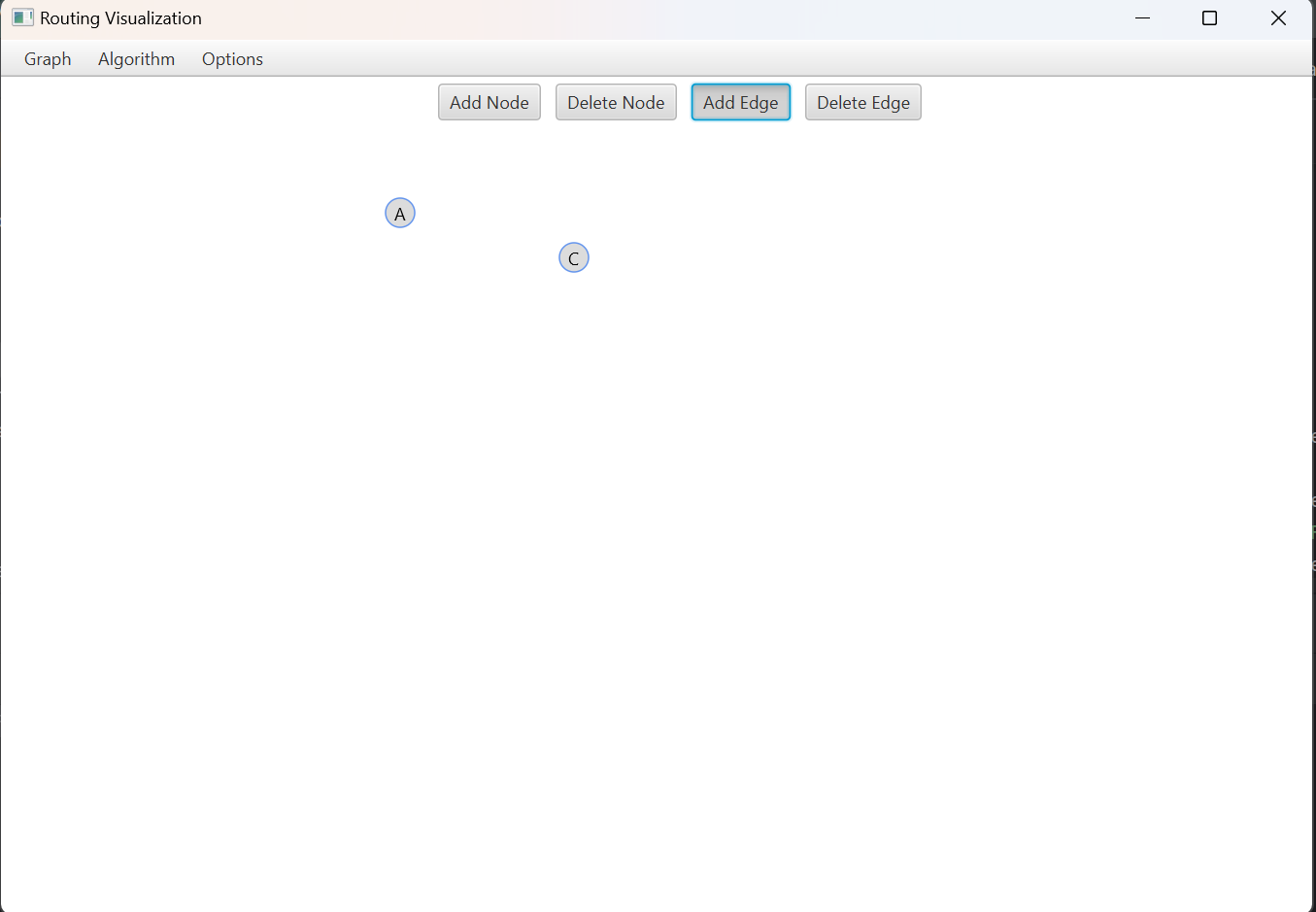


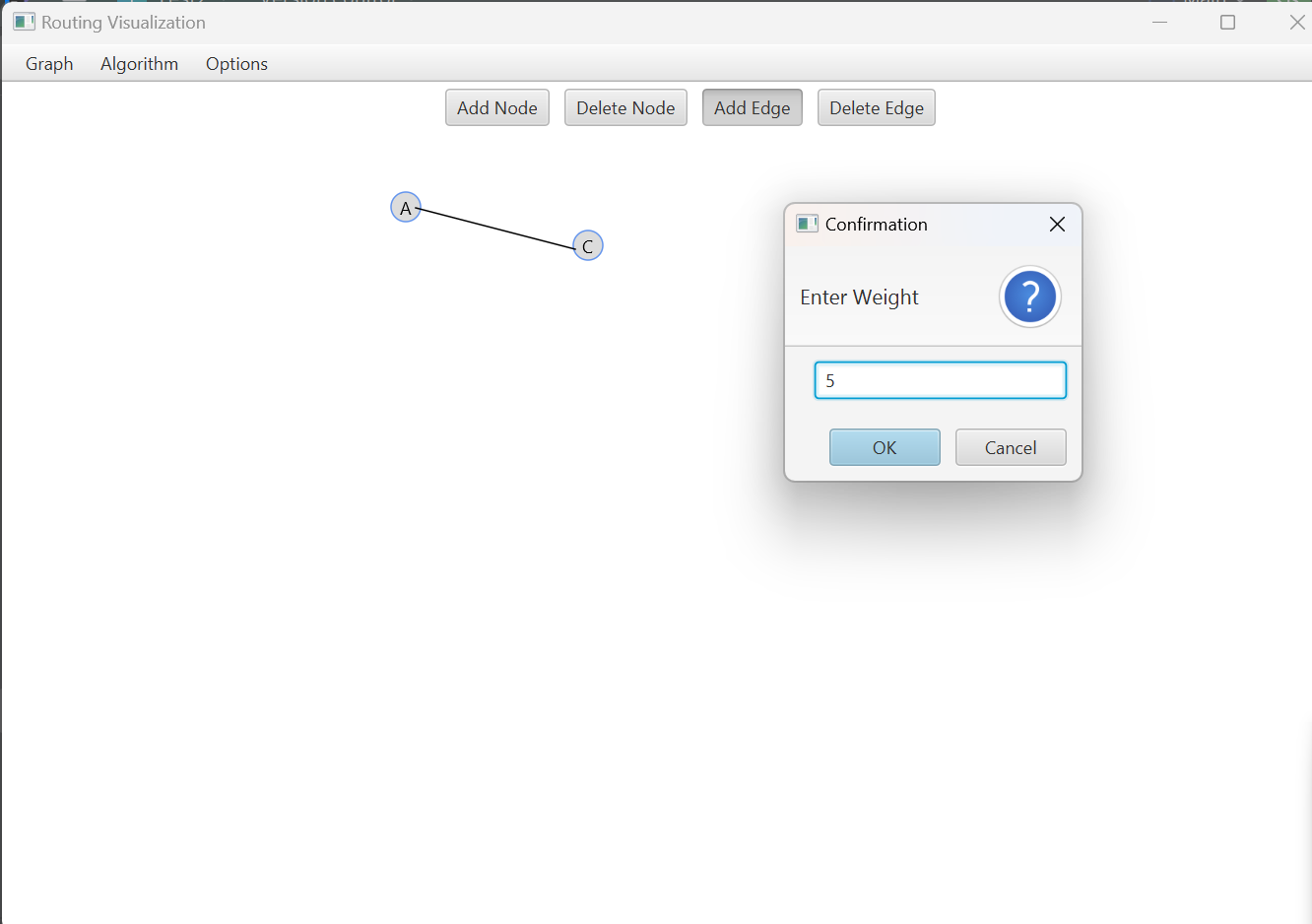
1. Thao tác Xóa 1 Node trong đồ thị: Nhấn nút “Delete Node” trên màn hình, sau đó click vào nút cần xóa, Node sẽ được xóa.

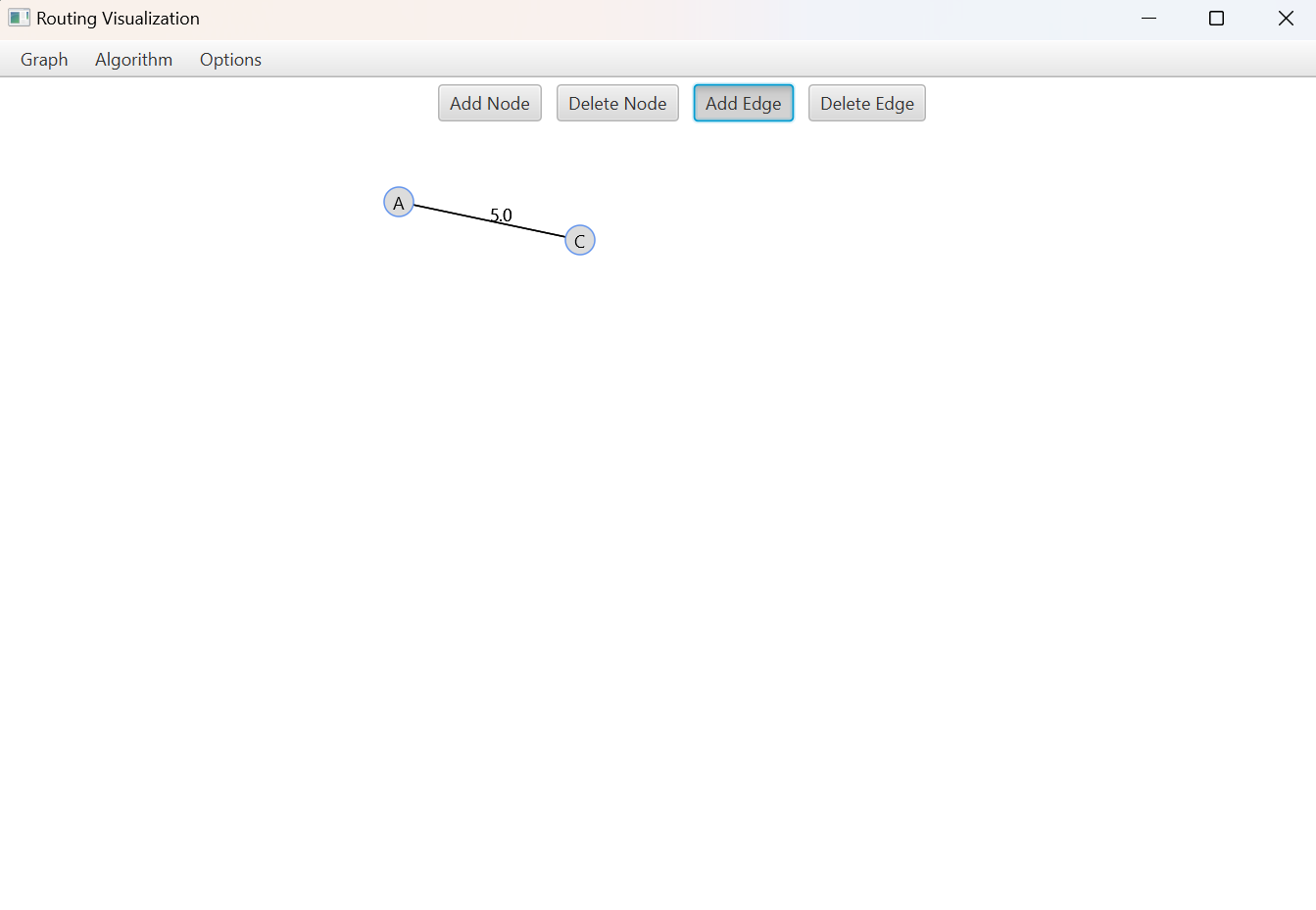




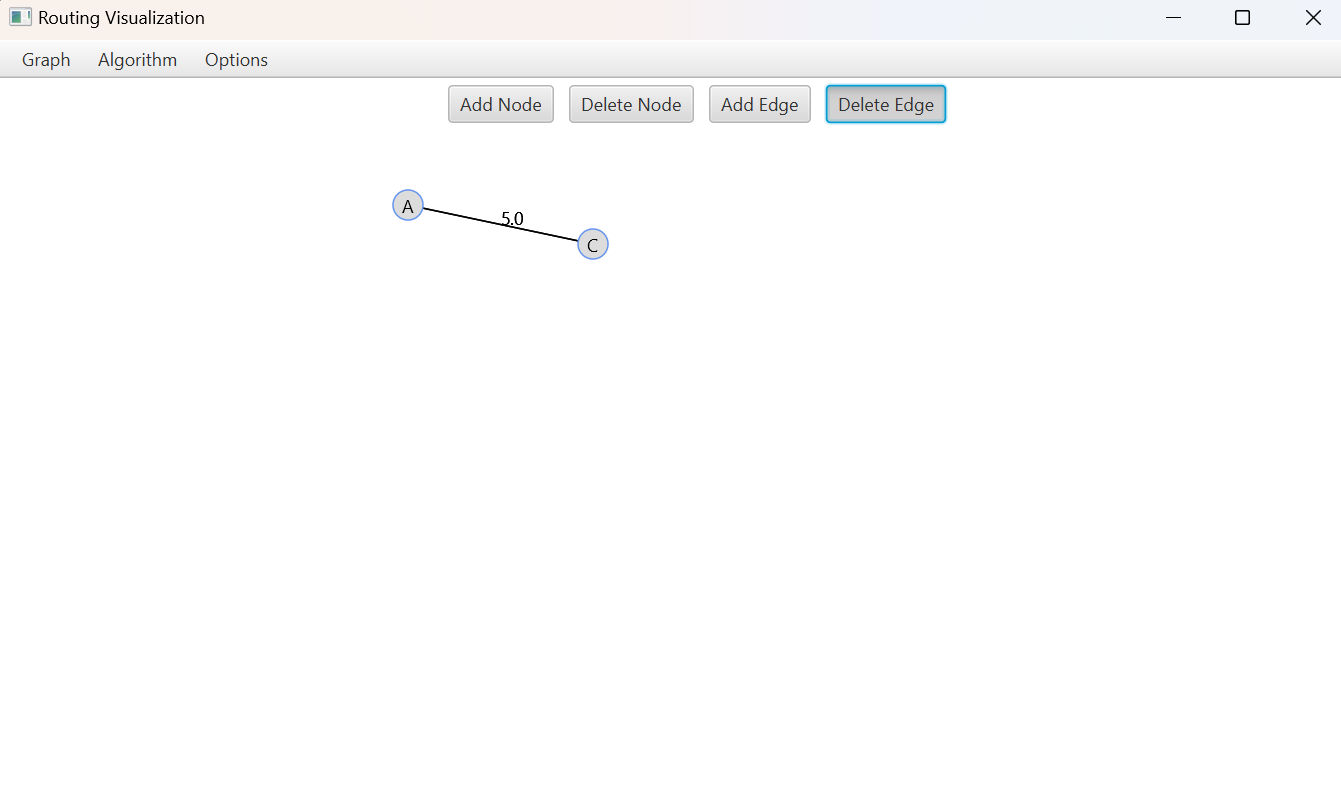
1. Thao tác Thêm 1 Edge: Nhấn nút “Add Edge”, sau đó click lần lượt vào 2 Node cần tạo Edge, một Dialog xuất hiện, điền Weight và nhấn OK, Edge mới sẽ được tạo.

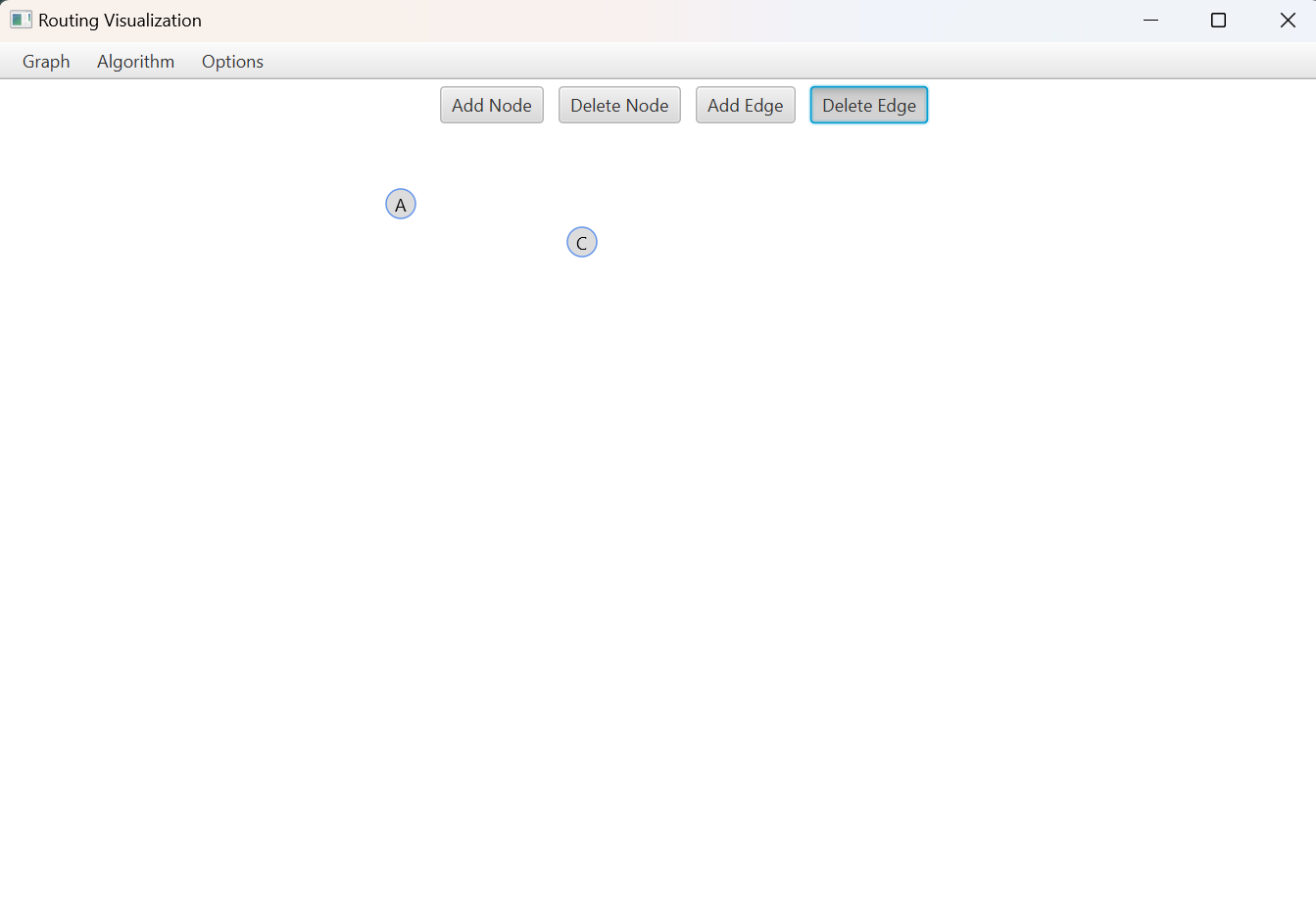






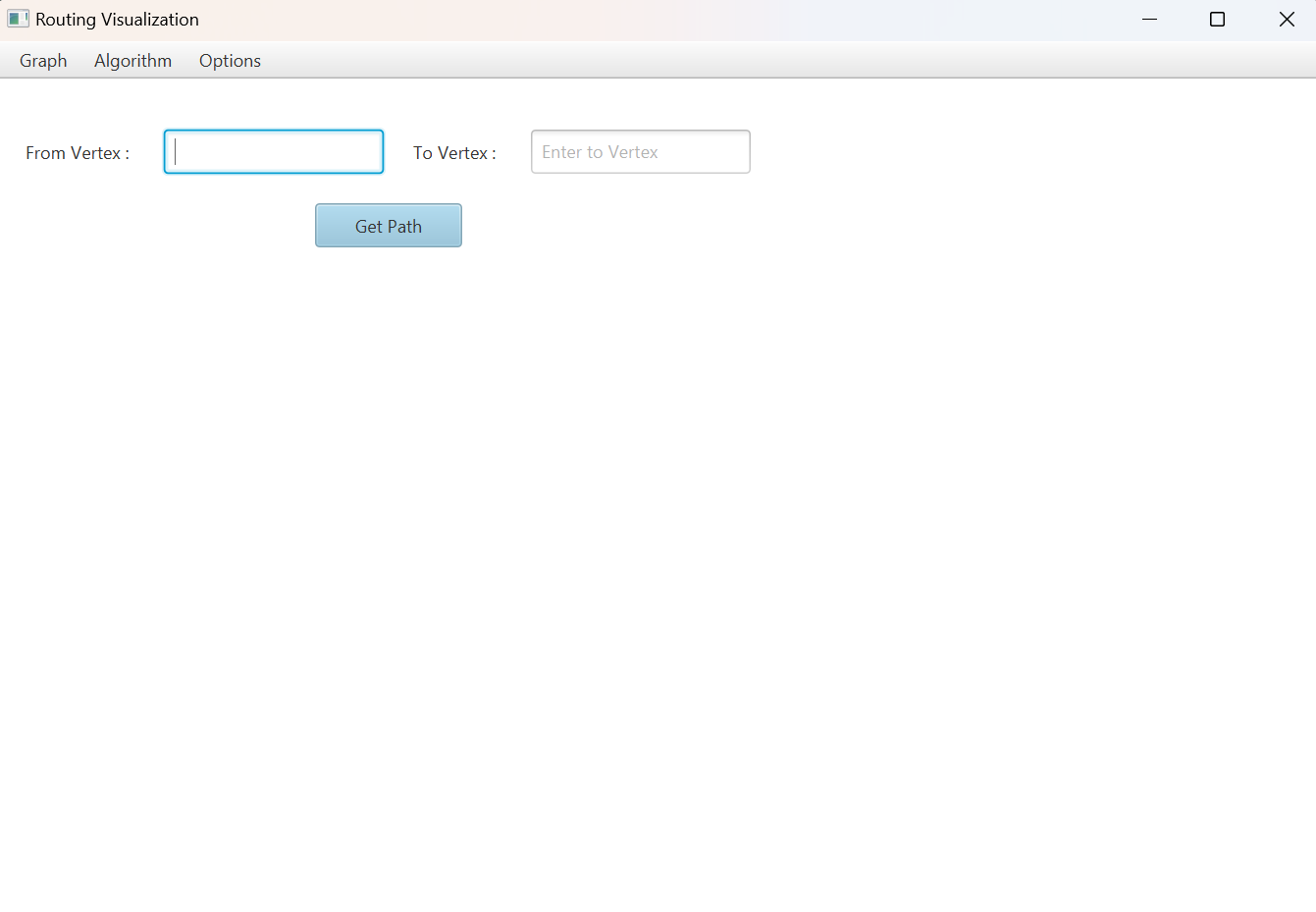
1. Thao tác Xóa 1 Edge: Nhấn nút “Delete Edge”, sau đó click vào Edge cần xóa, Edge đó sẽ được xóa.



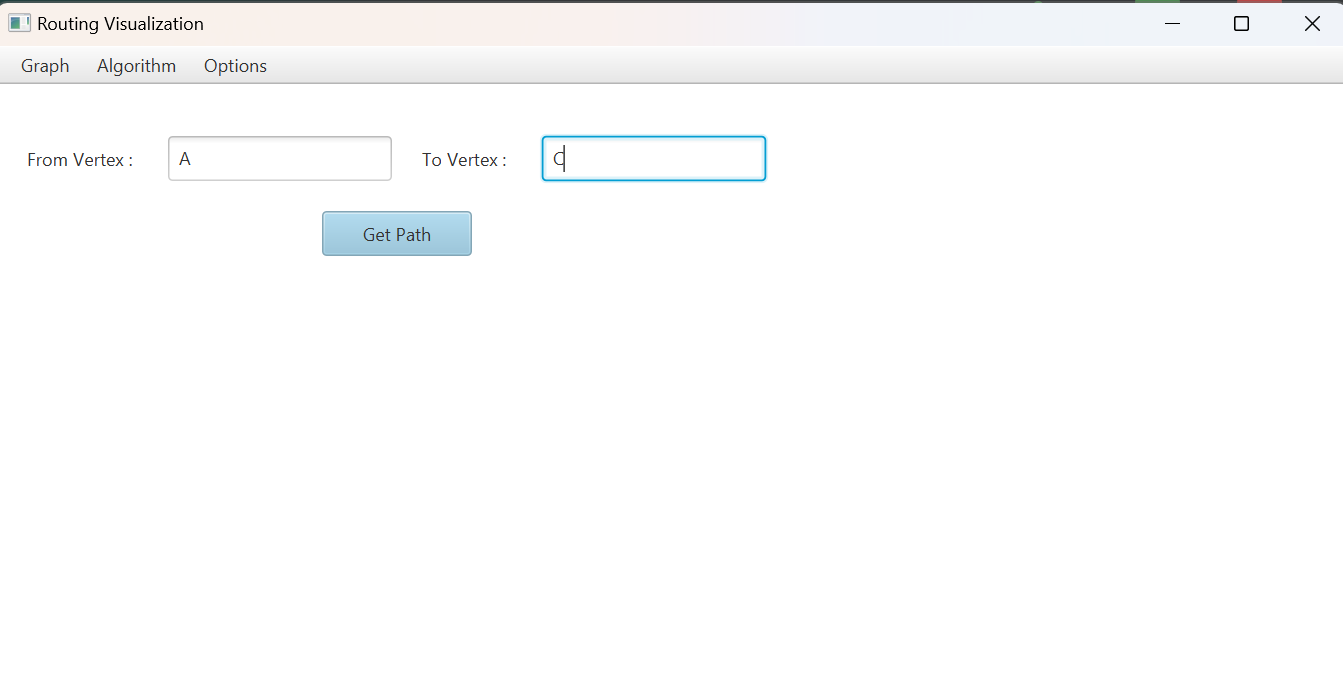


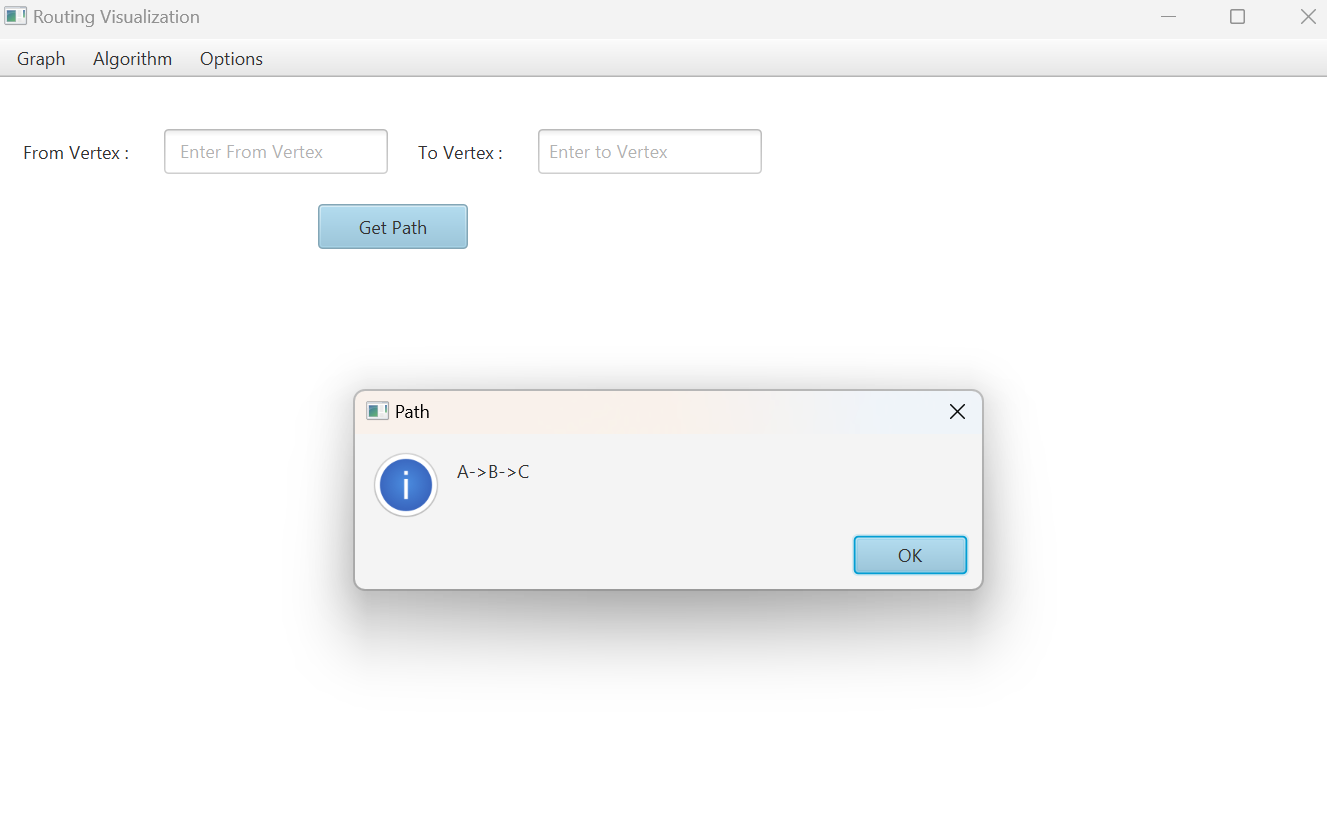
### 4.2.3. Giao diện Get Path

Màn hình chính:



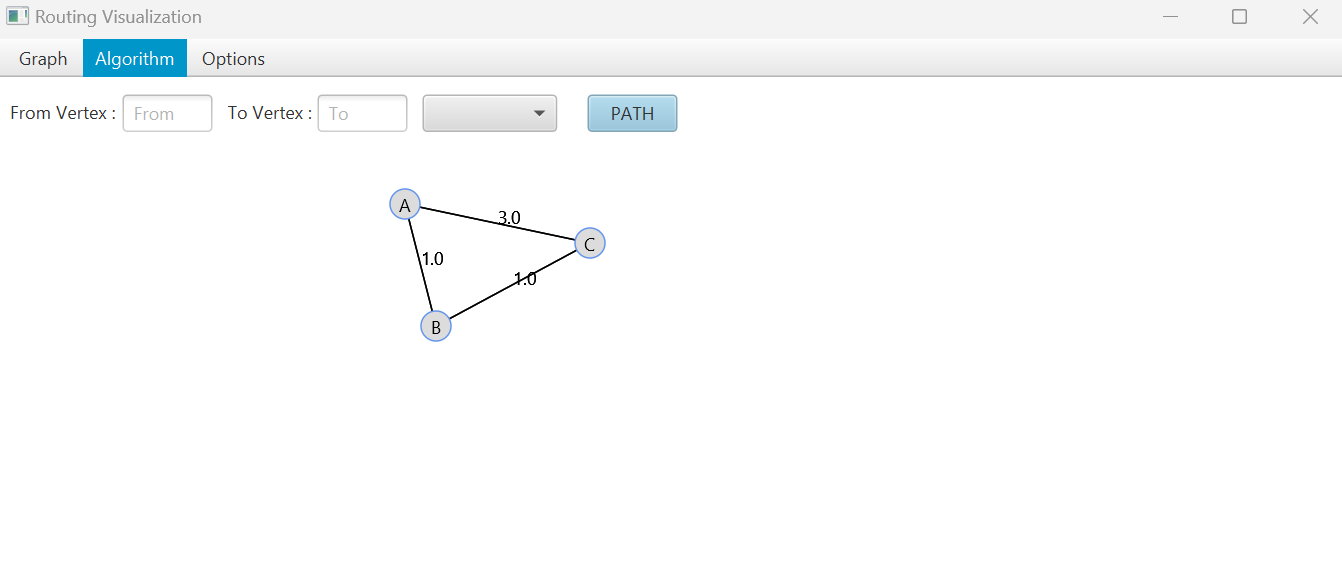
Các chức năng tương tác có trong giao diện: Người dùng nhập Node đầu và cuối, sau đó nhấn Get Path, mà hình sẽ hiển thị Path theo thuật toán Dijkstra.



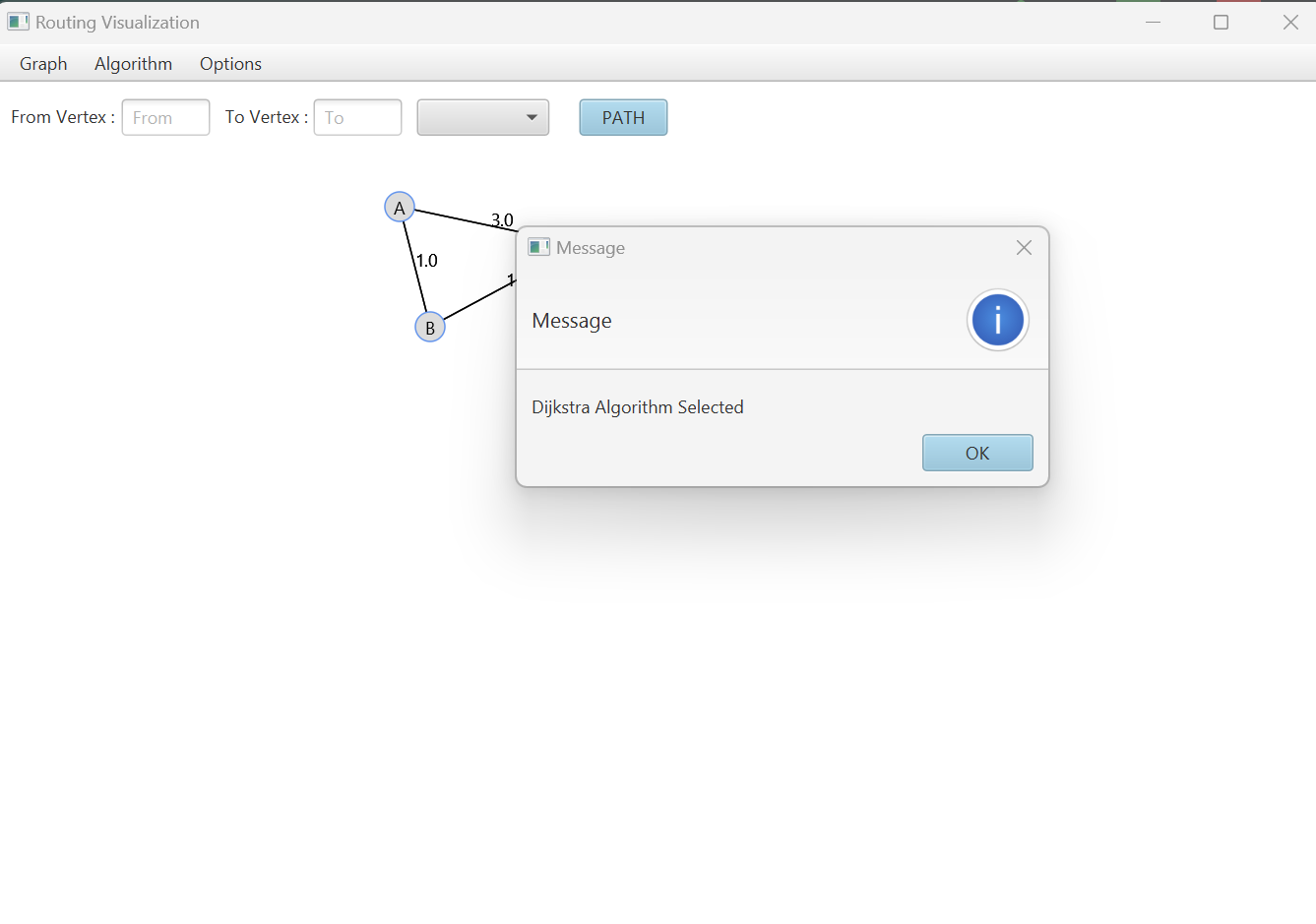


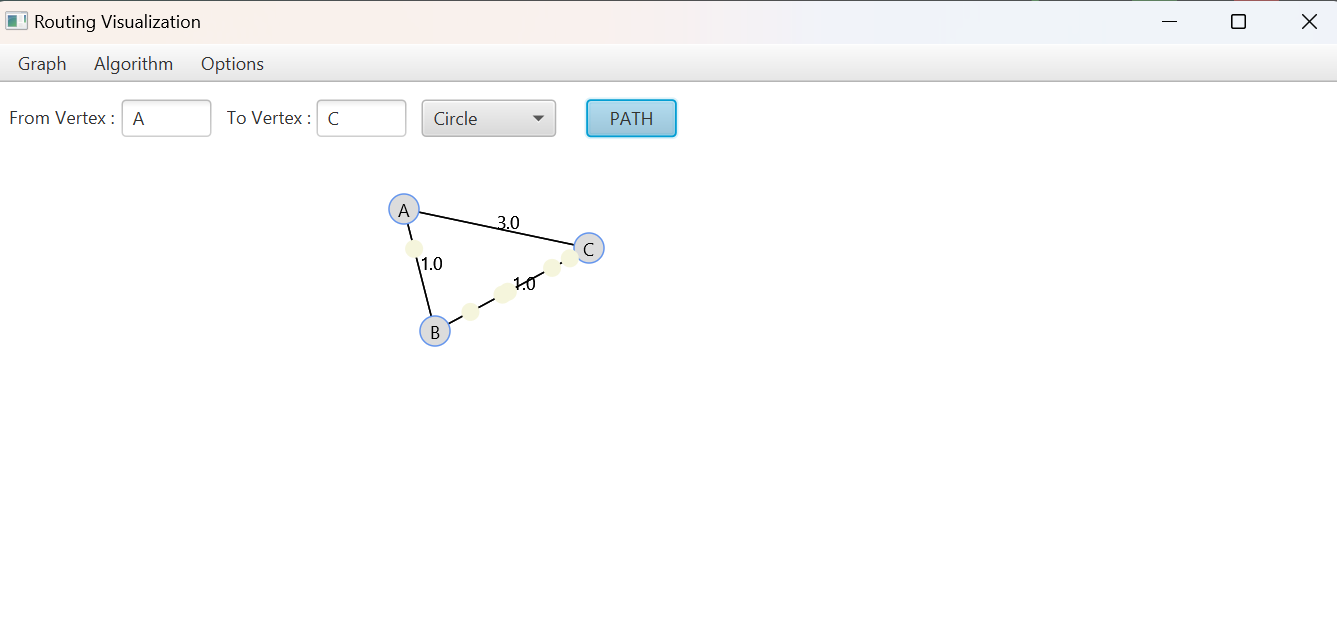
### 4.2.3. Giao diện Algorithm

Màn hình chính:



Các chức năng tương tác có trong giao diện: Bạn chọn thuật toán muốn sử dụng, sau đó nhập các Node bắt đầu và đích, sau đó nhấn PATH, kết quả sẽ hiện ra.





## 4.3. Kiểm thử các chức năng đã thực hiện

| Chức năng vẽ đồ thị (Show Path) | Hoạt động |
| --- | --- |
| Chức năng tìm Path (Get Path) | Hoạt động |
| Chức năng tìm Path theo thuật toán Dijkstra | Hoạt động |
| Chức năng tìm Path theo thuật toán Bellman-Ford | Hoạt động |
| Chức năng tìm Path theo thuật toán Flooding | Hoạt động |

# KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

Chương trình Minh họa Thuật toán Định tuyến gói tin tổng quan được phát triển dựa trên các kiến thức đã được giảng dạy trên lớp và tự tìm hiểu nên còn sơ sài, tồn tại nhiều thiếu sót và hạn chế, một phần cũng do các thành viên trong nhóm chưa có kinh nghiệm xây dựng và phát triển ứng dụng. Sau quá trình phát triển và thử nghiệm ứng dụng, nhóm chúng em xin đưa ra kết luận và hướng phát triển như sau:

1. Ưu điểm:

* Đáp ứng, hoàn thành đầy đủ các yêu cầu của đề tài, các chức năng cơ bản.
* Ứng dụng chạy mượt mà và không gặp các lỗi như crash ứng dụng hay lỗi đồ hoạ.

2. Nhược điểm:

* Đồ hoạ chưa thực sự bắt mắt, chức năng còn sơ sài.

3. Hướng phát triển:

Với những ưu nhược điểm như trên nhóm có một vài ý tưởng phát triển mới cho ứng dụng đó là:

* Thêm các thao tác cập nhật ảnh cho từng cán bộ
* Phát triển thao tác có thể lưu trữ được cơ sở dữ liệu.
* Cải thiện giao diện đồ hoạ thân thiện và dễ dàng tiếp cận hơn.
* Sửa các lỗi sẵn có.

# TÀI LIỆU THAM KHẢO

1. Bài giảng học phần Lập trình hướng đối tượng của cô Nguyễn Thị Thu Trang
2. Lập trình hướng đối tượng với Java – Đoàn Văn Ban – NXB Khoa học và kỹ thuật.
3. Phân tích và thiết kế hệ thống thông tin - Nguyễn Văn Ba.
4. Các video, tài liệu về lập trình hướng đối tượng trên Youtube.