**BỘ CÔNG THƯƠNG**

**TRƯỜNG ĐẠI HỌC CÔNG THƯƠNG TP HỒ CHÍ MINH**

**KHOA CÔNG NGHỆ THÔNG TIN**



**HỌC PHẦN: TRÍ TUỆ NHÂN TẠO**

**BÁO CÁO CUỐI MÔN**

**ĐỀ TÀI : GIẢI THUẬT GTS**

**NHÓM 11**

**Giảng viên hướng dẫn**: Trần Đình Toàn

**Sinh viên thực hiện:**

Trương Minh Thực – 2001216200

Huỳnh Thanh Nhã - 2001210125

Đỗ Công Tôn Sách – 2001210731

*TP. Hồ Chí Minh, tháng 11 năm 2023*

**A blue and black frame

Description automatically generatedBỘ CÔNG THƯƠNG**

**TRƯỜNG ĐẠI HỌC CÔNG THƯƠNG TP HỒ CHÍ MINH**

**KHOA CÔNG NGHỆ THÔNG TIN**



**HỌC PHẦN: TRÍ TUỆ NHÂN TẠO**

**BÁO CÁO CUỐI MÔN**

**ĐỀ TÀI : GIẢI THUẬT GTS**

**Nhóm: 11**

**Giảng viên hướng dẫn**: Trần Đình Toàn

**Sinh viên thực hiện:**

1. Trương Minh Thực – 2001216200
2. Huỳnh Thanh Nhã - 2001210125
3. Đỗ Công Tôn Sách - 2001210731

*TP. Hồ Chí Minh, tháng 12 năm 2023*

**ĐÁNH GIÁ KẾT QUẢ LÀM VIỆC NHÓM**

|  |  |  |  |
| --- | --- | --- | --- |
| **Mã số sinh viên** | **Họ và tên** | **Nội dung công việc** | **Tiến độ** |
| 2001216200 | Trương Minh Thực | Thiết kế Word, Code | 100% |
| 2001210125 | Đỗ Công Tôn Sách | Thiết kế Powerpoint, Code | 100% |
| 2001210731 | Huỳnh Thanh Nhã | Lên ý tưởng, Demo Code | 100% |

# LỜI CẢM ƠN

Để có thể hoàn thành được bài báo cáo cho đề tài tiểu luận này, nhóm chúng em xin được gửi lời cảm ơn chân thành đến Giảng viên hướng dẫn của nhóm chúng em, thầy Trần Đình Toàn – Giảng viên khoa Công Nghệ Thông Tin, trường Đại học Công Thương Thành phố Hồ Chí Minh vì sự quan tâm, tận tình hướng dẫn giảng dạy, hỗ trợ giúp chúng em có thêm được nhiều kiến thức quý báu để áp dụng và hoàn thành bài tiểu luận này.

Xin cảm ơn tất cả các thành viên trong nhóm đã hợp tác, giúp đỡ lẫn nhau để nhóm có thể hoàn thành tốt đề tài.

Vì nhóm chúng em chưa có nhiều kinh nghiệm nên sẽ không tránh khỏi những điểm hạn chế cũng như những sai sót nhất định. Do đó, chúng em rất mong nhận được sự phản hồi và góp ý của quý thầy cô giáo, quý giảng viên để có thể rút kinh nghiệm cho bản thân.

Chúng em xin chân thành cảm ơn!

*Nhóm sinh viên thực hiện*

*NHÓM 11*

*MỤC LỤC*

[LỜI CẢM ƠN 2](#_Toc153140972)

[I. MỞ ĐẦU 5](#_Toc153140973)

[1. Lý do chọn đề tài 5](#_Toc153140974)

[2. Mục tiêu nghiên cứu 5](#_Toc153140975)

[3. Đối tượng và phạm vi nghiên cứu 5](#_Toc153140976)

[4. Phương pháp nghiên cứu 6](#_Toc153140977)

[5. Cấu trúc đề tài: 6](#_Toc153140978)

[II. GIỚI THIỆU PHƯƠNG PHÁP HEURISTIC 7](#_Toc153140979)

[2.1. Định nghĩa 7](#_Toc153140980)

[2.2. Tìm kiếm theo phương pháp Heuristic là gì? 7](#_Toc153140981)

[2.3. Ứng Dụng của Tìm Kiếm Heuristic trong AI 7](#_Toc153140982)

[2.4. Các Thuật Toán Tìm Kiếm Heuristic Phổ Biến 8](#_Toc153140983)

[2.5. Lợi ích và hạn chế của tìm kiếm Heuristic trong AI 9](#_Toc153140984)

[*2.5.1. Lợi ích: 9*](#_Toc153140985)

[*2.5.2. Hạn chế: 9*](#_Toc153140986)

[2.6. Tương Lai của Tìm Kiếm Heuristic và Trí Tuệ Nhân Tạo 10](#_Toc153140987)

[III. GIẢI THUẬT GTS TRONG BÀI TOÁN TÌM LỘ TRÌNH TỐI ƯU ĐỂ ĐƯA XE KHÁCH THAM QUAN 11](#_Toc153140988)

[3.1. Giới thiệu thuật toán GTS 11](#_Toc153140989)

[3.2. Mô tả thuật toán 11](#_Toc153140990)

[3.3. Ưu nhược điểm của thuật toán 12](#_Toc153140991)

[*3.3.1. Ưu điểm 12*](#_Toc153140992)

[*3.3.2. Nhược điểm 13*](#_Toc153140993)

[3.4. Ứng dụng của thuật toán 13](#_Toc153140994)

[3.5. Giải thích code 14](#_Toc153140995)

[3.6. Kết luận thuật toán 30](#_Toc153140996)

[IV. TÀI LIỆU THAM KHẢO 31](#_Toc153140997)

# MỞ ĐẦU

## Lý do chọn đề tài

Việc tìm kiếm lộ trình tối ưu để đưa khách tham quan các danh thắng lịch sử ở Thành phố Hồ Chí Minh là một vấn đề có độ phức tạp cao, không chỉ do số lượng và sự phân bố của các điểm tham quan mà còn bởi yếu tố giao thông và thời gian di chuyển. Giải thuật GTS, với khả năng kết hợp tìm kiếm cục bộ và toàn cục, là một lựa chọn phù hợp để giải quyết vấn đề này. Giải thuật GTS cho phép tìm kiếm lộ trình tối ưu bằng cách khám phá không gian giải pháp rộng lớn và sau đó tinh chỉnh các giải pháp tiềm năng. Điều này đảm bảo rằng các lộ trình được đề xuất không chỉ ngắn và tiết kiệm thời gian di chuyển, mà còn phù hợp với các yêu cầu cụ thể của tour du lịch, như thời gian mở cửa của các điểm tham quan. Trong ngành du lịch, việc tối ưu hóa chi phí và thời gian cho mỗi tour du lịch là rất quan trọng. GTS giúp tối ưu hóa cả hai yếu tố này bằng cách đề xuất lộ trình hiệu quả, giảm thiểu thời gian di chuyển và chờ đợi, từ đó cung cấp trải nghiệm tốt nhất cho khách du lịch.

## Mục tiêu nghiên cứu

Nghiên cứu về thuật toán GTS (Giải thuật Tìm kiếm Tabu) đặt ra mục tiêu chính là xác định lộ trình di chuyển ngắn nhất và hiệu quả nhất. Mục tiêu này không chỉ giúp cải thiện trải nghiệm của du khách thông qua việc giảm thiểu thời gian di chuyển và chờ đợi không cần thiết mà còn góp phần tiết kiệm chi phí cho các công ty du lịch, qua việc tối ưu hóa sử dụng phương tiện và nhiên liệu. Qua đó, nghiên cứu cũng hướng đến việc nâng cao sự hài lòng của khách hàng, qua việc cung cấp một lịch trình tham quan thoải mái và hợp lý. Một mục tiêu quan trọng khác là phản ánh đặc thù văn hóa và giao thông của Thành phố Hồ Chí Minh trong các lộ trình tham quan, giúp du khách có cái nhìn sâu sắc hơn về thành phố.

## Đối tượng và phạm vi nghiên cứu

* Đối tượng: Giải Thuật GTS
* Phạm vi nghiên cứu: thuật toán GTS trong việc tìm đường đi tối ưu cho khách du lịch tới các danh thắng lịch sử tại Thành phố Hồ Chí Minh.

## Phương pháp nghiên cứu

* Nghiên cứu, phân tích, tham khảo các nguồn tài liệu trên mạng, slide bài giảng môn Trí Tuệ Nhân Tạo trường Đại học Công Thương Thành phố Hồ Chí Minh.

1. **Cấu trúc đề tài:**
2. Mở đầu
3. Giới thiệu phương pháp Heuristic
4. Giải thuật GTS trong bài toán tìm lộ trình tối ưu để đưa xe khách tham quan
5. Tư liệu tham khảo.

# GIỚI THIỆU PHƯƠNG PHÁP HEURISTIC

## Định nghĩa

Heuristic là một phương pháp tìm kiếm hoặc khám phá dựa trên kinh nghiệm và kiến thức có sẵn để đưa ra quyết định nhanh chóng và hiệu quả. Nó không đảm bảo sẽ tìm ra giải pháp tối ưu hay hoàn hảo, nhưng thường mang lại một giải pháp khả thi trong thời gian ngắn.

## Tìm kiếm theo phương pháp Heuristic là gì?

Heuristic là một kỹ thuật để giải quyết một vấn đề nhanh hơn các phương pháp cổ điển hoặc để tìm một giải pháp gần đúng khi các phương pháp cổ điển không làm được. Đây là một loại phím tắt vì chúng ta thường đánh đổi một trong những yếu tố tối ưu, đầy đủ, chính xác hoặc chính xác để lấy tốc độ.

## Ứng Dụng của Tìm Kiếm Heuristic trong AI

Tìm kiếm heuristic (heuristic search) là một phương pháp quan trọng trong Trí tuệ Nhân tạo (Artificial Intelligence – AI), và nó có nhiều ứng dụng thực tế quan trọng. Dưới đây là một số ví dụ về cách tìm kiếm heuristic được áp dụng trong các lĩnh vực khác nhau của cuộc sống:

* **Xác định đường đi tối ưu**: Tìm kiếm heuristic được sử dụng trong các ứng dụng bản đồ trực tuyến như Google Maps để xác định đường đi tối ưu từ một điểm đến một điểm khác dựa trên thông tin về lưu lượng giao thông và khoảng cách. Heuristic giúp ước tính thời gian di chuyển và tối ưu hóa tuyến đường.
* **Trò chơi và trí tuệ nhân tạo**: Trong trò chơi như cờ vua hoặc cờ cờ, tìm kiếm heuristic được sử dụng để đánh giá và chọn bước đi tốt nhất dựa trên các tiêu chí như điểm số, vị trí và chiến thuật.
* **Lập lịch và quản lý tài nguyên**: Trong lĩnh vực quản lý dự án và lập lịch, tìm kiếm heuristic có thể giúp xác định thời gian và tài nguyên cần thiết để hoàn thành một dự án. Nó cũng được sử dụng trong quản lý tồn kho để tối ưu hóa lựa chọn vận chuyển và cung cấp hàng hóa.
* **Xử lý ngôn ngữ tự nhiên**: Trong lĩnh vực xử lý ngôn ngữ tự nhiên, tìm kiếm heuristic được sử dụng để tìm kiếm các từ khóa và cụm từ trong văn bản, giúp đưa ra các kết quả tìm kiếm chính xác và hiệu quả.
* **Chẩn đoán y học**: Trong y học, tìm kiếm heuristic được sử dụng để hỗ trợ việc chẩn đoán bệnh và lập kế hoạch điều trị. Nó có thể giúp đưa ra các ước tính về nguy cơ bệnh tật dựa trên dữ liệu lâm sàng và lịch sử bệnh của bệnh nhân.
* **Tối ưu hóa hệ thống**: Trong các hệ thống tự động như hệ thống giao thông thông minh, tìm kiếm heuristic được sử dụng để tối ưu hóa luồng giao thông và đưa ra các quyết định về đèn giao thông và điều khiển lưu lượng xe cơ động.

## Các Thuật Toán Tìm Kiếm Heuristic Phổ Biến

Có nhiều thuật toán tìm kiếm heuristic phổ biến trong lĩnh vực Trí tuệ Nhân tạo (AI) và tối ưu hóa. Dưới đây là một số trong số những thuật toán này:

* **Thuật toán A (A-Star)\*\*:** A là một trong những thuật toán tìm kiếm heuristic phổ biến nhất. Nó kết hợp giữa tìm kiếm theo chiều rộng (BFS – Breadth-First Search) và tìm kiếm theo chi phí tối ưu nhất. A\* sử dụng một hàm heuristic để ước tính chi phí còn lại đến mục tiêu và tự động điều chỉnh để tối ưu hóa đường đi.
* **Thuật toán Best-First Search**: Best-First Search là một loại tìm kiếm theo chi phí tối ưu nhất, trong đó nó luôn chọn nút con có giá trị heuristic nhỏ nhất. Thuật toán này thường được sử dụng trong các bài toán tìm kiếm không có chi phí đồng nhất đến mục tiêu.
* **Thuật toán Simulated Annealing**: Simulated Annealing là một thuật toán tối ưu hóa được lấy cảm hứng từ quá trình luyện kim. Nó được sử dụng để tìm kiếm không gian tối ưu của các biến số trong các bài toán tối ưu hóa.
* **Thuật toán Hill Climbing**: Hill Climbing là một thuật toán tìm kiếm heuristic đơn giản. Nó bắt đầu từ một giải pháp và liên tục cố gắng cải thiện giải pháp bằng cách di chuyển đến các giải pháp lân cận có giá trị heuristic tốt hơn.
* **Thuật toán Tabu Search**: Tabu Search là một phương pháp tìm kiếm local search được sử dụng trong tối ưu hóa tổ hợp. Nó duyệt qua các giải pháp xung quanh giải pháp hiện tại và sử dụng danh sách “tabu” để tránh quay lại các giải pháp trước đó.

## Lợi ích và hạn chế của tìm kiếm Heuristic trong AI

### Lợi ích:

* Tốc độ: Tìm kiếm heuristic thường nhanh hơn so với tìm kiếm không thông minh hoặc brute-force vì nó tập trung vào các lựa chọn tiềm năng có khả năng cao hơn để giải quyết vấn đề.
* Khả năng tìm giải pháp tốt: Heuristic giúp tìm kiếm tiềm năng tới những giải pháp gần như tối ưu, thậm chí trong các không gian tìm kiếm rộng lớn.
* Áp dụng rộng rãi: Tìm kiếm heuristic có thể áp dụng trong nhiều lĩnh vực, từ tối ưu hóa tổ hợp trong khoa học máy tính đến tìm đường đi trong hệ thống thông tin địa lý.
* Tích hợp với lý thuyết AI khác: Tìm kiếm heuristic có thể được kết hợp với các phương pháp và lý thuyết AI khác để cải thiện hiệu suất và đưa ra các giải pháp thú vị.

### Hạn chế:

* Khả năng chưa tối ưu: Tìm kiếm heuristic không đảm bảo tìm thấy giải pháp tối ưu. Nó chỉ tập trung vào các lựa chọn tiềm năng có vẻ tốt, nhưng có thể bỏ lỡ giải pháp tốt nhất nếu không được định hình đúng cách.
* Phụ thuộc vào hàm heuristic: Hiệu suất của tìm kiếm heuristic phụ thuộc vào chất lượng của hàm heuristic. Một hàm heuristic không tốt có thể dẫn đến kết quả không chính xác hoặc chậm.
* Nguy cơ mắc kẹt tại cực tiểu cục bộ: Tìm kiếm heuristic có thể bị mắc kẹt tại các cực tiểu cục bộ, nghĩa là không thể di chuyển ra khỏi một giải pháp không tối ưu.
* Không thể áp dụng cho mọi loại vấn đề: Tìm kiếm heuristic không phải lúc nào cũng phù hợp cho tất cả loại vấn đề. Có những vấn đề mà tìm kiếm theo chi phí tối ưu hoặc tìm kiếm khác có hiệu suất tốt hơn.
* Cần sự hiểu biết về bài toán: Để áp dụng tìm kiếm heuristic, người phát triển cần có kiến thức về bài toán và khả năng xây dựng hàm heuristic hiệu quả.

## Tương Lai của Tìm Kiếm Heuristic và Trí Tuệ Nhân Tạo

Tương lai của tìm kiếm heuristic và Trí Tuệ Nhân Tạo (AI) hứa hẹn có nhiều triển vọng và tiềm năng lớn. Dưới đây là một số xu hướng và tương lai tiềm năng cho tìm kiếm heuristic và AI:

* **Tích hợp với học máy**: Tìm kiếm heuristic sẽ ngày càng kết hợp mạnh mẽ với các phương pháp học máy và học sâu (deep learning). Khi AI trở nên thông minh hơn trong việc học từ dữ liệu, việc kết hợp tìm kiếm heuristic với khả năng học này sẽ giúp cải thiện khả năng tìm kiếm và đưa ra các giải pháp tối ưu hơn.
* **Tối ưu hóa các ứng dụng thực tế**: Tìm kiếm heuristic sẽ tiếp tục được áp dụng rộng rãi trong các ứng dụng thực tế như tối ưu hóa lộ trình giao thông, tối ưu hóa sản xuất công nghiệp, và quản lý tài nguyên. Các thuật toán tìm kiếm heuristic cơ bản sẽ được phát triển và tùy chỉnh cho từng loại ứng dụng cụ thể.
* **Tìm kiếm trong không gian lớn**: Với sự phát triển của công nghệ và máy tính mạnh mẽ hơn, tìm kiếm heuristic có thể được áp dụng cho các vấn đề với không gian tìm kiếm lớn hơn. Điều này có thể bao gồm tìm kiếm trong dữ liệu lớn, tối ưu hóa trong thời gian thực, và nhiều ứng dụng khác.
* **Sự tương tác với con người**: AI và tìm kiếm heuristic có thể ngày càng phát triển để tương tác tốt hơn với con người. Điều này bao gồm việc tạo ra các giao diện người-máy thông minh hơn và hệ thống AI có khả năng học hỏi từ phản hồi của con người.
* **Độ tin cậy và an toàn**: Các thuật toán tìm kiếm heuristic sẽ được phát triển với khả năng đảm bảo độ tin cậy và an toàn. Điều này quan trọng trong các lĩnh vực như tài chính, y tế, và tự động hóa nơi sự chính xác và độ tin cậy là yếu tố quyết định.
* **Tạo ra giải pháp sáng tạo**: Tìm kiếm heuristic có thể được sử dụng để tạo ra các giải pháp sáng tạo cho các vấn đề phức tạp. Khả năng này có thể được ứng dụng trong nghiên cứu khoa học, thiết kế, và sáng tạo nghệ thuật.

# GIẢI THUẬT GTS TRONG BÀI TOÁN TÌM LỘ TRÌNH TỐI ƯU ĐỂ ĐƯA XE KHÁCH THAM QUAN

## Giới thiệu thuật toán GTS

Thuật toán GTS (Giải thuật Tìm kiếm Tabu) là một phương pháp tối ưu hóa metaheuristic được sử dụng để giải quyết các bài toán tối ưu hóa tổ hợp. Nó được phát triển bởi Fred Glover vào năm 1986 và đã được áp dụng rộng rãi trong nhiều lĩnh vực khác nhau. Cốt lõi của GTS là sử dụng một "danh sách tabu", hoặc danh sách cấm, để tránh việc tìm kiếm lặp lại trong cùng một khu vực của không gian tìm kiếm.

Thuật toán GTS được sử dụng để tìm đường đi ngắn nhất trong một mạng lưới có trọng số. Thuật toán này hoạt động bằng cách bắt đầu tại một đỉnh bất kỳ và liên tục chọn đỉnh có chi phí nhỏ nhất tiếp theo để đến thăm. Quá trình này tiếp tục cho đến khi tất cả các đỉnh trong mạng lưới đều được thăm.

## Mô tả thuật toán

Trong việc tìm lộ trình tối ưu tới các danh thắng lịch sử, việc sử dụng thuật toán GTS không chỉ giúp tối ưu hóa lộ trình dựa trên chiều dài và thời gian di chuyển mà còn có thể tính đến các yếu tố phức tạp khác như thời gian mở cửa của các điểm thăm quan, sở thích của du khách, và các điều kiện đường đi cụ thể:

1. **Khởi tạo:**

* Chọn một lộ trình ban đầu ngẫu nhiên hoặc dựa trên một số chiến lược cụ thể. Lộ trình này bao gồm thứ tự thăm quan các danh thắng lịch sử.
* Tạo danh sách cấm (tabu list) để theo dõi các thay đổi gần đây trong lộ trình, nhằm ngăn chặn việc quay trở lại các lộ trình đã xem xét.

1. **Tìm kiếm tiếp cận:**

* Xác định các lộ trình lân cận bằng cách đổi chỗ thứ tự của các điểm thăm quan hoặc sử dụng các kỹ thuật khác như "cắt và dán" một phần của lộ trình.
* Đánh giá các lộ trình lân cận này dựa trên tổng chiều dài lộ trình, thời gian di chuyển, hoặc các tiêu chí khác.

1. **Cập nhật danh sách cấm:**

* Chọn lộ trình tối ưu nhất từ những lộ trình lân cận không bị cấm bởi danh sách cấm.
* Cập nhật danh sách cấm bằng cách thêm các thay đổi mới vào và loại bỏ các thay đổi cũ khỏi danh sách.

1. **Đánh giá và cập nhật lộ trình:**

* So sánh lộ trình mới tìm được với lộ trình tốt nhất hiện tại. Nếu lộ trình mới tốt hơn, cập nhật lộ trình tốt nhất.
* Trong một số trường hợp, thuật toán có thể chấp nhận lộ trình tệ hơn để tránh mắc kẹt ở cực tiểu địa phương.

1. **Tiêu chí dừng:**

* Quyết định tiếp tục hoặc dừng quá trình tìm kiếm dựa trên các tiêu chí như số lượng vòng lặp tối đa, thời gian tính toán, hoặc khi không còn thấy cải thiện đáng kể trong lộ trình.

1. **Kết quả cuối cùng:**

* Khi đạt đến tiêu chí dừng, thuật toán trả về lộ trình tối ưu nhất mà nó đã tìm được.

## Ưu nhược điểm của thuật toán

### Ưu điểm

* **Tránh mắc kẹt tại cực tiểu địa phương:** Một trong những ưu điểm lớn của GTS là khả năng vượt qua các giải pháp cục bộ và tiếp tục tìm kiếm giải pháp tối ưu toàn cục.
* **Khả năng tìm kiếm rộng rãi:** Nhờ vào cơ chế danh sách cấm, thuật toán khám phá được một phạm vi rộng lớn của không gian giải pháp, giúp tìm ra những giải pháp tốt hơn.
* **Linh hoạt và có thể tuỳ chỉnh:** GTS cho phép các nhà nghiên cứu và kỹ sư tùy chỉnh các quy tắc và chiến lược tìm kiếm dựa trên đặc thù của bài toán cụ thể.
* **Hiệu quả trong các bài toán phức tạp:** Thuật toán này thường hoạt động rất hiệu quả trong việc giải quyết các bài toán tối ưu hóa phức tạp, nơi các thuật toán tìm kiếm cơ bản không hiệu quả.

### Nhược điểm

* **Đòi hỏi tài nguyên tính toán cao:** GTS có thể yêu cầu tài nguyên tính toán đáng kể, đặc biệt là với những bài toán có không gian tìm kiếm lớn.
* **Khó xác định các tham số tối ưu:** Việc chọn lựa kích thước danh sách cấm, tiêu chí dừng, và các tham số khác cần cân nhắc kỹ lưỡng và có thể phức tạp.
* **Không đảm bảo tìm ra giải pháp tối ưu toàn cục:** Mặc dù có khả năng vượt qua các giải pháp cục bộ, GTS không đảm bảo luôn tìm ra giải pháp tối ưu toàn cục.
* **Phụ thuộc vào giải pháp ban đầu và lộ trình tìm kiếm:** Kết quả của thuật toán có thể phụ thuộc nhiều vào giải pháp khởi tạo và lộ trình tìm kiếm được chọn.
* **Cần kiến thực chuyên sâu để hiệu chỉnh:** Để đạt hiệu quả cao, người dùng cần có kiến thức chuyên sâu về bài toán và kỹ năng tối ưu hóa thuật toán.

## Ứng dụng của thuật toán

Thuật toán GTS (Giải thuật Tìm kiếm Tabu) tìm thấy ứng dụng thực tế trong nhiều lĩnh vực khác nhau, nhờ vào khả năng tối ưu hóa mạnh mẽ và linh hoạt của nó.

* **Tối Ưu Hóa Lịch Trình Giao Hàng:** Trong ngành logistics, GTS được sử dụng để tối ưu hóa lịch trình giao hàng, giúp xác định lộ trình tối ưu cho xe tải giao hàng, giảm thời gian di chuyển và chi phí nhiên liệu.
* **Quản Lý Chuỗi Cung Ứng:** GTS tìm kiếm giải pháp tối ưu trong việc phân phối hàng hóa, quản lý kho hàng và dự báo nhu cầu, giúp các doanh nghiệp giảm thiểu chi phí và tăng cường hiệu suất.
* **Tối Ưu Hóa Mạng Lưới Viễn Thông:** Trong ngành viễn thông, GTS được áp dụng để tối ưu hóa thiết kế và quản lý mạng lưới, như việc phân bổ băng thông và tối ưu hóa định tuyến dữ liệu.
* **Quy Hoạch Đô Thị và Giao Thông:** GTS giúp lập kế hoạch và tối ưu hóa hệ thống giao thông đô thị, giảm ùn tắc và cải thiện hiệu quả di chuyển trong thành phố.
* **Tối Ưu Hóa Trong Y Học:** Trong ngành y tế, GTS có thể được sử dụng để lập kế hoạch phân phối nguồn lực y tế, lập lịch cho các phòng mổ, hoặc tối ưu hóa quy trình điều trị cho bệnh nhân.
* **Tối Ưu Hóa Thời Gian và Tài Nguyên Trong Giáo Dục:** GTS giúp các cơ sở giáo dục lập lịch học tập và phân bổ tài nguyên, như phòng học và giáo viên, một cách hiệu quả.
* **Tối Ưu Hóa Trong Khoa Học Dữ Liệu:** Áp dụng GTS trong việc tối ưu hóa các mô hình học máy, giúp chọn lựa các tham số mô hình tối ưu và cải thiện hiệu suất.

## Giải thích code

Ví dụ: Viết chương trình tìm lộ trình tối ưu để xe đưa khách tham quan đến các

danh thắng lịch sử ở thành phố HCM.

1. **INDEX 1:**

function initMap() {

const map = new google.maps.Map(document.getElementById("map"), {

center: { lat: 10.801720556370942, lng: 106.65530426171239 },

zoom: 13,

mapTypeControl: false,

});

const card = document.getElementById("pac-card");

const input = document.getElementById("pac-input");

const input1 = document.getElementById("pac-input1")

const biasInputElement = document.getElementById("use-location-bias");

const strictBoundsInputElement = document.getElementById("use-strict-bounds");

const options = {

fields: ["formatted\_address", "geometry", "name"],

strictBounds: false,

};

* Sử dụng API Google Maps để khởi tạo một bản đồ có tâm ở tọa độ cụ thể.
* Tạo các liên kết giữa các phần tử HTML và biến JavaScript để quản lý tìm kiếm địa chỉ trên bản đồ.
* Tùy chọn tìm kiếm được đặt để trả về thông tin như địa chỉ định dạng, hình học và tên địa điểm.
* Mã này có thể là một phần của ứng dụng web tích hợp Google Maps để hiển thị bản đồ và cho phép người dùng tìm kiếm địa chỉ.

card.style.position = "absolute";

card.style.left = "10px";

card.style.top = "10px";

* Thêm một phần tử HTML (được đại diện bởi biến **card**) vào cuối phần tử **<body>** của trang web.

const autocomplete = new google.maps.places.Autocomplete(input, options);

const autocomplete1 = new google.maps.places.Autocomplete(input1, options);

* Tạo hai đối tượng Autocomplete từ Google Places API để kích hoạt tính năng tự động hoàn thành cho hai ô tìm kiếm địa chỉ trên trang web.
* Đối tượng Autocomplete giúp người dùng dễ dàng và nhanh chóng nhập địa chỉ bằng cách đề xuất và hiển thị các kết quả tìm kiếm phù hợp khi họ nhập liệu vào ô tìm kiếm.

autocomplete.bindTo("bounds", map);

autocomplete1.bindTo("bounds", map);

const infowindow = new google.maps.InfoWindow();

const infowindowContent = document.getElementById("infowindow-content");

infowindow.setContent(infowindowContent);

* Thiết lập ràng buộc giữa ô tìm kiếm địa chỉ và ranh giới của bản đồ, giới hạn kết quả tìm kiếm trong phạm vi bản đồ.
* Khởi tạo một đối tượng InfoWindow để hiển thị thông tin chi tiết về địa chỉ khi người dùng chọn một kết quả từ ô tìm kiếm. Phần tử HTML "infowindow-content" được liên kết với InfoWindow để chứa thông tin chi tiết.

const marker = new google.maps.Marker({

map,

anchorPoint: new google.maps.Point(0, -29),

});

* Khởi tạo một đối tượng **Marker** của Google Maps và liên kết nó với bản đồ đã được tạo trước đó.

autocomplete.addListener("place\_changed", () => {

infowindow.close();

marker.setVisible(false);

const place = autocomplete.getPlace();

if (!place.geometry || !place.geometry.location) {

window.alert("No details available for input: '" + place.name + "'");

return;

}

if (place.geometry.viewport) {

map.fitBounds(place.geometry.viewport);

} else {

map.setCenter(place.geometry.location);

map.setZoom(17);

}

marker.setPosition(place.geometry.location);

marker.setVisible(true);

infowindowContent.children["place-name"].textContent = place.name;

infowindowContent.children["place-address"].textContent =

place.formatted\_address;

infowindow.open(map, marker);

});

* Đóng cửa sổ thông tin hiện tại và ẩn đối tượng đánh dấu trên bản đồ.
* Lấy thông tin chi tiết về địa điểm từ ô tìm kiếm.
* Kiểm tra và hiển thị cảnh báo nếu không có thông tin chi tiết về địa điểm.
* Điều chỉnh ranh giới bản đồ để chứa địa điểm được chọn hoặc đặt trung tâm và thu phóng nếu không có ranh giới.
* Đặt vị trí của đối tượng đánh dấu theo địa điểm được chọn, hiển thị đối tượng đánh dấu và cập nhật thông tin chi tiết trong cửa sổ thông tin. Cuối cùng, mở cửa sổ thông tin trên bản đồ.

Top of Form

function setupClickListener(id, types) {

const radioButton = document.getElementById(id);

radioButton.addEventListener("click", () => {

autocomplete.setTypes(types);

input.value = "";

});

}

* Lấy tham chiếu đến nút radio dựa trên ID được chỉ định.
* Thêm lắng nghe sự kiện "click" cho nút radio.
* Khi người dùng click vào nút radio, hàm thực hiện các hành động sau:
  + Thiết lập loại địa điểm (place types) cho ô tìm kiếm (**autocomplete**) bằng loại được chỉ định.
  + Xóa nội dung hiện tại của ô tìm kiếm địa chỉ (**input**), chuẩn bị cho một tìm kiếm mới với loại địa điểm mới được chọn.

setupClickListener("changetype-all", []);

setupClickListener("changetype-address", ["address"]);

setupClickListener("changetype-establishment", ["establishment"]);

setupClickListener("changetype-geocode", ["geocode"]);

setupClickListener("changetype-cities", ["(cities)"]);

setupClickListener("changetype-regions", ["(regions)"]);

biasInputElement.addEventListener("change", () => {

if (biasInputElement.checked) {

autocomplete.bindTo("bounds", map);

} else {

autocomplete.unbind("bounds");

autocomplete.setBounds({ east: 180, west: -180, north: 90, south: -90 });

strictBoundsInputElement.checked = biasInputElement.checked;

}

input.value = "";

});

* **Thiết lập Loại Địa Điểm:**
* Mỗi nút radio được liên kết với một loại địa điểm cụ thể (address, establishment, geocode, cities, regions).
* Khi người dùng chọn một nút radio, loại địa điểm của ô tìm kiếm được cập nhật tương ứng và nội dung ô tìm kiếm được xóa.
* **Thiết Lập Sự Kiện Cho Ô Checkbox Vị Trí Ưu Tiên:**
* Khi ô checkbox vị trí ưu tiên thay đổi trạng thái, mã xác định liệu nó có được chọn hay không.
* Nếu được chọn, ô tìm kiếm được ràng buộc với ranh giới của bản đồ.
* Nếu không được chọn, ràng buộc với ranh giới bản đồ sẽ được hủy bỏ và toàn bộ thế giới sẽ trở thành ranh giới. Đồng thời, chế độ ranh giới nghiêm ngặt cũng sẽ được vô hiệu hóa.
* **input.value = "";:**
* Xóa nội dung hiện tại của ô tìm kiếm địa chỉ để chuẩn bị cho tìm kiếm mới.

strictBoundsInputElement.addEventListener("change", () => {

autocomplete.setOptions({

strictBounds: strictBoundsInputElement.checked,

});

if (strictBoundsInputElement.checked) {

biasInputElement.checked = strictBoundsInputElement.checked;

autocomplete.bindTo("bounds", map);

}

input.value = "";

});

* Cập nhật tùy chọn của ô tìm kiếm (**autocomplete**) để sử dụng ranh giới nghiêm ngặt theo trạng thái mới của ô checkbox.
* Nếu ô checkbox được chọn:
* Đặt trạng thái của ô checkbox vị trí ưu tiên (**biasInputElement**) bằng với ô checkbox ranh giới nghiêm ngặt.
* Ràng buộc ô tìm kiếm với ranh giới của bản đồ.
* Nếu ô checkbox không được chọn, không có ràng buộc và trạng thái của ô checkbox vị trí ưu tiên được giữ nguyên.
* Xóa nội dung hiện tại của ô tìm kiếm địa chỉ (**input**).

1. **INDEX2:**

var map;

var directionsService;

var directionsRenderer;

var array = [];

const fileInput = document.querySelector('input[name="file"]');

* Khai báo các biến để lưu trữ đối tượng bản đồ, dịch vụ chỉ đường, và bộ vẽ chỉ đường của Google Maps.
* Khai báo một mảng (**array**) và lấy tham chiếu đến một phần tử HTML input có tên "file".

fileInput.addEventListener('change', () => {

const file = fileInput.files[0];

const reader = new FileReader();

reader.readAsText(file);

reader.onload = () => {

const data = reader.result;

array = data.split('|');

directions();

};

});

* **const file = fileInput.files[0];:**
* Lấy tệp được chọn thông qua thuộc tính **files** của phần tử input.
* **const reader = new FileReader();:**
  + Tạo một đối tượng **FileReader** để đọc nội dung của tệp.
* **reader.readAsText(file);:**
* Đọc nội dung của tệp dưới dạng văn bản.
* **reader.onload = () => { ... };:**
  + Thiết lập sự kiện được gọi khi quá trình đọc tệp hoàn thành.
* **const data = reader.result;:**
  + Lấy kết quả đọc được từ tệp, chứa trong thuộc tính **result** của đối tượng **FileReader**.
* **array = data.split('|');:**
  + Chia dữ liệu thành các phần bằng dấu "|" và gán cho mảng **array**.
* **directions();:**
  + Gọi hàm **directions()**. Tuy nhiên, mã không cung cấp chi tiết về nội dung hoặc chức năng của hàm **directions()**.

function directions() {

const map = new google.maps.Map(document.getElementById("map"), {

zoom: 7,

center: { lat: 10.801720556370942, lng: 106.65530426171239 },

});

directionsService = new google.maps.DirectionsService();

directionsRenderer = new google.maps.DirectionsRenderer({

draggable: true,

map: map,

panel: document.getElementById("panel1"),

});

directionsRenderer.addListener("directions\_changed", function () {

computeTotalDistance(directionsRenderer.getDirections());

});

const waypoints = getOptimizedWaypoints();

displayRoute(waypoints);

}

* + **Tạo Bản Đồ:**
  + Tạo một đối tượng bản đồ mới của Google Maps với các thuộc tính như độ phóng và tâm được thiết lập.
  + **Khởi Tạo Dịch Vụ và Bộ Vẽ Chỉ Đường:**
  + Tạo một đối tượng dịch vụ chỉ đường (**directionsService**) của Google Maps.
  + Tạo một đối tượng bộ vẽ chỉ đường (**directionsRenderer**) với các thuộc tính như có thể kéo thả, được gắn liền với bản đồ, và có một panel để hiển thị thông tin chỉ đường.
  + **Lắng Nghe Sự Kiện "directions\_changed":**
  + Thêm lắng nghe sự kiện "directions\_changed" cho bộ vẽ chỉ đường.
  + Khi hành trình chỉ đường thay đổi (chẳng hạn khi người dùng di chuyển các điểm trên bản đồ), hàm **computeTotalDistance()** được gọi để tính toán tổng khoảng cách của hành trình mới.
  + **Lấy Các Điểm Đi qua:**
  + Gọi hàm **getOptimizedWaypoints()** để lấy danh sách các điểm đi qua được tối ưu hóa.
  + **Hiển Thị Hành Trình Chỉ Đường:**
  + Gọi hàm **displayRoute(waypoints)** để hiển thị hành trình chỉ đường trên bản đồ, sử dụng các điểm đi qua đã lấy được.

function applyGTS(waypoints) {

const optimizedWaypoints = [waypoints[0]];

let cost = 0;

let v = waypoints[0];

for (let k = 1; k < waypoints.length; k++) {

const unusedWaypoints = waypoints.filter((point) => {

return !optimizedWaypoints.includes(point);

});

let w;

for (let i = 0; i < unusedWaypoints.length; i++) {

minCost = getCost(v, unusedWaypoints[i]).then((currentCost)=>{

cost += currentCost

});

w = unusedWaypoints[i];

}

optimizedWaypoints.push(w);

v = w;

}

return optimizedWaypoints;

}

* **Khởi Tạo Mảng optimizedWaypoints:**
  + Tạo một mảng **optimizedWaypoints** với một phần tử đầu tiên là điểm xuất phát (waypoints[0]).
* **Thực Hiện Vòng Lặp Cho Các Điểm Còn Lại:**
  + Sử dụng vòng lặp để xử lý từng điểm còn lại trong danh sách waypoints.
* **Lọc Các Điểm Chưa Được Sử Dụng:**
  + Sử dụng **filter** để lọc ra các điểm chưa được sử dụng, nằm trong danh sách **unusedWaypoints**.
* **Chọn Điểm W với Chi Phí Nhỏ Nhất:**
  + Sử dụng vòng lặp để lặp qua các điểm chưa được sử dụng và tính toán chi phí từ điểm hiện tại (V) đến từng điểm chưa được sử dụng (W).
  + Chọn điểm W sao cho chi phí là nhỏ nhất.
* **Thêm Cung (V, W) vào Hành Trình và Cập Nhật Chi Phí:**
  + Thêm điểm W vào mảng **optimizedWaypoints** để tạo cung (V, W) trong hành trình.
  + Cập nhật chi phí (cost) với chi phí từ V đến W.
* **Chuyển Điểm W thành Đỉnh Hiện Tại (V) Cho Bước Tiếp Theo:**
  + Đặt W là đỉnh hiện tại để xét bước kế tiếp.
* **Trả Về Mảng Tối Ưu Hóa:**
  + Kết thúc hàm và trả về mảng **optimizedWaypoints** chứa hành trình được tối ưu hóa.

function displayRoute(waypoints) {

const origin = waypoints[0];

const destination = waypoints[waypoints.length - 1];

const intermediateWaypoints = waypoints.slice(1, -1);

const request = {

origin: origin,

destination: destination,

waypoints: intermediateWaypoints.map((waypoint) => ({ location: waypoint })),

travelMode: google.maps.TravelMode.DRIVING,

avoidTolls: true,

};

directionsService.route(request, function (result, status) {

if (status === google.maps.DirectionsStatus.OK) {

directionsRenderer.setDirections(result);

} else {

alert("Yêu cầu hướng dẫn đường thất bại do " + status);

}

});

}

* **Xác Định Điểm Xuất Phát, Điểm Đến, và Các Điểm Trung Gian:**
  + Lấy điểm xuất phát (origin) là điểm đầu tiên trong danh sách **waypoints**.
  + Lấy điểm đến (destination) là điểm cuối cùng trong danh sách **waypoints**.
  + Lấy danh sách các điểm trung gian (intermediateWaypoints) bằng cách cắt bỏ điểm xuất phát và điểm đến từ danh sách **waypoints**.
* **Tạo Yêu Cầu Chỉ Đường:**
  + Tạo một đối tượng yêu cầu chỉ đường (**request**) với các thông tin như điểm xuất phát, điểm đến, danh sách các điểm trung gian, chế độ di chuyển (DRIVING), và tùy chọn tránh phí (avoidTolls).
* **Gửi Yêu Cầu Chỉ Đường:**
  + Gọi phương thức **directionsService.route(request, function (result, status) {...});** để gửi yêu cầu chỉ đường đến API của Google Maps.
* **Xử Lý Kết Quả:**
  + Trong hàm callback, kiểm tra xem kết quả có thành công hay không (status === google.maps.DirectionsStatus.OK).
  + Nếu thành công, sử dụng **directionsRenderer.setDirections(result);** để hiển thị hành trình trên bản đồ.
  + Nếu không thành công, hiển thị thông báo lỗi.

function computeTotalDistance(result) {

let total = 0;

const myroute = result.routes[0];

if (!myroute) {

return;

}

for (let i = 0; i < myroute.legs.length; i++) {

total += myroute.legs[i].distance.value;

}

total = total / 1000;

var id = document.getElementById('tong');

id.innerHTML = "Tổng quãng đường: " + total + " km";

}

* + **Khởi Tạo Biến total:**
  + Khởi tạo biến **total** để lưu trữ tổng quãng đường, bắt đầu từ giá trị 0.
* **Lấy Thông Tin Hành Trình Từ Kết Quả:**
  + Lấy thông tin về hành trình từ kết quả của yêu cầu chỉ đường.
* **Kiểm Tra Tồn Tại Hành Trình:**
  + Kiểm tra xem hành trình có tồn tại hay không (**!myroute**).
  + Nếu hành trình không tồn tại, hàm kết thúc mà không thực hiện bất kỳ tính toán nào.
* **Lặp Qua Các Đoạn (Legs) Của Hành Trình và Tính Tổng Quãng Đường:**
  + Sử dụng vòng lặp để lặp qua từng đoạn (leg) của hành trình và cộng dồn giá trị quãng đường vào biến **total**.
* **Chuyển Đổi Tổng Quãng Đường Sang Đơn Vị Km:**
  + Chia giá trị tổng quãng đường cho 1000 để chuyển đổi sang đơn vị kilômét.
* **Hiển Thị Tổng Quãng Đường Trên Giao Diện:**
  + Lấy tham chiếu đến phần tử có id là 'tong' trong HTML (**var id = document.getElementById('tong');**).
  + Gán nội dung của phần tử đó bằng thông tin về tổng quãng đường đã tính toán.

function getOptimizedWaypoints() {

var value = document.getElementById('pac-input').value

var value1 = document.getElementById('pac-input1').value

if (value != "" && value1 != "" && value !== value1) {

if (!array.includes(value) && !array.includes(value1)) {

array.push(value)

array.push(value1)

}

}

if (array.length === 0) {

alert("dữ liệu chua nhập đủ")

window.onload()

}

return array;

}

* **Lấy Giá Trị từ Hai Ô Tìm Kiếm:**
  + Lấy giá trị từ hai ô tìm kiếm có id là 'pac-input' và 'pac-input1'.
* **Kiểm Tra và Thêm Giá Trị vào Mảng array:**
  + Kiểm tra xem cả hai giá trị từ ô tìm kiếm có đủ dữ liệu và khác nhau hay không.
  + Nếu đúng, kiểm tra xem giá trị từ cả hai ô đã tồn tại trong mảng **array** hay chưa.
  + Nếu chưa tồn tại, thêm cả hai giá trị vào mảng **array**.
* **Kiểm Tra Độ Dài của Mảng array:**
  + Kiểm tra xem mảng **array** có chứa ít nhất một phần tử hay không.
  + Nếu mảng không chứa phần tử nào, hiển thị cảnh báo và gọi lại hàm **window.onload()** để tải lại trang.
* **Trả về Mảng array:**
  + Trả về mảng **array** sau khi đã kiểm tra và thêm giá trị.

function getOptimizedWaypoints1() {

// ... code hiện tại ...

var value = document.getElementById('pac-input').value

var value1 = document.getElementById('pac-input1').value

if (value != "" && value1 != "" && value !== value1) {

if (!waypoints.includes(value) && !waypoints.includes(value1)) {

waypoints.push(value)

waypoints.push(value1)

}

}

const optimizedWaypoints = applyGTS(waypoints);

return optimizedWaypoints;

}

* **Lấy Giá Trị từ Hai Ô Tìm Kiếm:**
  + Lấy giá trị từ hai ô tìm kiếm có id là 'pac-input' và 'pac-input1'.
* **Kiểm Tra và Thêm Giá Trị vào Mảng waypoints:**
  + Kiểm tra xem cả hai giá trị từ ô tìm kiếm có đủ dữ liệu và khác nhau hay không.
  + Nếu đúng, kiểm tra xem giá trị từ cả hai ô đã tồn tại trong mảng **waypoints** hay chưa.
  + Nếu chưa tồn tại, thêm cả hai giá trị vào mảng **waypoints**.
* **Sử Dụng Thuật Toán GTS để Tối Ưu Hóa Mảng waypoints:**
  + Gọi hàm **applyGTS(waypoints)** để tối ưu hóa thứ tự của các điểm trung gian trong mảng **waypoints**.
* **Trả về Mảng optimizedWaypoints:**
  + Trả về mảng **optimizedWaypoints** sau khi đã áp dụng thuật toán GTS.

function getCost(origin, destination) {

const directionsService = new google.maps.DirectionsService();

// Tạo một yêu cầu hướng dẫn từ origin đến destination

const request = {

origin: origin,

destination: destination,

travelMode: google.maps.TravelMode.DRIVING,

avoidTolls: true,

};

// Gửi yêu cầu hướng dẫn và tính toán chi phí

return new Promise((resolve, reject) => {

directionsService.route(request, function (response, status) {

if (status === google.maps.DirectionsStatus.OK) {

const route = response.routes[0];

let bestCost = Infinity;

// Tìm chi phí tốt nhất từ các tuyến đường có thể

for (let i = 0; i < route.legs.length; i++) {

const leg = route.legs[i];

let totalDistance = leg.distance.value;

const costPerMeter = 0.1; // Giả sử chi phí là 0.1 đơn vị tiền tệ cho mỗi mét

const cost = totalDistance \* costPerMeter;

if (cost < bestCost) {

bestCost = cost;

}

}

resolve(bestCost);

} else {

reject("Không tìm thấy đường đi. Lỗi: " + status);

}

});

});

}

* Hàm **getCost(origin, destination)** nhận vào hai điểm là điểm xuất phát và điểm đến, sau đó sử dụng dịch vụ chỉ đường của Google Maps để tính toán và trả về chi phí tối ưu cho hành trình giữa hai điểm đó.
* Hàm trả về một Promise, giải quyết với giá trị chi phí tối ưu nếu tính toán thành công, hoặc bị từ chối với thông báo lỗi nếu có vấn đề trong quá trình tính toán.
* Chi phí được tính dựa trên các chi phí giả định cho mỗi mét của hành trình, và hàm sử dụng thuật toán GTS để tối ưu hóa đường đi.

function GTS() {

const optimizedWaypoints = getOptimizedWaypoints1();

const directionsService = new google.maps.DirectionsService();

const directionsRenderer = new google.maps.DirectionsRenderer();

const mapOptions = {

center: { lat: 10.801720556370942, lng: 106.65530426171239 },

zoom: 12,

};

const map = new google.maps.Map(document.getElementById("map"), mapOptions);

directionsRenderer.setMap(map);

// Create a directions request

const request = {

origin: optimizedWaypoints[0],

destination: optimizedWaypoints[optimizedWaypoints.length - 1],

waypoints: optimizedWaypoints.slice(1, -1).map((waypoint) => ({ location: waypoint })),

travelMode: google.maps.TravelMode.DRIVING,

avoidTolls: true,

};

// Send the directions request and display the result on the map

directionsService.route(request, function (response, status) {

if (status === google.maps.DirectionsStatus.OK) {

directionsRenderer.setDirections(response);

} else {

console.log("Failed to find the route. Error: " + status);

}

});

}

* **Lấy Các Điểm Đi Qua Tối Ưu Hóa:**
  + Gọi hàm **getOptimizedWaypoints1()** để lấy danh sách các điểm đi qua được tối ưu hóa.
* **Tạo Bản Đồ và Đối Tượng Renderer:**
  + Tạo một bản đồ mới của Google Maps với tùy chọn zoom và tâm được thiết lập.
  + Tạo một đối tượng **DirectionsRenderer** để hiển thị kết quả chỉ đường trên bản đồ.
* **Đặt Bản Đồ và Renderer:**
  + Gọi **directionsRenderer.setMap(map);** để đặt bản đồ cho đối tượng **DirectionsRenderer** để hiển thị kết quả trên bản đồ.
* **Tạo Yêu Cầu Chỉ Đường:**
  + Tạo một yêu cầu chỉ đường (**request**) với thông tin về điểm xuất phát, điểm đến, và danh sách các điểm trung gian từ danh sách các điểm đi qua đã được tối ưu hóa.
* **Gửi Yêu Cầu và Hiển Thị Kết Quả:**
  + Sử dụng **directionsService.route(request, function (response, status) {...});** để gửi yêu cầu chỉ đường đến API của Google Maps và xử lý kết quả trong một hàm callback.
  + Nếu kết quả thành công (status === google.maps.DirectionsStatus.OK), sử dụng **directionsRenderer.setDirections(response);** để hiển thị kết quả chỉ đường trên bản đồ.
  + Nếu không thành công, ghi log thông báo lỗi vào console.

## Kết luận thuật toán

Giải thuật Greedy Traveling Salesman (GTS) là một phương pháp heuristic được thiết kế để tìm đường đi ngắn nhất trong đồ thị có trọng số. Thuật toán này hoạt động bằng cách liên tục chọn cạnh có trọng số nhỏ nhất tại mỗi bước. Mặc dù GTS có những ưu điểm như độ phức tạp thời gian thấp (O(n^2)) và tính dễ hiểu, nhưng cũng đối mặt với nhược điểm quan trọng là không đảm bảo tìm ra đường đi ngắn nhất trong đồ thị.

Để cải thiện khả năng tìm kiếm của GTS, có thể áp dụng các cải tiến như sử dụng heuristic để đánh giá cạnh có trọng số nhỏ nhất và kỹ thuật backtracing để khám phá các đường đi khác. Mặc dù những cải tiến này có thể giúp GTS tìm ra lời giải tốt hơn trong nhiều trường hợp, nhưng vẫn cần lưu ý rằng không có giải thuật nào hoàn hảo cho mọi tình huống.

Kết luận chung, GTS là một công cụ heuristic linh hoạt và hiệu quả, đặc biệt là đối với các đồ thị có kích thước nhỏ đến vừa. Tuy nhiên, khi cần độ chính xác cao hơn và tìm kiếm trên đồ thị lớn, việc xem xét các thuật toán tìm kiếm toàn cục hoặc sử dụng các cải tiến như heuristic và backtracing là cần thiết để đảm bảo hiệu suất tối ưu.

1. Top of Form
2. Bottom of Form

# TÀI LIỆU THAM KHẢO

1. AI. *ChatGPT*. Được truy lục từ Giải thuật GTS: https://chat.openai.com
2. Zdocs. *GTS.* Được truy lục từ https://zdocs.tips/doc/bai-tapsadas-d6wn0557jg68
3. *tailieu.vn*. Được truy lục từ nguyenlyheuristic
4. *Tài liệu đại học*. Được truy lục từ https://www.tailieudaihoc.com/3doc/1434849.html
5. *123.docz.net*. Được truy lục từ https://123docz.net/timkiem/t%C3%ACm+hi%E1%BB%83u+c%C3%A0i+%C4%91%E1%BA%B7t+thu%E1%BA%ADt+to%C3%A1n+gts1+v%C3%A0+gts2.htm
6. *tritueviet.vn*. Được truy lục từ AI\_Algorthim\_GTS: https://trituevietvn.com/chi-tiet/-thuat-giai-gts-greedy-traveling-saleman--10154