BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC NHA TRANG
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO THỰC TẬP CƠ SỞ

MÔ PHỎNG THUẬT TOÁN TÌM ĐƯỜNG ĐI NGẮN NHẤT TRÊN ĐỒ THỊ

Giảng viên hướng dẫn: **Th.S Bùi Chí Thành**
Sinh viên thực hiện: **Trần Thanh Trí**
Mã số sinh viên: **64132675**

Khánh Hoà – 2025

BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC NHA TRANG

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO THỰC TẬP CƠ SỞ

MÔ PHỎNG THUẬT TOÁN TÌM ĐƯỜNG ĐI NGẮN NHẤT TRÊN ĐỒ THỊ

GVHD: Th.S Bùi Chí Thành

SVTH: Trần Thanh Trí

MSSV: 64132675

Khánh Hoà – Tháng 1/2025

TRƯỜNG ĐẠI HỌC NHA TRANG

KHOA CÔNG NGHỆ THÔNG TIN

PHIẾU THEO DÕI TIẾN ĐỘ VÀ ĐÁNH GIÁ THỰC TẬP CƠ SỞ

(Dùng cho CBHD và nộp cùng báo cáo thực tập cơ sở của sinh viên)

Tên đề tài: Mô phỏng thuật toán tìm đường đi ngắn nhất trên đồ thị

Giảng viên hướng dẫn: Th.S. Bùi Chí Thành

Sinh viên được hướng dẫn: Trần Thanh Trí **MSSV:** 64132675

Khóa: 64 **Ngành:** Công nghệ thông tin

Lần KT	***Ngày***	***Nội dung***	***Nhận xét của GVHD***
1	07/12/2024	Hoàn thành việc xây dựng cơ bản giao diện mô phỏng	
2	14/12/2024	Tìm hiểu chuyên sâu các thuật toán mô phỏng	
3	21/12/2024	Cài đặc mô phỏng các thuật toán	
4	28/12/2024	Hoàn thiện và sửa lỗi các tính năng của sản phẩm	
5	06/01/2024	Hoàn thiện báo cáo và thêm tính năng nhập tay và điều chỉnh tốc độ	
6			

Nhận xét chung (sau khi sinh viên hoàn thành thực tập cơ sở):

……..……

……..……

……..……

Điểm hình thức:……/10 Điểm nội dung:...../10 **Điểm tổng kết**:……/10

Kết luận sinh viên: Được bảo vệ: ☐ Không được bảo vệ: ☐

Khánh Hòa, ngày…….tháng…….năm……

Cán bộ hướng dẫn

(Ký và ghi rõ họ tên)

Bùi Chí Thành

TRƯỜNG ĐẠI HỌC NHA TRANG

KHOA CÔNG NGHỆ THÔNG TIN

PHIẾU CHẤM ĐIỂM THỰC TẬP CƠ SỞ

(Dành cho cán bộ chấm phản biện)

1. Họ tên người chấm: .....

2. Sinh viên thực hiện: ..... **MSSV:** .....

3. Tên đề tài: .....

.....

4. Nhận xét

a) Kết quả thực hiện đề tài

.....

.....

.....

b) Báo cáo thực tập

- Hình thức: .....

.....

.....

- Nội dung: .....

.....

.....

.....

Điểm hình thức:……/10 Điểm nội dung:...../10 **Điểm tổng kết**:……/10

Khánh Hòa, ngày…….tháng…….năm……

Cán bộ chấm phản biện

(Ký và ghi rõ họ tên)

MỤC LỤC

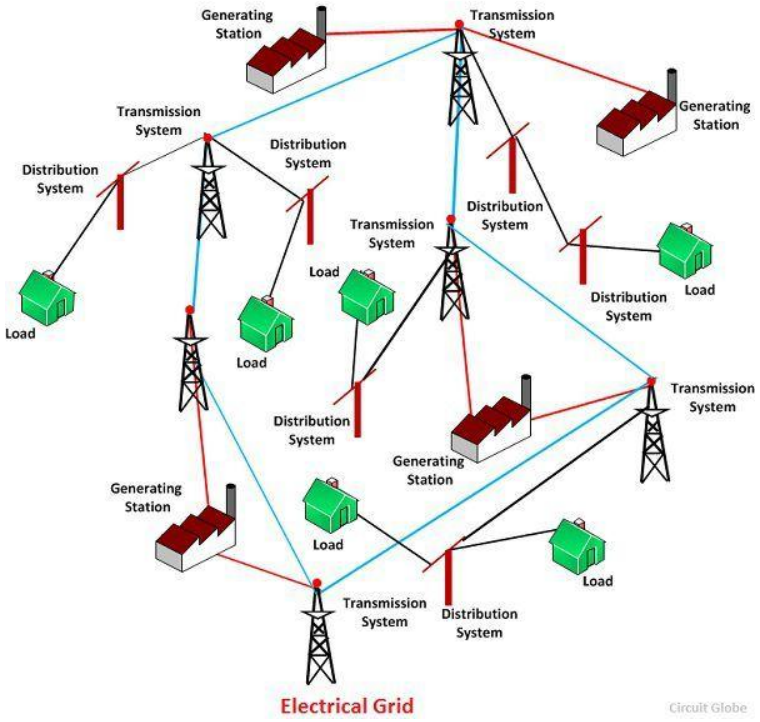
DANH MỤC HÌNH ẢNH – ĐỒ THỊ

Chương 1. TỔNG QUAN VỀ ĐỀ TÀI

1.1. Giới thiệu về đề tài

1.1.1. Bối cảnh

Trong thời đại công nghệ số hiện nay, đồ thị được ứng dụng rộng rãi trong nhiều lĩnh vực khác nhau của đời sống. Từ mạng xã hội, hệ thống giao thông, mạng lưới điện, cho đến các ứng dụng trong khoa học máy tính và trí tuệ nhân tạo, đồ thị đóng vai trò quan trọng trong việc biểu diễn và phân tích các mối quan hệ giữa các đối tượng. Một trong những bài toán cơ bản và quan trọng trên đồ thị là bài toán tìm đường đi ngắn nhất. Bài toán này có ứng dụng rất lớn trong thực tế, ví dụ như tìm đường đi ngắn nhất giữa hai điểm trên bản đồ, tìm đường truyền dữ liệu tối ưu trong mạng máy tính, hay lên kế hoạch vận chuyển hàng hóa hiệu quả.



Hình 1. Ứng dụng của đồ thị trong mạng lưới điện

1.1.2. Lý do chọn đề tài

Đề tài "Mô phỏng thuật toán tìm đường đi ngắn nhất trên đồ thị" được lựa chọn vì những lý do sau:

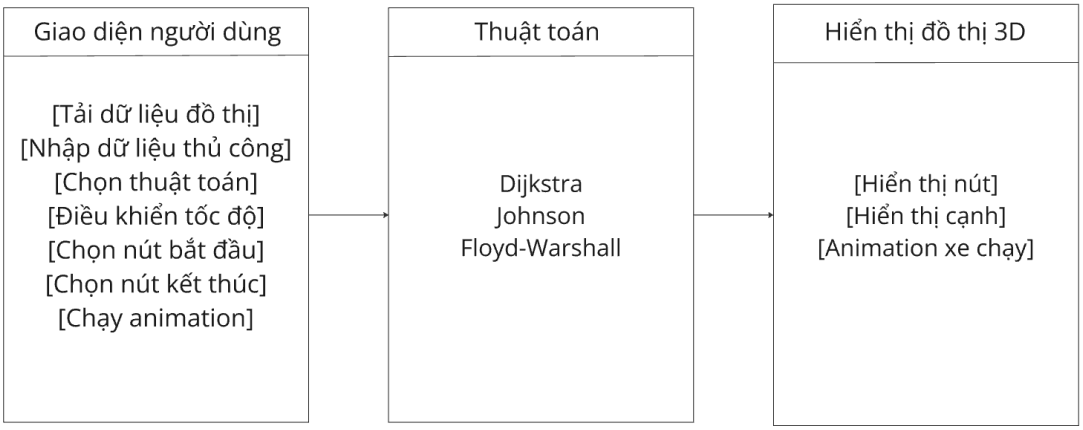
Tính thực tiễn cao: Bài toán tìm đường đi ngắn nhất có ứng dụng rộng rãi trong nhiều lĩnh vực, từ đời sống hàng ngày đến các hệ thống phức tạp. Việc nghiên cứu và mô phỏng các thuật toán tìm đường đi ngắn nhất giúp hiểu rõ hơn về cách thức hoạt động của chúng và áp dụng vào thực tế.

Khả năng phát triển: Đề tài này có thể được mở rộng và phát triển thêm bằng cách nghiên cứu các thuật toán nâng cao, áp dụng cho các loại đồ thị khác nhau, hoặc kết hợp với các kỹ thuật học máy để giải quyết các bài toán phức tạp hơn.

Tính trực quan: Mô phỏng đồ thị và thuật toán bằng công cụ Three.js giúp hình dung rõ ràng cách thức hoạt động của thuật toán, từ đó dễ dàng tiếp cận và hiểu bài toán hơn.

1.1.3. Khái quát về nội dung đề tài

Đề tài tập trung vào việc nghiên cứu và mô phỏng ba thuật toán tìm đường đi ngắn nhất phổ biến: Dijkstra, Johnson và Floyd-Warshall. Báo cáo sẽ trình bày cơ sở lý thuyết về đồ thị và các thuật toán, sau đó đi vào chi tiết cài đặt ứng dụng mô phỏng bằng Three.js. Ứng dụng cho phép người dùng tạo đồ thị, chọn thuật toán và quan sát trực quan quá trình tìm đường đi ngắn nhất.



Hình 2. Sơ đồ khối mô tả cấu trúc tổng quan của ứng dụng

1.2. Mục tiêu của đề tài

1.2.1. Mục tiêu tổng quát

Mục tiêu tổng quát của đề tài là xây dựng một ứng dụng mô phỏng trực quan các thuật toán tìm đường đi ngắn nhất trên đồ thị bằng thư viện Three.js. Ứng dụng này sẽ giúp người dùng dễ dàng hình dung và hiểu rõ hơn về cách thức hoạt động của các thuật toán Dijkstra, Johnson và Floyd-Warshall. Đồng thời, đề tài cũng góp phần nâng cao kỹ năng lập trình, đặc biệt là kỹ năng làm việc với đồ họa 3D và thư viện Three.js.

1.2.2. Mục tiêu cụ thể

Để đạt được mục tiêu tổng quát, đề tài đặt ra các mục tiêu cụ thể sau:

- Nghiên cứu và trình bày cơ sở lý thuyết về đồ thị, bài toán tìm đường đi ngắn nhất và các thuật toán Dijkstra, Johnson, Floyd-Warshall.

- Thiết kế và cài đặt giao diện người dùng thân thiện, cho phép người dùng tạo đồ thị, tùy chỉnh các thuộc tính của đồ thị, lựa chọn thuật toán và chạy mô phỏng.
- Cài đặt các thuật toán Dijkstra, Johnson và Floyd-Warshall trong JavaScript.
- Sử dụng thư viện Three.js để hiển thị đồ thị dưới dạng mô hình 3D, bao gồm các nút, cạnh và nhãn.
- Tạo hiệu ứng animation trực quan cho quá trình tìm kiếm đường đi ngắn nhất, giúp người dùng dễ dàng theo dõi và hiểu rõ thuật toán.
- Đánh giá hiệu quả của ứng dụng thông qua việc thử nghiệm và so sánh kết quả với các tài liệu tham khảo.

Thông qua việc đạt được các mục tiêu cụ thể này, đề tài sẽ góp phần hoàn thành mục tiêu tổng quát và mang lại những kiến thức bổ ích về lý thuyết đồ thị và kỹ năng thực hành lập trình.

1.3. Cấu trúc báo cáo

Báo cáo được chia thành ba chương chính, mỗi chương tập trung vào một khía cạnh cụ thể của đề tài:

Chương 1: Tổng quan về đề tài. Chương này giới thiệu về bối cảnh, lý do chọn đề tài, mục tiêu và phạm vi nghiên cứu. Đồng thời, chương này cũng trình bày tổng quan về cấu trúc của toàn bộ báo cáo, giúp người đọc nắm bắt được nội dung chính của từng chương.

Chương 2: Cơ sở lý thuyết. Chương này trình bày những kiến thức nền tảng về đồ thị và thuật toán tìm đường đi ngắn nhất. Các khái niệm cơ bản về đồ thị, các loại đồ thị, cách biểu diễn đồ thị sẽ được giới thiệu. Tiếp theo, chương này sẽ đi sâu vào phân tích ba thuật toán tìm đường đi ngắn nhất được sử dụng trong đề tài, bao gồm Dijkstra, Johnson và Floyd-Warshall.

Chương 3: Cài đặt ứng dụng. Chương này tập trung vào quá trình cài đặt ứng dụng mô phỏng, từ việc lựa chọn công cụ và môi trường phát triển, thiết kế giao diện người dùng, đến cài đặt các thuật toán và hiển thị kết quả. Chương này cũng sẽ hướng dẫn cách sử dụng ứng dụng để người dùng có thể tự mình khám phá và trải nghiệm.

Chương 2. CƠ SỞ LÝ THUYẾT

2.1. Đồ thị

2.1.1 Định nghĩa

Đồ thị là một cấu trúc toán học dùng để biểu diễn các mối quan hệ giữa các đối tượng. Một đồ thị G được định nghĩa bởi hai tập hợp: tập hợp các đỉnh (vertex) V và tập hợp các cạnh (edge) E. Mỗi cạnh nối hai đỉnh với nhau.

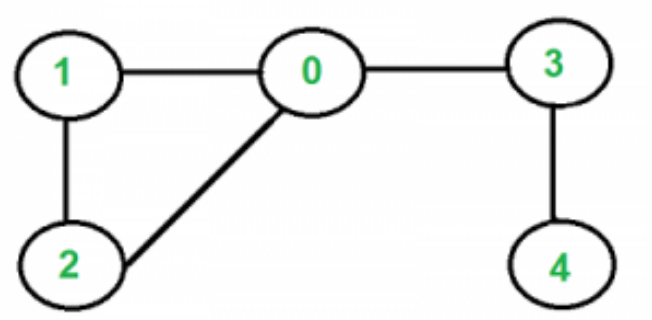
Ký hiệu: G = (V, E)

Ví dụ: Ta có thể biểu diễn một mạng lưới giao thông bằng một đồ thị, trong đó các đỉnh là các thành phố và các cạnh là các tuyến đường nối giữa các thành phố.

Tùy thuộc vào đặc điểm của các cạnh, đồ thị có thể được phân loại thành các loại khác nhau như vô hướng, có hướng, có trọng số. Việc lựa chọn loại đồ thị phù hợp phụ thuộc vào bài toán cụ thể cần giải quyết.

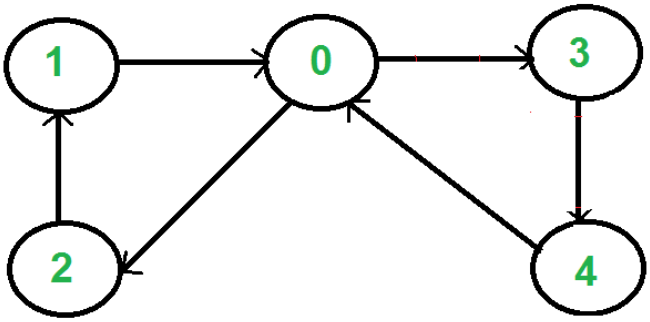
2.1.2. Các loại đồ thị

- **Đồ thị vô hướng (Undirected Graph):** Mỗi cạnh trong đồ thị vô hướng không có hướng, thể hiện mối quan hệ hai chiều giữa hai đỉnh. Ví dụ, một mạng lưới đường bộ có thể được biểu diễn bằng đồ thị vô hướng, với các thành phố là đỉnh và các con đường là cạnh.



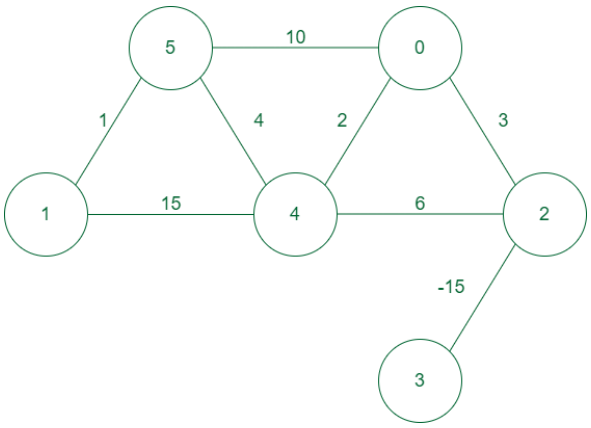
Hình 3. Đồ thị vô hướng

- **Đồ thị có hướng (Directed Graph):** Ngược lại với đồ thị vô hướng, mỗi cạnh trong đồ thị có hướng có một hướng xác định, thể hiện mối quan hệ một chiều giữa hai đỉnh. Ví dụ, một sơ đồ luồng công việc có thể được biểu diễn bằng đồ thị có hướng, với các công việc là đỉnh và các mũi tên thể hiện thứ tự thực hiện các công việc là cạnh.



Hình 4. Đồ thị có hướng

- **Đồ thị có trọng số (Weighted Graph):** Trong đồ thị có trọng số, mỗi cạnh được gán thêm một giá trị số, được gọi là trọng số. Trọng số này có thể biểu thị khoảng cách, chi phí, thời gian, hoặc bất kỳ đại lượng nào liên quan đến mối quan hệ giữa hai đỉnh. Ví dụ, trong một bản đồ đường đi, trọng số của mỗi cạnh có thể là khoảng cách giữa hai địa điểm.



Hình 5. Đồ thị có trọng số

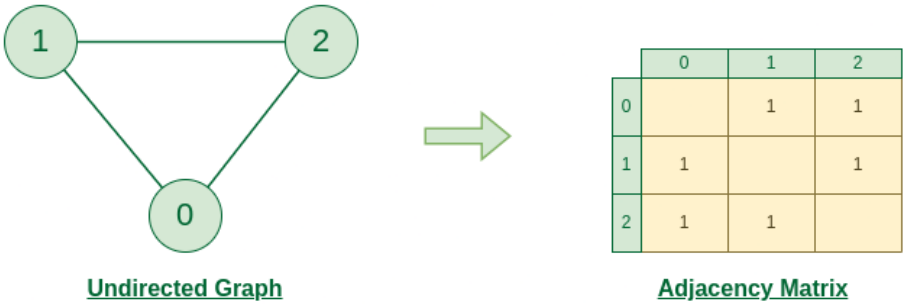
Ngoài ra còn có các loại đồ thị đặc biệt khác như đồ thị đầy đủ, đồ thị liên thông, cây... Việc hiểu rõ các loại đồ thị khác nhau sẽ giúp lựa chọn loại đồ thị phù hợp để biểu diễn và giải quyết các bài toán cụ thể.

2.1.3. Biểu diễn đồ thị

Để biểu diễn đồ thị trên máy tính, có hai cách phổ biến:

- **Ma trận kề (Adjacency Matrix):** Ma trận kề là một ma trận vuông, trong đó mỗi hàng và mỗi cột tương ứng với một đỉnh của đồ thị. Giá trị tại hàng i, cột j của ma trận thể hiện mối quan hệ giữa đỉnh i và đỉnh j. Ưu điểm của ma trận kề là đơn giản, dễ cài đặt và truy xuất thông tin nhanh chóng. Tuy nhiên, nhược điểm là tốn nhiều bộ nhớ, đặc biệt là với đồ thị có số lượng cạnh ít (đồ thị thưa).
- **Danh sách kề (Adjacency List):** Danh sách kề sử dụng một mảng để lưu trữ các đỉnh của đồ thị. Mỗi phần tử trong mảng là một danh sách liên kết chứa các đỉnh kề

với đỉnh đó. Ưu điểm của danh sách kề là tiết kiệm bộ nhớ hơn ma trận kề, đặc biệt là với đồ thị thưa. Tuy nhiên, việc truy xuất thông tin có thể chậm hơn so với ma trận kề.



Hình 6. Biểu diễn đồ thị của ma trận kề đồ thị vô hướng

2.2. Bài toán tìm đường đi ngắn nhất

2.2.1. Phát biểu bài toán

Ứng dụng mô phỏng được phát triển trong khuôn khổ đề tài này tập trung vào việc giải quyết bài toán tìm đường đi ngắn nhất trên một đồ thị có trọng số. Trong đồ thị này, mỗi đỉnh đại diện cho một ngôi nhà, và các cạnh nối giữa các đỉnh biểu thị cho đường đi giữa các ngôi nhà đó. Trọng số được gán cho mỗi cạnh tương ứng với khoảng cách giữa hai ngôi nhà mà cạnh đó kết nối.

Bài toán được phát biểu một cách cụ thể như sau: cho trước hai ngôi nhà (tương ứng với hai đỉnh trên đồ thị), xác định đường đi ngắn nhất giữa hai ngôi nhà đó. Đường đi ngắn nhất được định nghĩa là đường đi có tổng trọng số (tổng khoảng cách di chuyển) nhỏ nhất.

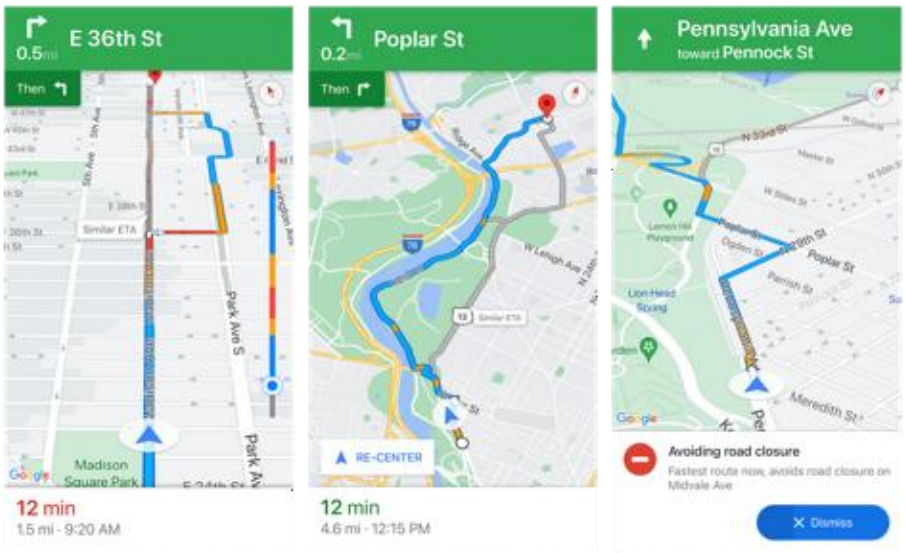
Để giải quyết bài toán này, ứng dụng cung cấp cho người dùng khả năng lựa chọn hai ngôi nhà bất kỳ trên đồ thị và sau đó hiển thị đường đi ngắn nhất giữa chúng. Quá trình này được thực hiện thông qua việc áp dụng các thuật toán tìm đường đi ngắn nhất, cụ thể là Dijkstra, Johnson và Floyd-Warshall, mà chúng ta sẽ tìm hiểu chi tiết trong các phần tiếp theo của báo cáo.

2.2.2. Ứng dụng

Mặc dù ứng dụng được xây dựng chủ yếu nhằm mục đích mô phỏng và minh họa các thuật toán tìm đường đi ngắn nhất, bài toán này có một vai trò vô cùng quan trọng và được ứng dụng rộng rãi trong nhiều lĩnh vực của đời sống. Dưới đây là một số ví dụ điển hình:

- **Ứng dụng bản đồ và định vị:** Các ứng dụng bản đồ trực tuyến như Google Maps, Apple Maps, Waze, v.v. sử dụng thuật toán tìm đường đi ngắn nhất để xác định lộ trình tối ưu giữa hai điểm do người dùng chỉ định. Các yếu tố được đưa vào tính toán

bao gồm khoảng cách địa lý, thời gian di chuyển dự kiến, tình trạng giao thông thực tế, loại hình đường sá (cao tốc, đường trong đô thị, đường nông thôn), và các tùy chọn của người dùng như tránh đường có thu phí, ưu tiên đường cao tốc, v.v.



Hình 7. Ứng dụng tìm đường đi ngắn nhất trong ứng dụng bản đồ

- **Logistics và vận tải:** Trong lĩnh vực vận tải và logistics, việc tối ưu hóa lộ trình vận chuyển hàng hóa là một yếu tố then chốt để giảm thiểu chi phí và thời gian giao hàng. Thuật toán tìm đường đi ngắn nhất được áp dụng để xác định lộ trình vận chuyển hiệu quả nhất, cân nhắc đến các yếu tố như khoảng cách, trọng tải, loại hình phương tiện, điều kiện đường sá, và các ràng buộc về thời gian.
- **Mạng máy tính:** Internet và các mạng máy tính khác hoạt động dựa trên việc truyền dữ liệu giữa các nút mạng (máy tính, router, server).Thuật toán tìm đường đi ngắn nhất được sử dụng để định tuyến các gói tin dữ liệu theo con đường tối ưu, đảm bảo tốc độ truyền tải nhanh chóng và hiệu quả. Các giao thức định tuyến như OSPF (Open Shortest Path First) sử dụng thuật toán Dijkstra để xác định đường đi ngắn nhất giữa các router trong mạng.
- **Thiết kế vi mạch:** Trong quá trình thiết kế vi mạch điện tử, việc bố trí các thành phần (transistor, cổng logic, v.v.) trên chip sao cho các kết nối giữa chúng ngắn nhất có thể là rất quan trọng. Điều này giúp giảm thiểu độ trễ tín hiệu, tiêu thụ năng lượng, và tăng hiệu suất hoạt động của chip. Thuật toán tìm đường đi ngắn nhất được sử dụng để tối ưu hóa việc bố trí các thành phần trên chip.
- **Lĩnh vực game:** Trong phát triển game, thuật toán tìm đường đi ngắn nhất được sử dụng để lập trình trí tuệ nhân tạo (AI) cho các nhân vật trong game, giúp chúng di

chuyển một cách thông minh trong môi trường ảo, tránh chướng ngại vật, và tìm đường đến đích một cách hiệu quả.

Ứng dụng mô phỏng được phát triển trong đề tài này sử dụng thư viện Three.js để hiển thị đồ thị dưới dạng mô hình 3D, tạo ra một môi trường trực quan và sinh động. Các ngôi nhà được biểu diễn bằng các mô hình 3D, và đường đi giữa chúng được hiển thị dưới dạng các đường nối. Người dùng có thể tương tác với mô hình, chọn hai ngôi nhà bất kỳ, và quan sát đường đi ngắn nhất được tính toán bởi ba thuật toán Dijkstra, Johnson và Floyd-Warshall. Ứng dụng này không chỉ minh họa cách thức hoạt động của các thuật toán mà còn giúp người dùng hiểu rõ hơn về ứng dụng của bài toán tìm đường đi ngắn nhất trong thực tế.

2.3. Thuật toán Dijkstra

Thuật toán Dijkstra là một trong những thuật toán cốt lõi được ứng dụng trong project này để giải quyết bài toán tìm đường đi ngắn nhất. Được phát triển bởi Edsger W. Dijkstra vào năm 1956, thuật toán này nổi tiếng với tính hiệu quả và khả năng ứng dụng rộng rãi trong việc tìm kiếm đường đi tối ưu trên đồ thị có trọng số không âm.

2.3.1. Ý tưởng

Thuật toán Dijkstra hoạt động dựa trên nguyên lý tham lam, bằng cách liên tục lựa chọn đỉnh "tốt nhất" tại mỗi bước. "Tốt nhất" ở đây được hiểu là đỉnh có khoảng cách ngắn nhất tạm thời từ đỉnh nguồn đến đỉnh đó. Quá trình này có thể được hình dung như việc "mở rộng dần" tập hợp các đỉnh đã biết đường đi ngắn nhất, bắt đầu từ đỉnh nguồn và lan tỏa ra toàn bộ đồ thị.

Cụ thể hơn, thuật toán Dijkstra thực hiện các bước sau:

- **Khởi tạo:** Gán cho mỗi đỉnh một nhãn, biểu thị khoảng cách ngắn nhất tạm thời từ đỉnh nguồn đến đỉnh đó. Đỉnh nguồn sẽ có nhãn bằng 0, trong khi tất cả các đỉnh còn lại được gán nhãn là vô cùng, thể hiện rằng chưa biết đường đi đến chúng.
- **Lựa chọn đỉnh:** Trong mỗi bước lặp, thuật toán chọn ra đỉnh u có nhãn nhỏ nhất trong số các đỉnh chưa được đưa vào tập hợp các đỉnh đã biết đường đi ngắn nhất.
- **Cập nhật nhãn:** Sau khi chọn đỉnh u, thuật toán cập nhật nhãn của tất cả các đỉnh v kề với u. Nếu khoảng cách từ đỉnh nguồn đến v thông qua u nhỏ hơn nhãn hiện tại của v, thì nhãn của v sẽ được cập nhật bằng giá trị mới này.
- **Lặp lại:** Các bước 2 và 3 được lặp lại cho đến khi tất cả các đỉnh đều được đưa vào tập hợp các đỉnh đã biết đường đi ngắn nhất.

2.3.2. Ưu điểm

- **Tính hiệu quả:** Thuật toán Dijkstra có độ phức tạp thời gian tương đối thấp, đặc biệt khi được cài đặt với cấu trúc dữ liệu heap để tối ưu hóa việc tìm đỉnh có nhãn nhỏ nhất.
- **Tính đơn giản:** Thuật toán dễ hiểu và cài đặt, thích hợp cho việc giảng dạy và học tập.
- **Tính ứng dụng:** Thuật toán Dijkstra được sử dụng rộng rãi trong nhiều ứng dụng thực tế, như đã đề cập ở phần 2.2.2.

2.3.3. Nhược điểm

- **Hạn chế về trọng số:** Thuật toán không hoạt động chính xác trên đồ thị có trọng số âm.
- **Hiệu suất với đồ thị lớn:** Mặc dù hiệu quả với hầu hết các đồ thị, hiệu suất của thuật toán có thể giảm khi xử lý các đồ thị có số lượng đỉnh và cạnh cực lớn.

2.4. Thuật toán Johnson

Thuật toán Johnson, được phát triển bởi Donald B. Johnson vào năm 1977, là một thuật toán hiệu quả để giải quyết bài toán tìm đường đi ngắn nhất giữa tất cả các cặp đỉnh trong một đồ thị có hướng, có trọng số và không chứa chu trình âm. Thuật toán này kết hợp thuật toán Bellman-Ford và thuật toán Dijkstra để đạt được hiệu suất tối ưu.

2.4.1. Ý tưởng

Ý tưởng chính của thuật toán Johnson là biến đổi đồ thị ban đầu thành một đồ thị mới có trọng số không âm, sau đó áp dụng thuật toán Dijkstra trên đồ thị mới này. Các bước chính của thuật toán bao gồm:

- **Thêm một đỉnh mới:** Thêm một đỉnh mới (ký hiệu là q) vào đồ thị và tạo các cạnh có hướng từ đỉnh q đến tất cả các đỉnh khác trong đồ thị. Trọng số của các cạnh mới này được gán bằng 0.
- **Chạy thuật toán Bellman-Ford:** Áp dụng thuật toán Bellman-Ford từ đỉnh q để tính toán khoảng cách ngắn nhất từ q đến tất cả các đỉnh khác. Nếu phát hiện chu trình âm, thuật toán sẽ dừng lại và báo lỗi.
- **Cập nhật trọng số:** Sử dụng kết quả từ thuật toán Bellman-Ford, cập nhật trọng số của mỗi cạnh (u, v) trong đồ thị ban đầu theo công thức: `w'(u, v) = w(u, v) + h(u) − h(v)`, trong đó w(u, v) là trọng số ban đầu của cạnh (u, v), h(u) là khoảng cách ngắn

nhất từ q đến u, và h(v) là khoảng cách ngắn nhất từ q đến v. Việc cập nhật trọng số này đảm bảo rằng đồ thị mới có trọng số không âm.

- **Chạy thuật toán Dijkstra:** Áp dụng thuật toán Dijkstra trên đồ thị mới (với trọng số đã được cập nhật) cho mỗi đỉnh để tìm đường đi ngắn nhất đến tất cả các đỉnh khác.
- **Điều chỉnh kết quả:** Sau khi chạy thuật toán Dijkstra, điều chỉnh kết quả bằng cách trừ đi hiệu số thế năng giữa hai đỉnh để thu được khoảng cách ngắn nhất trên đồ thị ban đầu.

2.4.2. Ưu điểm

- **Hiệu quả với đồ thị có trọng số âm:** Thuật toán Johnson có thể xử lý đồ thị có trọng số âm, miễn là đồ thị không chứa chu trình âm.
- **Tối ưu cho bài toán tất cả các cặp đỉnh:** Thuật toán này hiệu quả hơn so với việc chạy thuật toán Dijkstra nhiều lần cho từng đỉnh khi cần tìm đường đi ngắn nhất giữa tất cả các cặp đỉnh.

2.4.3. Nhược điểm

- **Độ phức tạp cao hơn Dijkstra:** Do sử dụng kết hợp cả thuật toán Bellman-Ford và Dijkstra, độ phức tạp thời gian của thuật toán Johnson cao hơn so với thuật toán Dijkstra.
- **Không hoạt động với đồ thị có chu trình âm:** Thuật toán không thể xử lý đồ thị có chứa chu trình âm, vì trong trường hợp này, không tồn tại đường đi ngắn nhất.

2.5. Thuật toán Floyd-Warshall

Thuật toán Floyd-Warshall là một thuật toán quy hoạch động được sử dụng để tìm đường đi ngắn nhất giữa tất cả các cặp đỉnh trong một đồ thị có trọng số. Thuật toán này có thể áp dụng cho cả đồ thị có hướng và đồ thị vô hướng, và có thể xử lý cả trọng số âm (miễn là không có chu trình âm).

2.5.1. Ý tưởng

Thuật toán Floyd-Warshall hoạt động dựa trên ý tưởng sau: xét lần lượt từng đỉnh k trong đồ thị, kiểm tra xem việc sử dụng đỉnh k làm đỉnh trung gian có tạo ra đường đi ngắn hơn giữa hai đỉnh i và j bất kỳ hay không. Nếu có, thuật toán sẽ cập nhật đường đi ngắn nhất giữa i và j.

Cụ thể hơn, thuật toán Floyd-Warshall xây dựng một ma trận khoảng cách D, trong đó D[i][j] lưu trữ khoảng cách ngắn nhất từ đỉnh i đến đỉnh j. Ban đầu, ma trận D được

khởi tạo bằng trọng số của các cạnh trực tiếp giữa các đỉnh. Sau đó, thuật toán lặp qua tất cả các đỉnh k và cập nhật ma trận D theo công thức:

```
D[i][j] = min(D[i][j], D[i][k] + D[k][j])
```

Công thức này có nghĩa là: khoảng cách ngắn nhất từ i đến j hoặc là bằng khoảng cách hiện tại D[i][j], hoặc là bằng khoảng cách từ i đến k cộng với khoảng cách từ k đến j.

2.5.2. Ưu điểm

- **Đơn giản và dễ cài đặt:** Thuật toán Floyd-Warshall có cách cài đặt đơn giản, dễ hiểu và dễ triển khai.
- **Xử lý trọng số âm:** Thuật toán có thể xử lý đồ thị có trọng số âm, miễn là không có chu trình âm.
- **Tìm đường đi ngắn nhất giữa tất cả các cặp đỉnh:** Thuật toán tìm đường đi ngắn nhất giữa tất cả các cặp đỉnh trong một lần chạy.

2.5.3. Nhược điểm

- **Độ phức tạp thời gian:** Độ phức tạp thời gian của thuật toán là $O(n^3)$, với n là số lượng đỉnh. Điều này làm cho thuật toán kém hiệu quả với đồ thị có số lượng đỉnh lớn.
- **Không gian lưu trữ:** Thuật toán yêu cầu không gian lưu trữ $O(n^2)$ để lưu trữ ma trận khoảng cách, có thể tốn kém bộ nhớ với đồ thị lớn.

Chương 3. CÀI ĐẶT ỨNG DỤNG

Chương này mô tả chi tiết quá trình cài đặt ứng dụng mô phỏng thuật toán tìm đường đi ngắn nhất trên đồ thị. Ứng dụng được phát triển với mục tiêu minh họa trực quan ba thuật toán Dijkstra, Johnson và Floyd-Warshall, đồng thời cung cấp cho người dùng một công cụ tương tác để khám phá và hiểu rõ hơn về cách thức hoạt động của các thuật toán này. Ứng dụng được xây dựng bằng ngôn ngữ lập trình JavaScript, tận dụng thư viện Three.js để tạo ra môi trường 3D sống động và trực quan, và được phát triển trên nền tảng trình soạn thảo mã Visual Studio Code.

3.1. Công cụ và môi trường phát triển

Để hiện thực hóa ứng dụng mô phỏng, chúng tôi đã lựa chọn và sử dụng kết hợp một số công cụ và công nghệ then chốt, bao gồm ngôn ngữ lập trình JavaScript, thư viện đồ họa Three.js, và môi trường phát triển tích hợp Visual Studio Code. Sự kết hợp này mang lại hiệu quả cao trong việc xây dựng ứng dụng, cho phép chúng tôi tạo ra một giao diện người dùng trực quan, dễ sử dụng và hỗ trợ tốt cho việc hiển thị đồ họa 3D.

3.1.1. Ngôn ngữ lập trình JavaScript

JavaScript là một ngôn ngữ lập trình kịch bản, đóng vai trò nền tảng trong phát triển web hiện đại. Với khả năng tương tác cao, JavaScript cho phép tạo ra các trang web động và các ứng dụng web phong phú, mang đến trải nghiệm người dùng tốt hơn. Sự lựa chọn JavaScript cho project này dựa trên những ưu điểm nổi bật sau:

- **Tính phổ biến:** JavaScript là một trong những ngôn ngữ lập trình phổ biến nhất trên thế giới, với một cộng đồng phát triển rộng lớn và nguồn tài nguyên dồi dào. Điều này giúp việc học tập, khắc phục sự cố, và tìm kiếm hỗ trợ trở nên dễ dàng hơn.
- **Tích hợp liền mạch với trình duyệt web:** JavaScript được thiết kế để hoạt động trong trình duyệt web, cho phép tạo ra các ứng dụng tương tác và trực quan mà không cần cài đặt thêm phần mềm.
- **Hỗ trợ đồ họa 3D:** JavaScript có nhiều thư viện mạnh mẽ hỗ trợ xây dựng đồ họa 3D, trong đó Three.js là một lựa chọn phổ biến và hiệu quả.
- **Dễ học và sử dụng:** JavaScript có cú pháp tương đối đơn giản, dễ học và dễ sử dụng, thích hợp cho cả người mới bắt đầu và lập trình viên có kinh nghiệm.

3.1.2. Thư viện Three.js

Three.js là một thư viện JavaScript mã nguồn mở, cung cấp một tập hợp các API đơn giản và trực quan để tạo ra đồ họa 3D trong trình duyệt web. Three.js cho phép render các

hình học 3D, ánh sáng, vật liệu, animation, và nhiều tính năng khác mà không cần phải sử dụng các plugin hoặc phần mềm bên ngoài.

Trong project này, Three.js đóng vai trò quan trọng trong việc:

- **Hiển thị đồ thị:** Three.js được sử dụng để biểu diễn các đỉnh của đồ thị (các ngôi nhà) bằng các mô hình 3D được tải từ các tệp .glb. Các cạnh của đồ thị được vẽ bằng các đường thẳng nối giữa các đỉnh.
- **Tạo hiệu ứng trực quan:** Three.js cho phép thêm các yếu tố môi trường như cây cối, tuyết rơi, và animation cho xe chạy dọc theo đường đi ngắn nhất, tạo nên một môi trường 3D sống động và thu hút.
- **Tương tác người dùng:** Three.js cung cấp các công cụ để người dùng có thể tương tác với mô hình 3D, bao gồm xoay, phóng to, thu nhỏ, và di chuyển camera để quan sát đồ thị từ nhiều góc độ khác nhau.

3.1.3. Trình soạn thảo mã Visual Studio Code

Visual Studio Code (VS Code) là một trình soạn thảo mã nguồn mở, đa nền tảng được phát triển bởi Microsoft. VS Code được thiết kế để cung cấp một môi trường lập trình hiện đại, nhẹ nhàng và hiệu quả, hỗ trợ nhiều ngôn ngữ lập trình và tích hợp nhiều tính năng mạnh mẽ.

Trong quá trình phát triển project, VS Code đã mang lại nhiều lợi ích đáng kể, bao gồm:

- **Hiệu suất cao:** VS Code khởi động nhanh chóng và tiêu thụ ít tài nguyên hệ thống, giúp quá trình lập trình diễn ra mượt mà và không bị gián đoạn.
- **Hỗ trợ đa ngôn ngữ:** VS Code hỗ trợ syntax highlighting, code completion, và debugging cho nhiều ngôn ngữ lập trình khác nhau, bao gồm JavaScript, HTML, CSS, Python, và nhiều ngôn ngữ khác.
- **Khả năng mở rộng:** VS Code có một kho plugin phong phú, cho phép người dùng cài đặt thêm các tính năng và công cụ hỗ trợ cho lập trình, gỡ lỗi, và quản lý mã nguồn. Các plugin phổ biến bao gồm ESLint, Prettier, Live Server, và Debugger for Chrome.
- **Tích hợp Git:** VS Code tích hợp sẵn Git, cung cấp các công cụ trực quan để quản lý mã nguồn và làm việc với các kho lưu trữ Git.

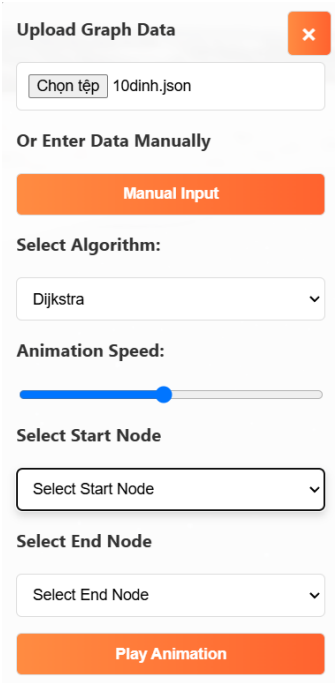
3.2. Thiết kế giao diện

Giao diện người dùng đóng vai trò then chốt trong việc mang đến trải nghiệm người dùng tích cực và hiệu quả. Trong project "Mô phỏng thuật toán tìm đường đi ngắn nhất trên đồ thị", giao diện được thiết kế với mục tiêu tối ưu hóa sự tương tác, giúp người dùng dễ dàng điều khiển, khám phá và hiểu rõ hơn về quá trình mô phỏng.

3.2.1. Cấu trúc giao diện

Ứng dụng được thiết kế với giao diện trực quan, thân thiện, bao gồm hai phần chính:

- **Khu vực điều khiển:** Nằm ở góc trên bên trái màn hình, chứa các nút bấm và thanh trượt để điều khiển các chức năng của ứng dụng.



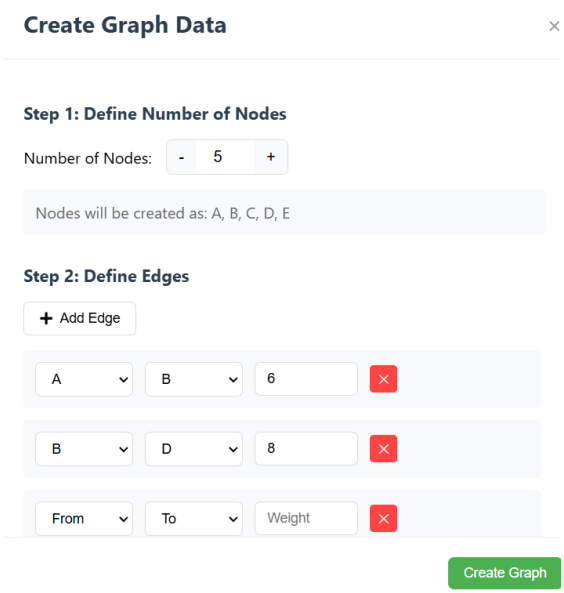
Hình 8. Giao diện điều khiển

- **Khu vực hiển thị đồ thị 3D:** Chiếm phần lớn diện tích màn hình, hiển thị mô hình 3D của đồ thị, bao gồm các ngôi nhà (nút), đường đi (cạnh), cây cối, hiệu ứng tuyết rơi, và animation xe chạy.

3.2.2. Các thành phần giao diện

Khu vực điều khiển được bố trí gọn gàng, tập trung các công cụ chính, cho phép người dùng:

- **Tải lên dữ liệu đồ thị từ file JSON:** Nút tải lên file cho phép người dùng nhập dữ liệu đồ thị từ một file định dạng JSON, bao gồm thông tin về các nút và các cạnh.
- **Tạo đồ thị mới:** Nút "Manual Input" mở ra một cửa sổ cho phép người dùng tạo đồ thị bằng cách nhập số lượng nút, cạnh và trọng số tương ứng.



Hình 9. Giao diện nhập tay đồ thị

- **Lựa chọn thuật toán:** Người dùng có thể lựa chọn một trong ba thuật toán: Dijkstra, Johnson và Floyd-Warshall để thực hiện việc tìm kiếm đường đi ngắn nhất.
- **Điều khiển tốc độ animation:** Thanh trượt cho phép điều chỉnh tốc độ animation của xe chạy dọc theo đường đi ngắn nhất.
- **Lựa chọn nút bắt đầu và kết thúc:** Hai menu dropdown cho phép người dùng chọn nút bắt đầu và kết thúc cho quá trình tìm kiếm.
- **Chạy animation:** Nút "Play Animation" bắt đầu quá trình mô phỏng, hiển thị đường đi ngắn nhất và animation xe chạy trên đồ thị 3D.
- **Ẩn/hiện bảng điều khiển:** Nút "✕" cho phép ẩn/hiện bảng điều khiển để tối ưu không gian hiển thị cho mô hình 3D, tạo sự tập trung cho người dùng khi quan sát.

Khu vực hiển thị đồ thị 3D là nơi đồ thị được thể hiện trực quan, sinh động với:

- **Mô hình 3D:** Các ngôi nhà (nút), đường đi (cạnh), cây cối, hiệu ứng tuyết rơi, và animation xe chạy đều được hiển thị rõ ràng trong không gian 3D.
- **Tương tác:** Người dùng có thể tương tác với mô hình 3D bằng cách xoay, phóng to/thu nhỏ, và di chuyển camera để quan sát từ nhiều góc độ khác nhau.

3.3. Cài đặt thuật toán

Ứng dụng mô phỏng được xây dựng để minh họa ba thuật toán tìm đường đi ngắn nhất: Dijkstra, Johnson và Floyd-Warshall. Mỗi thuật toán được cài đặt trong một hàm riêng biệt trong file graph.js. Các hàm này được gọi từ hàm handlePlayAnimation trong file index.js dựa trên lựa chọn của người dùng trên giao diện.

3.3.1. Cài đặt thuật toán Dijkstra

Trong ứng dụng, thuật toán Dijkstra được hiện thực hóa thông qua hàm dijkstra(graph, start, end) nằm trong file graph.js. Để thực hiện chức năng này, hàm dijkstra nhận vào ba tham số đầu vào:

- **graph:** Đồ thị cần tìm đường đi ngắn nhất, được biểu diễn dưới dạng một đối tượng JavaScript chứa thông tin về các nút và các cạnh.
- **start:** ID của nút bắt đầu tìm kiếm đường đi.
- **end:** ID của nút kết thúc. Tham số này có thể là null nếu muốn tìm đường đi ngắn nhất từ nút bắt đầu đến tất cả các nút khác.

Bên trong hàm dijkstra, ba cấu trúc dữ liệu chính được sử dụng để hỗ trợ quá trình tính toán:

- **distances:** Một đối tượng JavaScript dùng để lưu trữ khoảng cách ngắn nhất từ nút bắt đầu đến các nút khác.
- **previous:** Một đối tượng JavaScript dùng để lưu vết đường đi, nghĩa là với mỗi nút, previous sẽ lưu trữ nút liền trước nó trên đường đi ngắn nhất từ nút bắt đầu.
- **visited:** Một tập hợp (Set) dùng để theo dõi các nút đã được xét trong quá trình tìm kiếm.

Thuật toán được cài đặt bằng cách sử dụng một vòng lặp while để duyệt qua các nút trong đồ thị. Trong mỗi vòng lặp, thuật toán thực hiện các bước sau:

Bước 1: Tìm nút chưa xét có khoảng cách tạm thời nhỏ nhất.

Bước 2: Nếu nút này là nút kết thúc hoặc tất cả các nút đã được xét, kết thúc vòng lặp.

Bước 3: Xóa nút này khỏi hàng đợi và thêm nó vào tập hợp visited.

Bước 4: Xét tất cả các nút kề với nút hiện tại. Với mỗi nút kề, tính toán khoảng cách từ nút bắt đầu đến nút kề thông qua nút hiện tại. Nếu khoảng cách này nhỏ hơn khoảng cách tạm thời hiện tại của nút kề, cập nhật khoảng cách tạm thời và lưu nút hiện tại vào previous.

Kết quả trả về của hàm dijkstra phụ thuộc vào giá trị của tham số end:

- **Nếu end khác null:** Đường đi ngắn nhất từ nút bắt đầu đến nút kết thúc, dưới dạng một mảng các ID nút.
- **Nếu end là null:** Một đối tượng chứa khoảng cách ngắn nhất từ nút bắt đầu đến tất cả các nút khác.

3.3.2. Cài đặt thuật toán Johnson

Hàm johnson(graph) trong file graph.js cài đặt thuật toán Johnson. Hàm này nhận vào đồ thị graph làm tham số và trả về một đối tượng chứa đường đi ngắn nhất giữa tất cả các cặp nút. Thuật toán Johnson được cài đặt theo các bước sau:

Bước 1: Tạo một nút mới q và thêm các cạnh từ q đến tất cả các nút khác với trọng số 0.

Bước 2: Sử dụng thuật toán Bellman-Ford (được gọi gián tiếp thông qua việc kiểm tra chu trình âm) để tính toán khoảng cách ngắn nhất từ q đến tất cả các nút và lưu vào biến h.

Bước 3: Tạo một đồ thị mới với trọng số được cập nhật dựa trên công thức:

```
w'(u, v) = w(u, v) + h(u) - h(v).
```

Bước 4: Sử dụng hàm dijkstraWithPaths (một biến thể của thuật toán Dijkstra) để tính toán đường đi ngắn nhất từ mỗi nút đến tất cả các nút khác trong đồ thị mới.

Bước 5: Điều chỉnh khoảng cách ngắn nhất thu được từ dijkstraWithPaths để có được khoảng cách ngắn nhất trên đồ thị ban đầu.

3.3.3. Cài đặt thuật toán Floyd-Warshall

Hàm floydWarshall(graph) trong file graph.js cài đặt thuật toán Floyd-Warshall. Hàm này nhận vào đồ thị graph làm tham số và trả về một đối tượng chứa hai ma trận: distance (lưu trữ khoảng cách ngắn nhất giữa tất cả các cặp nút) và paths (lưu trữ đường đi ngắn nhất tương ứng).

Các bước cài đặt thuật toán Floyd-Warshall trong hàm floydWarshall bao gồm:

Bước 1: Khởi tạo ma trận khoảng cách distance với kích thước n x n, với n là số lượng nút trong đồ thị.

Bước 2: Gán giá trị ban đầu cho các phần tử của ma trận distance:

- Distance[i][i] = 0 (khoảng cách từ một nút đến chính nó bằng 0).
- Distance[i][j] bằng trọng số của cạnh từ nút i đến nút j nếu có cạnh nối giữa chúng.
- Distance[i][j] = Infinity nếu không có cạnh nối từ nút i đến nút j.

Bước 3: Khởi tạo ma trận next với kích thước n x n, ma trận này dùng để lưu trữ thông tin về nút tiếp theo trên đường đi ngắn nhất giữa hai nút bất kỳ.

Bước 4: Gán giá trị ban đầu cho các phần tử của ma trận next:

- Next[i][j] = j nếu có cạnh nối từ nút i đến nút j.

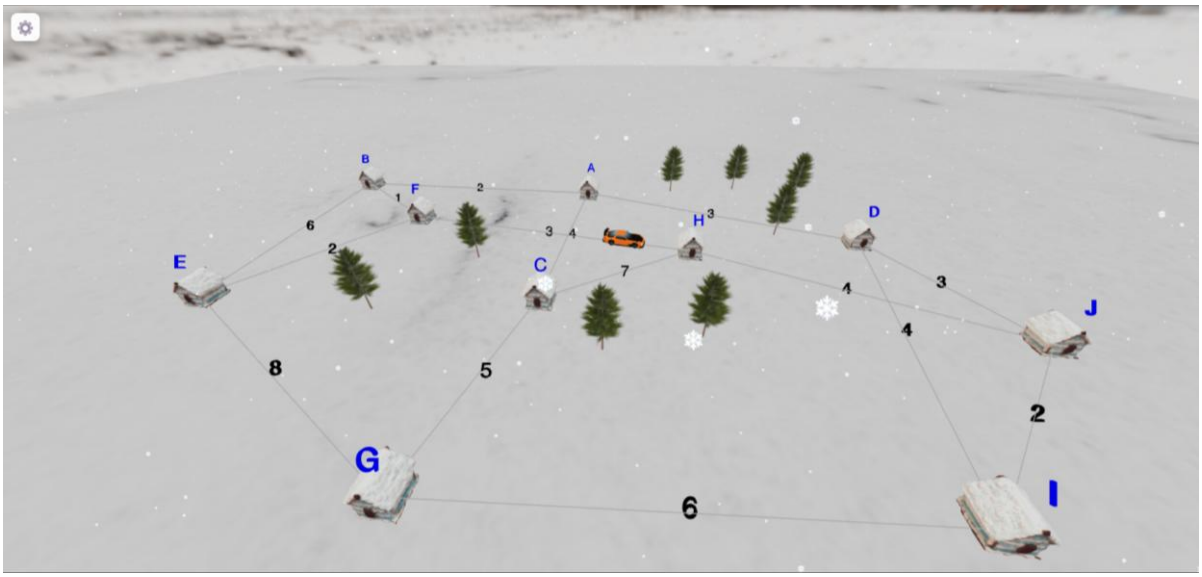
- Next[i][j] = null nếu không có cạnh nối từ nút i đến nút j.

Bước 5: Sử dụng ba vòng lặp lồng nhau để duyệt qua tất cả các nút k (nút trung gian). Trong mỗi vòng lặp, kiểm tra xem sử dụng nút k làm nút trung gian có tạo ra đường đi ngắn hơn giữa hai nút i và j hay không. Nếu có, cập nhật distance[i][j] và next[i][j] tương ứng.

Bước 6: Sau khi hoàn thành việc lặp qua tất cả các nút trung gian, xây dựng ma trận paths dựa trên ma trận next. Mỗi phần tử paths[i][j] sẽ chứa một mảng các ID nút, biểu diễn đường đi ngắn nhất từ nút i đến nút j.

3.4. Hiển thị đồ thị và animation

Ứng dụng sử dụng thư viện Three.js để hiển thị đồ thị dưới dạng mô hình 3D trực quan và sinh động. Việc hiển thị này bao gồm hai thành phần chính: hiển thị các nút và cạnh của đồ thị, và tạo animation cho xe chạy dọc theo đường đi ngắn nhất.



Hình 10. Giao diện mô phỏng đồ họa 3D

3.4.1. Hiển thị nút và cạnh

Các nút của đồ thị, đại diện cho các ngôi nhà, được hiển thị bằng các mô hình 3D được tải từ file house.glb. Để đảm bảo tính trực quan và dễ dàng nhận biết, mỗi nút được gắn nhãn với ID tương ứng. Các nhãn này được tạo bằng TextGeometry và luôn hướng về phía camera để đảm bảo khả năng đọc dù người dùng có xoay mô hình như thế nào. Vị trí của các nút được xác định bởi thuật toán applyForceDirectedLayout trong file dataLoader.js. Thuật toán này sắp xếp các nút sao cho đồ thị được hiển thị một cách cân đối và dễ nhìn.

Các cạnh của đồ thị, biểu diễn cho đường đi giữa các ngôi nhà, được hiển thị bằng các đường thẳng nối giữa các nút. Trọng số của mỗi cạnh, thể hiện khoảng cách giữa hai

ngôi nhà, được hiển thị bằng chữ số nằm giữa cạnh. Tương tự như nhãn của nút, trọng số của cạnh cũng luôn hướng về phía camera để người dùng dễ dàng quan sát.

Việc hiển thị nút và cạnh được thực hiện trong hai hàm displayNodes và displayEdges (file graphVisualizer.js). Hai hàm này sử dụng các đối tượng và phương thức của Three.js để tạo ra các hình học, vật liệu và thêm chúng vào cảnh 3D.

3.4.2. Animation xe chạy

Sau khi người dùng chọn hai nút và thuật toán tìm đường đi ngắn nhất, ứng dụng sẽ thực hiện animation xe chạy từ nút bắt đầu đến nút kết thúc theo đường đi ngắn nhất đã tìm được. Mô hình 3D của xe được tải từ file car.glb.

Để tạo animation mượt mà và chân thực, xe không chỉ di chuyển dọc theo đường đi mà còn xoay theo hướng di chuyển. Quá trình này được thực hiện bằng cách tính toán hướng di chuyển dựa trên điểm hiện tại và điểm tiếp theo trên đường đi, sau đó cập nhật góc xoay của xe. Tốc độ di chuyển của xe có thể được điều chỉnh bởi người dùng thông qua thanh trượt trên giao diện.

Animation xe chạy được thực hiện trong hàm animateCar (file carAnimation.js). Hàm này sử dụng requestAnimationFrame để tạo vòng lặp animation và cập nhật vị trí và góc xoay của xe trong mỗi khung hình.

3.5. Hướng dẫn sử dụng

Ứng dụng mô phỏng thuật toán tìm đường đi ngắn nhất trên đồ thị được thiết kế với giao diện trực quan, thân thiện với người dùng, giúp dễ dàng tương tác và khám phá. Dưới đây là hướng dẫn chi tiết để sử dụng ứng dụng:

3.5.1. Khởi chạy project

Trước tiên, cần chắc chắn rằng đã cài đặt Node.js và npm (hoặc yarn) trên máy tính. Sau đó, mở terminal, di chuyển đến thư mục chứa project và chạy lệnh *npx vite* để khởi động. Lệnh này sẽ khởi động một server phát triển và tự động mở ứng dụng trong trình duyệt web.

Để khởi chạy ứng dụng, trước tiên, đảm bảo rằng Node.js và npm (hoặc yarn) đã được cài đặt trên máy tính, có thể kiểm tra bằng lệnh `node -v` và `npm -v`. Nếu chưa, có thể tải Node.js từ trang chủ Node.js. Sau đó, mở terminal, di chuyển đến thư mục chứa project bằng lệnh `cd <tên-thư-mục-project>` và cài đặt các gói phụ thuộc bằng `npm install` hoặc `yarn install`. Khi đã hoàn tất, khởi động server phát triển bằng lệnh `npx vite`, ứng dụng sẽ tự động mở trên trình duyệt tại địa chỉ mặc định là `http://localhost:5173`. Để

dừng server, nhấn tổ hợp phím `Ctrl + C` trong terminal. Nếu gặp lỗi liên quan đến gói phụ thuộc, chạy lại lệnh `npm install` để khắc phục. Với các bước này, ứng dụng sẽ sẵn sàng để sử dụng.

3.5.2. Tải dữ liệu đồ thị

Ứng dụng cho phép người dùng tải dữ liệu đồ thị từ một tệp JSON. Nhấn nút "Upload Graph Data" và chọn tệp từ máy tính. Tệp JSON cần tuân theo định dạng sau:

```
{
  "nodes": [
    {"id": "A"},
    {"id": "B"},
    {"id": "C"}
  ],
  "edges": [
    {"source": "A", "target": "B", "weight": 10},
    {"source": "A", "target": "C", "weight": 15},
    {"source": "B", "target": "C", "weight": 5}
  ]
}
```

Trong đó:

- **nodes:** Mảng chứa danh sách các nút.
- **edges:** Mảng chứa danh sách các cạnh, mỗi cạnh bao gồm source (nút nguồn), target (nút đích), và weight (trọng số - biểu thị khoảng cách giữa hai nút).

3.5.3. Tạo đồ thị thủ công

Ngoài việc tải dữ liệu từ file, người dùng có thể tạo đồ thị mới trực tiếp trên ứng dụng bằng cách nhấn nút "Manual Input". Một cửa sổ sẽ hiện ra, cho phép người dùng thực hiện các thao tác sau:

- **Chọn số lượng nút:** Sử dụng các nút "+" và "-" hoặc nhập trực tiếp vào ô input để xác định số lượng nút trong đồ thị.
- **Thêm cạnh:** Nhấn nút "Add Edge" để thêm cạnh mới. Người dùng cần chọn nút nguồn, nút đích và nhập trọng số cho cạnh đó.
- **Xóa cạnh:** Nhấn nút "✕" bên cạnh mỗi cạnh để xóa cạnh đó.
- **Tạo đồ thị:** Sau khi hoàn tất việc nhập dữ liệu, nhấn nút "Create Graph" để tạo đồ thị.

3.5.4. Chọn thuật toán

Ứng dụng hỗ trợ ba thuật toán tìm đường đi ngắn nhất: Dijkstra, Johnson và Floyd-Warshall. Người dùng có thể lựa chọn thuật toán mong muốn từ menu dropdown "Select Algorithm".

3.5.5. Chọn nút bắt đầu và kết thúc

Sử dụng hai menu dropdown "Select Start Node" và "Select End Node" để lựa chọn nút bắt đầu và nút kết thúc cho việc tìm kiếm đường đi ngắn nhất.

3.5.6. Chạy animation

Nhấn nút "Play Animation" để bắt đầu mô phỏng. Ứng dụng sẽ hiển thị đường đi ngắn nhất giữa hai nút đã chọn, cùng với animation xe chạy dọc theo đường đi đó.

3.5.7. Điều khiển animation

Tốc độ: Sử dụng thanh trượt "Animation Speed" để điều chỉnh tốc độ animation của xe.

Tương tác với mô hình 3D: Người dùng có thể xoay, phóng to/thu nhỏ, và di chuyển camera để quan sát đồ thị 3D từ nhiều góc độ khác nhau.

Ẩn/hiện bảng điều khiển: Nhấn nút "✕" để ẩn/hiện bảng điều khiển, giúp người dùng có cái nhìn toàn cảnh hơn về mô hình 3D.

3.6. Lưu ý khi sử dụng ứng dụng

Đồ thị đầu vào: Ứng dụng được thiết kế để xử lý các đồ thị có trọng số không âm. Điều này có nghĩa là trọng số của mỗi cạnh (biểu thị khoảng cách giữa hai nút) phải là một số không âm. Nếu dữ liệu đồ thị đầu vào có trọng số âm, thuật toán Dijkstra sẽ không hoạt động chính xác, và kết quả hiển thị có thể không phải là đường đi ngắn nhất.

Chu trình âm: Mặc dù thuật toán Johnson và Floyd-Warshall có thể xử lý đồ thị có trọng số âm, chúng không hoạt động với đồ thị có chu trình âm. Chu trình âm là một chu trình trong đó tổng trọng số các cạnh là âm. Nếu đồ thị đầu vào có chứa chu trình âm, sẽ không tồn tại đường đi ngắn nhất, và ứng dụng sẽ không thể tìm ra kết quả chính xác.

Lựa chọn thuật toán: Hãy lựa chọn thuật toán phù hợp với đặc điểm của đồ thị và mục tiêu.

- Nếu cần tìm đường đi ngắn nhất từ một nút đến tất cả các nút khác, thuật toán Dijkstra là một lựa chọn tốt.

- Nếu cần tìm đường đi ngắn nhất giữa tất cả các cặp nút và đồ thị có thể có trọng số âm (nhưng không có chu trình âm), hãy sử dụng thuật toán Johnson hoặc Floyd-Warshall.

Hiệu suất: Thuật toán Floyd-Warshall có độ phức tạp thời gian cao hơn so với Dijkstra và Johnson, đặc biệt là với đồ thị có số lượng nút lớn. Nếu đang làm việc với một đồ thị rất lớn, hãy cân nhắc sử dụng Dijkstra hoặc Johnson để tối ưu hiệu suất.

Tương tác: Ứng dụng có các tính năng tương tác để khám phá đồ thị và quan sát kết quả mô phỏng. Người dùng có thể xoay, phóng to/thu nhỏ, và di chuyển camera để quan sát đồ thị từ nhiều góc độ khác nhau. Cũng có thể điều chỉnh tốc độ animation để theo dõi quá trình tìm kiếm đường đi ngắn nhất một cách chi tiết.

Chương 4. KẾT LUẬN

4.1. Tóm tắt kết quả đạt được

4.1.1. Về mặt lý thuyết

Đề tài đã hoàn thành việc nghiên cứu và trình bày một cách có hệ thống cơ sở lý thuyết về đồ thị và bài toán tìm đường đi ngắn nhất. Các khái niệm cơ bản về đồ thị, các loại đồ thị, và phương pháp biểu diễn đồ thị đã được trình bày rõ ràng, đầy đủ.

Đặc biệt, đề tài đã đi sâu phân tích ba thuật toán tìm đường đi ngắn nhất, bao gồm Dijkstra, Johnson và Floyd-Warshall. Mỗi thuật toán được trình bày chi tiết về ý tưởng, cách thức hoạt động, ưu điểm và nhược điểm, cũng như so sánh hiệu quả giữa chúng.

Việc nghiên cứu này giúp người đọc nắm vững kiến thức nền tảng về lý thuyết đồ thị và các thuật toán tìm đường đi ngắn nhất, từ đó có thể áp dụng vào các bài toán thực tế.

4.1.2. Về mặt thực hành

Đề tài đã thành công xây dựng một ứng dụng mô phỏng trực quan ba thuật toán tìm đường đi ngắn nhất bằng thư viện Three.js. Ứng dụng có giao diện người dùng thân thiện, cho phép người dùng:

- Tạo đồ thị mới hoặc tải dữ liệu đồ thị từ tệp.
- Lựa chọn thuật toán để tìm đường đi ngắn nhất.
- Chọn nút bắt đầu và kết thúc.
- Quan sát kết quả và animation xe chạy trên mô hình 3D.
- Tương tác với mô hình 3D (xoay, phóng to/thu nhỏ, di chuyển camera).

Ứng dụng này không chỉ minh họa trực quan cách thức hoạt động của các thuật toán, mà còn là một công cụ hữu ích để người dùng tự mình khám phá và hiểu rõ hơn về bài toán tìm đường đi ngắn nhất.

Đề tài cũng góp phần nâng cao kỹ năng lập trình, đặc biệt là kỹ năng xây dựng ứng dụng web tương tác và sử dụng thư viện đồ họa Three.js.

4.2. Hạn chế và hướng phát triển

4.2.1 Hạn chế

Số lượng thuật toán: Đề tài mới chỉ tập trung vào ba thuật toán phổ biến là Dijkstra, Johnson và Floyd-Warshall. Vẫn còn nhiều thuật toán tìm đường đi ngắn nhất khác có thể được nghiên cứu và bổ sung vào ứng dụng, ví dụ như thuật toán A*, Bellman-Ford, thuật toán tham lam,...

Loại đồ thị: Ứng dụng mới chỉ hỗ trợ đồ thị có trọng số không âm. Việc mở rộng để hỗ trợ các loại đồ thị khác, chẳng hạn như đồ thị có hướng, đồ thị có trọng số âm (với điều kiện không có chu trình âm), sẽ nâng cao tính ứng dụng của ứng dụng.

Tối ưu hiệu năng: Đối với đồ thị có số lượng nút và cạnh lớn, hiệu suất của ứng dụng có thể bị giảm sút. Cần nghiên cứu và áp dụng các kỹ thuật tối ưu hiệu năng, chẳng hạn như sử dụng cấu trúc dữ liệu phù hợp, tối ưu thuật toán, và cải thiện cách hiển thị đồ họa.

4.2.2. Hướng phát triển

Mở rộng thư viện thuật toán: Nghiên cứu và cài đặt thêm các thuật toán tìm đường đi ngắn nhất khác như A*, Bellman-Ford, thuật toán tham lam,...

Hỗ trợ nhiều loại đồ thị: Mở rộng ứng dụng để có thể xử lý các loại đồ thị khác nhau, bao gồm đồ thị có hướng, đồ thị có trọng số âm.

Nâng cao hiệu năng: Tối ưu hiệu suất của ứng dụng bằng cách sử dụng cấu trúc dữ liệu phù hợp, tối ưu thuật toán, và cải thiện cách hiển thị đồ họa.

Bổ sung tính năng: Thêm các tính năng mới như tùy chỉnh giao diện, cung cấp thông tin chi tiết về quá trình tìm kiếm, và kết hợp với bản đồ thực tế.

Phát triển ứng dụng thực tế: Ứng dụng các thuật toán tìm đường đi ngắn nhất vào các bài toán thực tế, ví dụ như xây dựng ứng dụng định vị, lập lịch trình vận chuyển, hoặc tối ưu hóa mạng lưới giao thông.

TÀI LIỆU THAM KHẢO

[1] *Three.js – JavaScript 3D Library*. (2025). Threejs.org. https://threejs.org/

[2] GeeksforGeeks. (2024, January 17). *Graph Algorithms*. GeeksforGeeks. https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/

[3] *Johnson, D. B. (1977). Efficient Algorithms for Shortest Paths in Sparse Networks. Journal of the ACM, 24(1), 1–13.https://doi.org/10.1145/321992.321993*

[4] *Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. Numerische Mathematik, 1(1), 269–271. https://doi.org/10.1007/bf01386390*