

TRƯỜNG ĐH SƯ PHẠM KỸ THUẬT TP.HỒ CHÍ MINH

KHOA CÔNG NGHỆ THÔNG TIN



TIÊU LUẬN CUỐI KÌ

Môn học: Trí tuệ nhân tạo

Tên tiêu luận:

TÌM ĐƯỜNG ĐI TRÊN BẢN ĐỒ GIAO THÔNG

Giảng viên: PGS.TS. Hoàng Văn Dũng

Danh sách sinh viên thực hiện

Mã số SV	Họ và tên	Mức độ đóng góp (%)
20110120	Huỳnh Thanh Tuấn	100%
20110121	Nguyễn Thành Đạt	100%
20110560	Phan Hồng Sơn	100%

TP. Hồ Chí Minh, tháng 5 năm 2022

MỤC MỤC

DANH SÁCH CÁC CỤM TỪ/ TỪ VIẾT TẮT	1
DANH SÁCH HÌNH VẼ	2
PHẦN 1: MỞ ĐẦU	3
1.1 Phát biểu bài toán	3
1.2 Mục đích.....	3
1.3 Phạm vi và đối tượng.....	3
PHẦN 2: CƠ SỞ LÝ THUYẾT THỰC HIỆN BÀI TOÁN	4
2.1 Công cụ và môi trường lập trình	4
2.2 Thư viện và ngôn ngữ lập trình.	4
2.3 Phương pháp và kĩ thuật sử dụng.	4
PHẦN 3: PHÂN TÍCH VÀ THIẾT KẾ GIẢI PHÁP.....	5
3.1 Sơ đồ khái và ý tưởng thuật toán.....	5
3.1.1 Sơ đồ khái.....	5
3.1.2 Thuật toán A*	5
3.1.3 Thuật toán UCS	5
3.2 Chi tiết về các thuật toán chính.	6
3.2.1. Thuật toán A*	6
3.2.1 Thuật toán UCS(<i>Uniform Cost Search</i>)	10
PHẦN 4: THỰC NGHIỆM, ĐÁNH GIÁ, PHÂN TÍCH KẾT QUẢ	15
4.1 Mô tả dữ liệu và các thuật toán cài đặt	15
4.1.1 Mô tả dữ liệu	15
4.1.2 Cài đặt thuật toán	15
4.2 Trình bày các kết quả thử nghiệm	18
4.2.1. Map A	18
4.2.2. Map B	24
4.3 Giải thích, hướng dẫn, thực thi phần mềm.....	30
PHẦN 5: KẾT LUẬN.....	35
5.1 Đánh giá những kết quả đã thực hiện được	35
5.2 Định hướng phát triển	35
TÀI LIỆU THAM KHẢO:	36

DANH SÁCH CÁC CỤM TỪ/ TỪ VIẾT TẮT

1. Uniform Cost Search - UCS
2. A Star - A*

DANH SÁCH HÌNH VẼ

Hình 1: Đồ thị minh họa thuật toán A*	7
Hình 2: Giá trị heuristic	8
Hình 3: Bảng quá trình duyệt đồ thị với thuật toán A*	8
Hình 4: Kết quả cây tìm kiếm của thuật toán A*	9
Hình 5: Đồ thị minh họa thuật toán UCS	12
Hình 6: Bảng quá trình duyệt đồ thị với thuật toán UCS	12
Hình 7: Kết quả cây tìm kiếm của thuật toán UCS	13
Hình 8: Lần 1 thuật toán A Star	19
Hình 9: Lần 1 thuật toán UCS	19
Hình 10: Lần 2 thuật toán A Star	20
Hình 11: Lần 2 thuật toán UCS	20
Hình 12: Lần 3 thuật toán A Star	21
Hình 13: Lần 3 thuật toán UCS	21
Hình 14: Lần 4 thuật toán A Star	22
Hình 15: Lần 4 thuật toán UCS	22
Hình 16: Lần 5 thuật toán A Star	23
Hình 17: Lần 5 thuật toán UCS	23
Hình 18: Bảng tổng hợp kết quả thử nghiệm MAP A	24
Hình 19. Lần 1 thuật toán A Star	24
Hình 20. Lần 1 thuật toán UCS	25
Hình 21. Lần 2 thuật toán A Star	25
Hình 22. Lần 2 thuật toán UCS	26
Hình 23. Lần 3 thuật toán A Star	26
Hình 24. Lần 3 thuật toán UCS	27
Hình 25. Lần 4 thuật toán A Star	27
Hình 26. Lần 4 thuật toán UCS	28
Hình 27: Lần 5 thuật toán A Star	28
Hình 28. Lần 5 thuật toán UCS	29
Hình 29. Bảng tổng hợp kết quả thử nghiệm MAP B	29
Hình 30: Điểm xuất phát	30
Hình 31: Điểm đến	30
Hình 32: Các đỉnh duyệt qua	31
Hình 33: Kết quả tìm đường đi	31
Hình 34: Giao diện khởi động	32
Hình 35: Kết quả chọn Map A	32
Hình 36: Chọn điểm đầu	33
Hình 37: Hộp thoại thông báo	33
Hình 38: Kết quả chạy	34

PHẦN 1: MỞ ĐẦU

1.1 Phát biểu bài toán.

Nhằm tổng kết và vận dụng các kiến thức mà chúng em đã được học trong môn Trí Tuệ Nhân Tạo của thầy Dũng, chúng em đã xây dựng một chương trình tìm đường đi trên bản đồ giao thông. Chương trình gồm chức năng tìm đường đi dựa trên chiến lược tìm kiếm mù với thuật toán UCS và chiến lược tìm kiếm kinh nghiệm với thuật toán A*.

1.2 Mục đích.

Để có được cái nhìn bao quát về các thuật toán được học và so sánh sự khác nhau giữa chúng. Từ đó, tìm ra được ưu điểm và nhược điểm của các thuật toán trong việc tìm đường đi trong bản đồ giao thông.

1.3 Phạm vi và đối tượng.

Đối tượng: Hoạt động của thuật toán A * và UCS.

Phạm vi: Tìm đường trong bản đồ giao thông.

PHẦN 2: CƠ SỞ LÝ THUYẾT THỰC HIỆN BÀI TOÁN

2.1 Công cụ và môi trường lập trình

Để xây dựng chương trình tìm đường đi trong bản đồ giao thông nhóm em đã sử dụng:



Visual studio code



Spyder

2.2 Thư viện và ngôn ngữ lập trình.

Thư viện **tkinter**: dùng để tạo giao diện đồ họa.

Thư viện **time**: dùng để lấy thời gian.

Thư viện **queue**: dùng để tạo hàng đợi ưu tiên.

Thư viện **collections**: dùng để lưu data theo hàm defaultdict().

Thư viện **math**: dùng để tính toán.

```
from collections import defaultdict
from queue import PriorityQueue
import math
import time
from tkinter import *
```

Ngôn ngữ lập trình sử dụng là:



Python

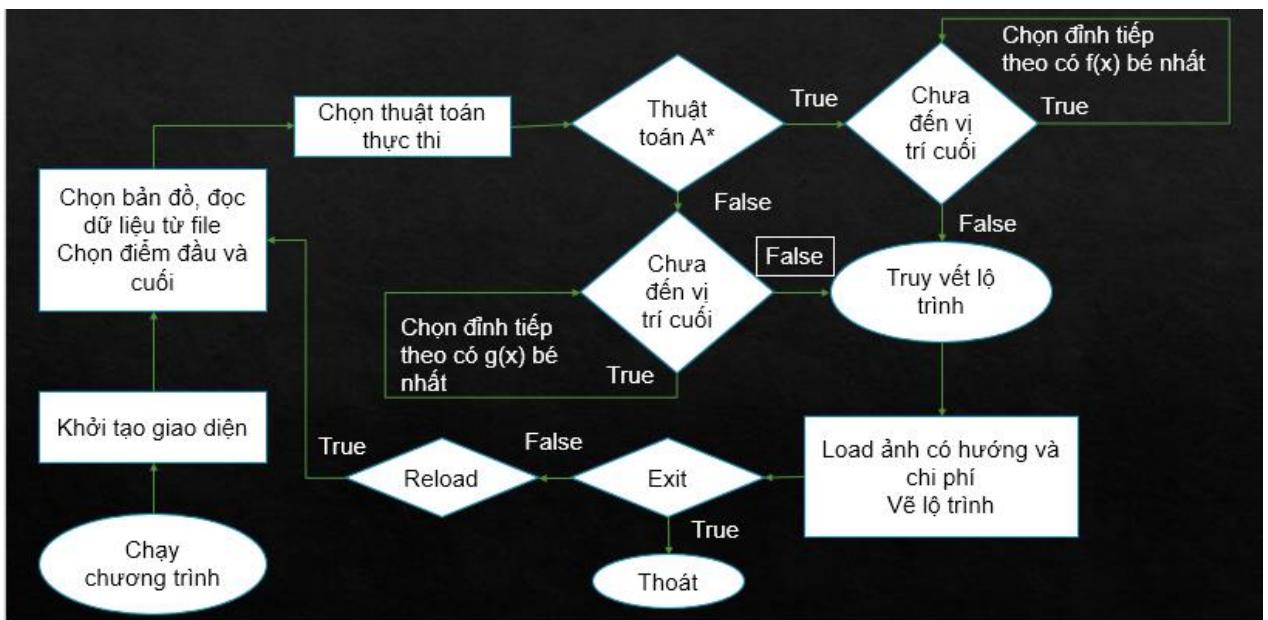
2.3 Phương pháp và kỹ thuật sử dụng.

Phương pháp lập trình: hướng đối tượng để tạo ra các node, tạo ra giao diện đồ họa. Kỹ thuật lập trình sử dụng: đệ quy dùng để truy vết lô trình.

PHẦN 3: PHÂN TÍCH VÀ THIẾT KẾ GIẢI PHÁP

3.1 Sơ đồ khái và ý tưởng thuật toán.

3.1.1 Sơ đồ khái



3.1.2 Thuật toán A*

Xét bài toán tìm đường - bài toán mà A* thường được dùng để giải. A* xây dựng tăng dần tất cả các tuyến đường từ điểm xuất phát cho tới khi nó tìm thấy một đường đi chạm tới đích. Tuy nhiên, cũng như tất cả các thuật toán tìm kiếm có thông tin, nó chỉ xây dựng các tuyến đường "có vẻ" dẫn về phía đích.

Để biết những tuyến đường nào có khả năng sẽ dẫn tới đích, A* sử dụng một "đánh giá heuristic" về khoảng cách từ điểm bất kỳ cho trước tới đích. Trong trường hợp tìm đường đi, đánh giá này có thể là khoảng cách đường chim bay - một đánh giá xấp xỉ thường dùng cho khoảng cách của đường giao thông.

3.1.3 Thuật toán UCS

Thay vì chèn tất cả các đỉnh của đồ thị vào hàng đợi ưu tiên như thuật toán Dijkstra. Thì thuật toán UCS là một biến thể của thuật toán Dijkstra, nó chỉ chèn trạng thái đầu vào hàng đợi ưu tiên Open và mọi nút mới sẽ được chèn vào Open theo mức độ ưu tiên chi phí thấp nhất. Nút ở đầu Open sẽ được lấy ra vào thêm vào hàng đợi ưu tiên Close, mục đích thêm vào Close để biết nút này đã được duyệt trong các bước tiếp theo, thuật toán sẽ không lặp lại các nút đã được duyệt

3.2 Chi tiết về các thuật toán chính.

3.2.1. Thuật toán A*

a) Khái niệm thuật toán.

Đây là thuật toán tìm kiếm có thông tin, thuật toán này tìm một đường đi từ một nút khởi đầu tới một nút đích cho trước (hoặc tới một nút thỏa mãn một điều kiện đích). Thuật toán A* kết hợp các tính năng của thuật toán UCS và thuật toán Greedy, nhờ đó giải quyết vấn đề một cách hiệu quả. A* tìm kiếm tuyến đường tốt nhất thông qua không gian trạng thái sử dụng hàm “đánh giá heuristic”, thuật toán này mở rộng ít cây tìm kiếm và cung cấp kết quả tối ưu nhanh hơn. Thuật toán A* tương tự như thuật toán UCS ngoại trừ việc sử dụng hàm đánh giá $g(x) + h(x)$ thay vì $g(x)$. Việc làm rõ về các hàm đánh giá này sẽ được thể hiện ở phần sau.

b) Mô tả thuật toán.

A* lưu giữ các đường đi qua đồ thị, bắt đầu từ nút xuất phát. Tập lời giải này được lưu trong một hàng đợi ưu tiên (priority queue). Thứ tự ưu tiên gán cho một đường đi x được quyết định bởi hàm $f(x) = g(x) + h(x)$.

Trong đó, $g(x)$ là chi phí đường đi cho đến nút hiện tại. Nghĩa là tổng trọng số của các cạnh đã đi qua, $h(x)$ là hàm đánh giá heuristic về khoảng cách nhỏ nhất để đến đích từ nút hiện tại theo đường chim bay trên bản đồ giao thông, cụ thể hàm h được tính bằng công thức:

$$h = \text{sqrt}((\text{current.x} - \text{goal.x})^2 + (\text{current.y} - \text{goal.y})^2)$$

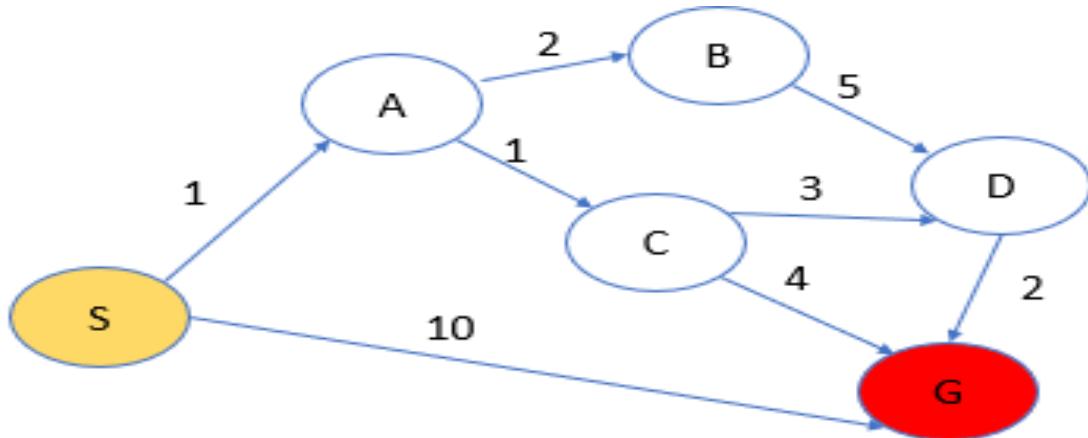
Trong đó current là vị trí hiện tại, goal là vị trí đích và sqrt là hàm căn bậc 2. Hàm $f(x)$ có giá trị càng thấp thì độ ưu tiên của x càng cao.

Thuật toán A* sử dụng hàng đợi ưu tiên Open để lưu các nút đã được sinh ra nhưng chưa được xét đến và hàng đợi ưu tiên Close để lưu các nút đã duyệt qua. Quy tắc để thêm một nút mới sinh ra vào trong hàng đợi ưu tiên Open là nút đó không được có trong Open và Close.

Thuật toán A* hoạt động như sau:

1. Khởi tạo hàng đợi ưu tiên Open và Close. Sau đó thêm trạng thái đầu vào Open.
2. Khởi tạo vòng lặp
 - Kiểm tra nếu Open rỗng thì trả về False (Thoát chương trình, tìm kiếm thất bại)
 - Lấy trạng thái p đầu trong hàng đợi Open (Xoá p khỏi Open)
 - Nếu p là trạng thái kết thúc thì trả về True (Thoát chương trình tìm kiếm thành công)
 - Chuyển p vào trong Close và tạo trạng thái kế tiếp q sau p.
 - Nếu q không có trong close
 - Nếu q không có trong Open
 - $g(q) = g(p) + \text{Cost}(p, q)$
 - $f(q) = g(q) + h(q)$
 - $q.\text{par} = p$ (Định cha của q là p)
 - Thêm q vào Open
 - Nếu q đã có trong Open
 - Nếu $g(q) > g(p) + \text{Cost}(p, q)$
 - $g(q) = g(p) + \text{Cost}(p, q)$
 - $f(q) = g(q) + h(q)$
 - $q.\text{par} = p$ (Định cha của q là p)

Minh Họa:



Hình 1: Đồ thị minh họa thuật toán A*

State	$h(x)$
S	5
A	3
B	4
C	2
D	6
G	0

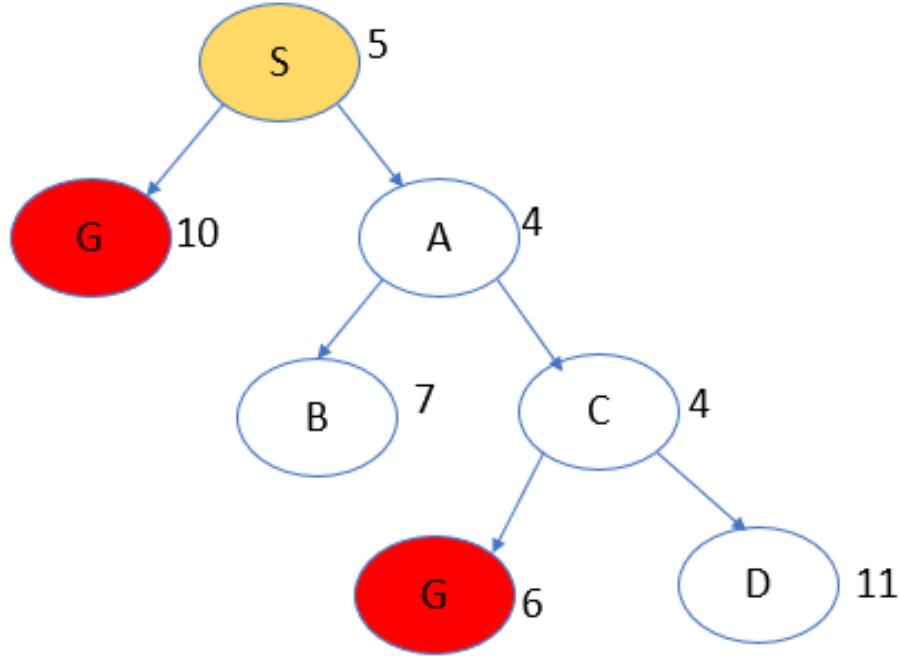
Hình 2: Giá trị heuristic

- Đỉnh bắt đầu là S
- Đỉnh kết thúc là G
- Ước lượng khoảng cách từ đỉnh hiện tại cho đến đỉnh kết thúc $f(x)=g(x)+h(x)$ trong đó h là khoảng cách ngắn nhất từ đỉnh hiện tại đến đích. Ví dụ: $f(A) = 1 + 3$.

Bước	P	Các đỉnh kề với P	Open	Close
0			S5	O
1	S	A, G	A4, G10	S
2	A	B, C	C4, B7, G10	S, A
3	C	D, G	G6, B7, D11	S, A, C
4	G(dừng)			

Hình 3: Bảng quá trình duyệt đồ thị với thuật toán A*

Cây tìm kiếm ứng với đồ thị trên.



*Hình 4: Kết quả cây tìm kiếm của thuật toán A**

c) Nhận xét.

Độ phức tạp về thời gian, trong một đồ thị có thể phải di chuyển hết các cạnh để đi từ trạng thái nguồn đến trạng thái đích. Vì vậy, độ phức tạp trong trường hợp xấu nhất là X trong đó X là số cạnh của đồ thị.

Độ phức tạp về bộ nhớ. Trong trường hợp xấu nhất chúng ta có thể xem hết tất cả các đỉnh trong hàng đợi Open. Vì vậy yêu cầu bộ nhớ trong trường hợp xấu nhất là $O(V)$, với V là tổng số đỉnh của đồ thị.

Ưu điểm: đây là một thuật chứa cả tìm kiếm chiều sâu, tìm kiếm chiều rộng và những nguyên lý Heuristic. Nên A* gần như tìm đường đi ngắn nhất nếu tồn tại một đường đi như vậy.

Nhược điểm: Tốn nhiều bộ nhớ để lưu lại những trạng thái đã đi qua. Và Thuật toán tìm kiếm A * không phải lúc nào cũng tìm ra đường đi ngắn nhất, vì nó phụ thuộc nhiều vào heuristics. Ngoài ra A* không đảm bảo sẽ chạy nhanh hơn các thuật toán tìm kiếm đơn giản hơn. Trong một môi trường dạng mê cung, cách duy nhất để đến đích có thể là trước hết phải đi về phía xa đích và cuối cùng mới quay lại. Trong trường hợp đó, việc thử các nút theo thứ tự "gần đích hơn thì được thử trước" có thể gây tốn thời gian.

d) Ứng dụng của thuật toán.

Thiết kế các trò chơi (Video games)

Các bài toán tìm đường đi (Pathing Problem)

Thiết lập chuyển động của Robot (Robot motion planning)

Phân tích ngôn ngữ (Language Analysis)

Dịch tự động (Machine Translation)

Nhận dạng giọng nói (Speech recognition)

3.2.1 Thuật toán UCS(Uniform Cost Search)

a) Khái niệm thuật toán.

Thuật toán tìm kiếm với chi phí cực tiểu (UCS) là một cách duyệt cây dùng cho việc duyệt hay tìm kiếm trên cây có trọng lượng chi phí, đây là thuật toán tìm kiếm không có thông tin vì nó không xem xét trạng thái của nút hoặc không gian tìm kiếm. Thuật toán sẽ phát triển các nút chưa xét có chi phí thấp nhất – các nút được xét theo thứ tự chi phí tăng dần. Khi đó thi có chi phí ở mỗi bước là như nhau thì thuật toán trở thành phương pháp tìm kiếm theo chiều rộng.

b) Mô tả thuật toán

Như đã được đề cập ở thuật toán A*, thuật toán UCS và thuật toán A* khá giống nhau. Thuật toán UCS cũng lưu giữ các đường đi qua đồ thị, bắt đầu từ nút xuất phát và tập lời giải này cũng được lưu trong một hàng đợi ưu tiên (priority queue), nhưng thứ tự ưu tiên được gán trong hàng đợi lúc này là hàm $g(x)$. Trong đó $g(x)$ là chi phí đường đi từ nút gốc cho đến nút hiện tại. Nghĩa là tổng trọng số của các cạnh đã đi qua. Hàm $g(x)$ của từng nút được xác định đơn giản bằng công thức:

$$\text{Child.g} = \text{Parent.g} + \text{Cost(Parent} \rightarrow \text{Child)}$$

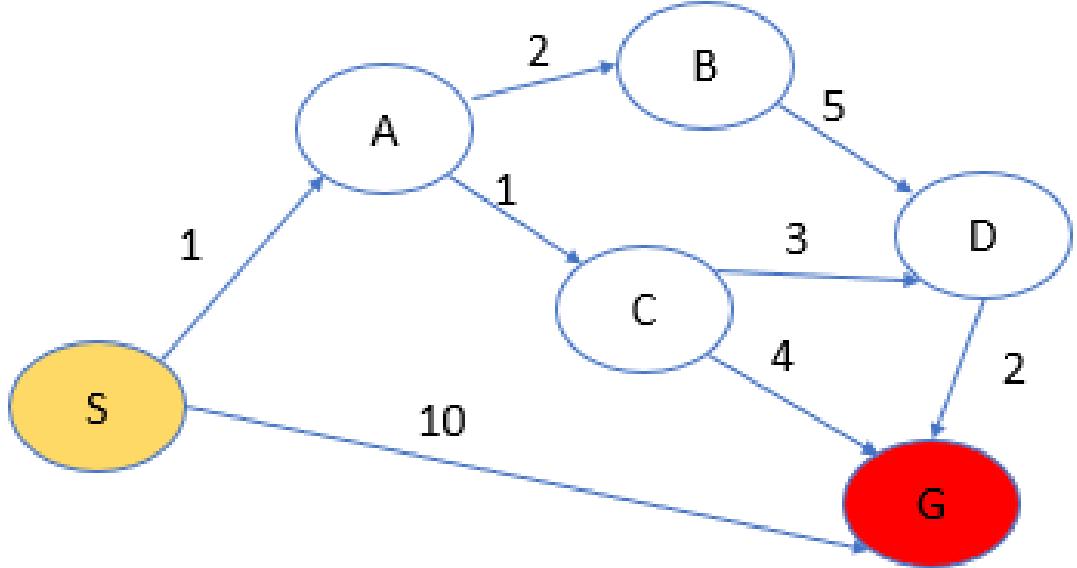
Trong đó: Child là nút con(nút kế tiếp được sinh ra), Parent là nút cha(nút hiện tại), Cost(Parent \rightarrow Child) chi phí đi từ nút cha(hiện tại) \rightarrow nút con(nút kế tiếp). Hàm $g(x)$ có giá trị càng thấp thì độ ưu tiên càng cao.

Tương tự như thuật toán A*, thuật toán UCS sử dụng hàng đợi ưu tiên Open để lưu các nút đã được sinh ra nhưng chưa được xét đến và hàng đợi ưu tiên Close để lưu các nút đã duyệt qua. Quy tắc để một nút có được thêm vào hàng đợi ưu tiên Open là nút đó không có trong hàng đợi ưu tiên Open và Close, với trường hợp nút đó đã tồn tại trong hàng đợi ưu tiên Open, cần cập nhật lại hàm $g(x)$ của nút đó trong hàng đợi ưu tiên Open nếu chi phí hàm $g(x)$ hiện tại nhỏ hơn chi phí hàm $g(x)$ của nút đó trong hàng đợi ưu tiên Open.

Thuật toán UCS hoạt động như sau:

1. Khởi tạo hàng đợi ưu tiên Open và Close. Sau đó thêm trạng thái đầu vào hàng đợi Open
2. Khởi tạo vòng lặp:
 - Kiểm tra nếu hàng đợi Open không còn phần tử thì trả về False (thoát chương trình, tìm kiếm thất bại)
 - Lấy trạng thái p đầu trong hàng đợi ưu tiên Open, xóa p khỏi Open
 - Nếu p là trạng kết thúc thì trả về True (Tìm kiếm thành công)
 - Chuyển p vào trong Close và tạo trạng thái q kế tiếp sau p.
 - Nếu q không có trong Close
 - ❖ Nếu q không có trong Open
 - $g(q) = g(p) + \text{Cost}(p, q)$
 - $q.\text{par} = p$ (Đỉnh cha của q là p)
 - Thêm q vào Open
 - ❖ Nếu q đã có trong Open
 - Nếu $g(q) > g(p) + \text{Cost}(p, q)$
 - $g(q) = g(p) + \text{Cost}(p, q)$
 - $q.\text{par} = p$ (Đỉnh cha của q là p)

Minh họa:



Hình 5: Đồ thị minh họa thuật toán UCS

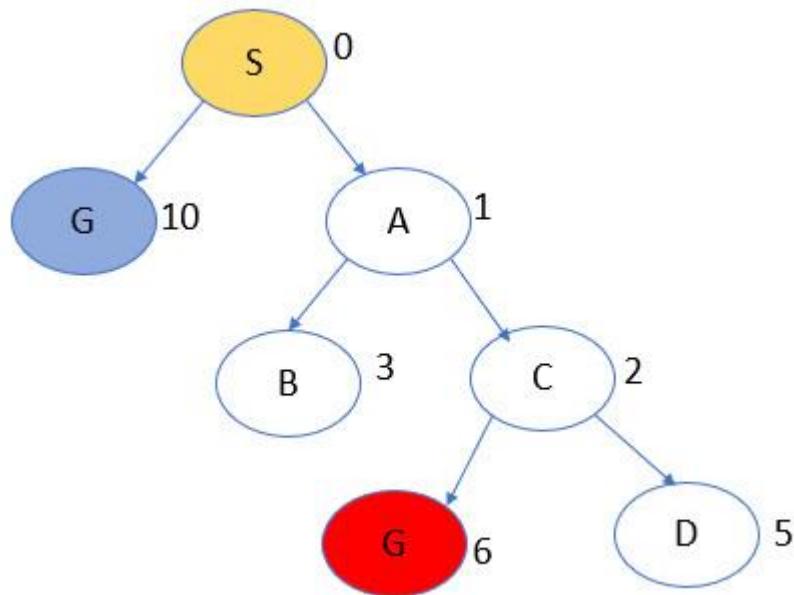
- Đỉnh bắt đầu là S
- Đỉnh kết thúc là G
- Hàm đánh g(x) xác định bằng chi phí từ đỉnh bắt đầu đến đỉnh hiện đang xét

Bước	P	Các đỉnh kề với P	Open	Close
0			S0	\emptyset
1	S	A, G	A1, G10	S
2	A	B, C	C2, B3, G10	S, A
3	C	D, G	B3, D5, G6	S, A, C
4	B	D	D5, G6	S, A, C, B
5	D	G	G6	S, A, C, B, D
6	G(dừng)			

Hình 6: Bảng quá trình duyệt đồ thị với thuật toán UCS

Ở bảng minh họa cách thức thuật toán UCS hoạt động ở trên, cần chú ý ở bước số 3, với đỉnh C có 2 đỉnh kề cận là D, G. Tuy G đã tồn tại trong hàng đợi ưu tiên Open nhưng vì chi phí cho G trong hàng đợi là 10, nhỏ hơn chi phí hiện tại là 6, vì vậy G sẽ được cập nhật lại vào hàng đợi ưu tiên Open. Ở bước thứ 4 và 5 cũng tương tự như vậy. Ở bước 4, đỉnh B kề với 2 đỉnh A và D, D đã tồn tại trong Open và chi phí của D trong Open là 5, nhỏ hơn so với chi phí hiện đang xét là 10, vì vậy D không được cập nhật. Tương tự như vậy, đỉnh G không được cập nhật ở bước số 5.

Cây tìm kiếm ứng với đồ thị trên:



Hình 7: Kết quả cây tìm kiếm của thuật toán UCS

c) Nhận xét

Uniform Cost Search(UCS) là một loại thuật toán tìm kiếm không có thông tin và là giải pháp tối ưu để tìm đường dẫn từ nút gốc đến nút đích với chi phí tích lũy thấp nhất trong không gian tìm kiếm có trọng số với các nút có chi phí truyền tải khác nhau.

Thuật toán UCS tương tự như thuật toán A* nhưng sử dụng hàm $g(x)$ là chi phí từ đỉnh xuất phát đến đỉnh hiện tại để đánh giá. UCS là một trường hợp đặc biệt của A* trong đó đánh giá heuristic là một hàm hằng $h(x) = 0$ với mọi x .

Trong trường hợp chi phí tất cả các nút là như nhau, thuật toán UCS tương đương với thuật toán duyệt theo chiều rộng BFS, vì vậy đối với trường hợp đỉnh kết thúc ở xa đỉnh gốc sẽ phải sinh ra rất nhiều nút của cây tìm kiếm, có thể dẫn đến treo hoặc hết bộ nhớ nếu số nút quá lớn.

Độ phức tạp về thời gian: Gọi C^* là Chi phí của giải pháp tối ưu và ϵ là mỗi bước để tiến gần hơn đến nút mục tiêu. Khi đó số bước là $= C^*/\epsilon + 1$. Vì vậy độ phức tạp trong trường hợp xấu nhất của thuật toán là $O(b1 + [C^*/\epsilon])$

Độ phức tạp về không gian: Logic tương tự là đối với độ phức tạp không gian, vì vậy, độ phức tạp không gian trong trường hợp xấu nhất của thuật toán UCS là $O(b1 + [C^*/\epsilon])$

Ưu điểm: Giúp tìm ra đường đi với chi phí tích lũy thấp nhất bên trong một đồ thị có trọng số có chi phí khác nhau. Được coi là giải pháp tối ưu vì ở mỗi trạng thái, đường đi có chi phí thấp nhất được chọn

Nhược điểm: Hàng đợi Open phải sử dụng hàng đợi ưu tiên vì cần ưu tiên trạng thái có chi phí thấp nhất. Bộ nhớ được yêu cầu lớn theo cấp số nhân. Thuật toán có thể bị treo đối với những bài toán có số đỉnh lớn và chi phí các nút là như nhau

d) *Ứng dụng của thuật toán.*

Giải quyết bài toán duyệt biều đồ, duyệt cây có nhu cầu về chi phí tối ưu:

Ứng dụng trong các bài toán tìm đường

Điều hướng Robot

Thiết kế Protein

Lập lịch (Scheduling/Manufacturing)

PHẦN 4: THỰC NGHIỆM, ĐÁNH GIÁ, PHÂN TÍCH KẾT QUẢ

4.1 Mô tả dữ liệu và các thuật toán cài đặt

4.1.1 Mô tả dữ liệu

Dữ liệu đầu vào gồm một tệp txt lưu toạ độ của các đỉnh của đồ thị với dòng đầu là số lượng đỉnh của đồ thị và n dòng tiếp theo lưu lần lượt giá toạ độ x, y của đỉnh thứ n. Và một tệp txt dùng để lưu các đỉnh kề của đồ thị với dòng đầu tiên là số lượng đỉnh của đồ thị và các dòng tiếp theo với số đầu tiên sẽ lưu đỉnh x và các số tiếp theo sẽ lần lượt là đỉnh kề với x và trọng số.

4.1.2 Cài đặt thuật toán

a) Thuật toán A Star(A*)

Tạo hàng đợi ưu tiên Open và Close

```
Open = PriorityQueue()
Close = PriorityQueue()
```

Tính HX từ đỉnh đầu đến đỉnh đích và thêm đỉnh đầu vào hàng đợi

```
S.h = HX(S.name, G.name)
Open.put(S)
```

Khởi tạo vòng lặp và thực hiện:

+ Kiểm tra nếu Open rỗng thì trả về là False(Tìm kiếm thất bại)

```
if(Open.empty() == True):
    print("Tìm kiếm thất bại")
    return False
```

+ Lấy phần tử đầu ra hàng đợi Open gán vào O và đưa vào Close

```
O = Open.get()
Close.put(O)
```

+ Kiểm tra nếu O là G(trạng thái đích) thì in ra đường đi, tìm kiếm thành công

```
if(Sosanhbang(O, G) == True):
    print("Tìm kiếm thành công")
    DuongDi(O)
    print("Khoảng cách: ", (O.g))
    return True
```

+ Khởi tạo vòng lặp sinh ra các đỉnh q kè với O và thực hiện kiểm tra nếu đỉnh kè q không có trong Close và Open thì thực hiện thêm q vào Open. Ngược lại nếu đã có trong Open thì kiểm tra nếu $g(q)$ lớn hơn $g(O) + \text{Cost}(O, q)$ thì cập nhật lại q.

```
i = 0
while i < len(data[0.name]):
    name = data[0.name][i]
    node_tmp = Node(name=name)
    vt1 = KiemTraTonTai(node_tmp, Open)
    vt2 = KiemTraTonTai(node_tmp, Close)
    if(vt2 == -1 and vt2 != None):
        if(vt1 == -1 and vt1 != None):
            node_tmp.g = 0.g + data[0.name][i+1]
            node_tmp.h = HX(node_tmp.name, G.name)
            node_tmp.par = 0
            Open.put(node_tmp)
        elif(Open.queue[vt1].g > 0.g + data[0.name][i+1]):
            Open.queue.remove(node_tmp)
            node_tmp.g = 0.g + data[0.name][i+1]
            node_tmp.h = HX(node_tmp.name, G.name)
            node_tmp.par = 0
            Open.put(node_tmp)
    i += 2
    ➤ Hàm kiểm tra tồn tại
```

```
def KiemTraTonTai(tmp, c):
    if(tmp == None):
        return None
    for i in range(c.qsize()):
        if(tmp == c.queue[i]):
            return i
    return -1
```

➤ Hàm tính HX

```
def HX(X, G):
    vitriX = int(X) - 1
    vitriG = int(G) - 1
    vitriX_x = Todo[vitriX][0]
```

```

vitriX_y = Todo[vitriX][1]
vitriG_x = Todo[vitriG][0]
vitriG_y = Todo[vitriG][1]
return math.sqrt((vitriX_x - vitriG_x)**2 + (vitriX_y - vitriG_y)**2)

```

➤ Hàm so sánh để thêm vào hàng đợi ưu tiên.

```

def __lt__(self, other):
    if(other == None):
        return False
    return self.g + self.h < other.g + other.h

```

b) Thuật toán Uniform Cost Search (UCS)

Tạo hàng đợi ưu tiên Open, Close và thêm đỉnh đầu vào Open

```

Open = PriorityQueue()
Close = PriorityQueue()
Open.put(S)

```

Khởi tạo vòng lặp và thực hiện:

+ Kiểm tra nếu Open rỗng thì trả về False, tìm kiếm thất bại

```

if(Open.empty() == True):
    print("Tìm kiếm thất bại")
    return False

```

+ Lấy phần tử đầu hàng đợi Open gán vào O và thêm vào Close

```

O = Open.get()
Close.put(O)

```

+ Kiểm tra nếu O là trạng thái cuối thì in ra đường đi và trả về True, tìm kiếm thành công.

```

if(Sosanhbang(O, G) == True):
    print("Tìm kiếm thành công")
    DuongDi(O)
    print("Khoảng cách: ", (O.g))
    return True

```

+ Khởi tạo vòng lặp sinh ra các đỉnh kề với O và thực hiện kiểm tra nếu đỉnh kề q không có trong Close và Open thì thực hiện thêm q vào Open. Ngược

lại nếu đã có trong Open thì kiểm tra nếu $g(q)$ lớn hơn $g(O) + \text{Cost}(O, q)$ thì cập nhật lại q .

```
i = 0
while i < len(data[0.name]):
    name = data[0.name][i]
    node_tmp = Node(name=name)
    vt1 = KiemTraTonTai(node_tmp, Open)
    t2 = KiemTraTonTai(node_tmp, Close)
    if(vt2 == -1 and vt2 != None):
        if( vt1 == -1 and vt1 != None):
            node_tmp.g = 0.g + data[0.name][i+1]
            node_tmp.par = 0
            Open.put(node_tmp)
        elif(Open.queue[vt1].g > 0.g + data[0.name][i+1]):
            Open.queue.remove(node_tmp)
            node_tmp.g = 0.g + data[0.name][i+1]
            node_tmp.par = 0
            Open.put(node_tmp)
    i += 2
```

➤ Hàm kiểm tra tồn tại

```
def KiemTraTonTai(tmp, c):
    if(tmp == None):
        return None
    for i in range(c.qsize()):
        if(tmp == c.queue[i]):
            return i
    return -1
```

➤ Hàm so sánh thêm vào hàng đợi ưu tiên

```
def __lt__(self, other):
    if(other == None):
        return False
    return self.g < other.g
```

4.2 Trình bày các kết quả thử nghiệm

4.2.1. Map A

Lần 1: Đi từ đỉnh 1 đến 19



Hình 8: Lần 1 thuật toán A Star



Hình 9: Lần 1 thuật toán UCS

Lần 2: Đi từ đỉnh 1 đến 25



Hình 10: Lần 2 thuật toán A Star



Hình 11: Lần 2 thuật toán UCS

Lần 3: Đi từ đỉnh 25 đến 4



Hình 12: Lần 3 thuật toán A Star



Hình 13: Lần 3 thuật toán UCS

Lần 4: Đi từ đỉnh 13 đến 5



Hình 14: Lần 4 thuật toán A Star

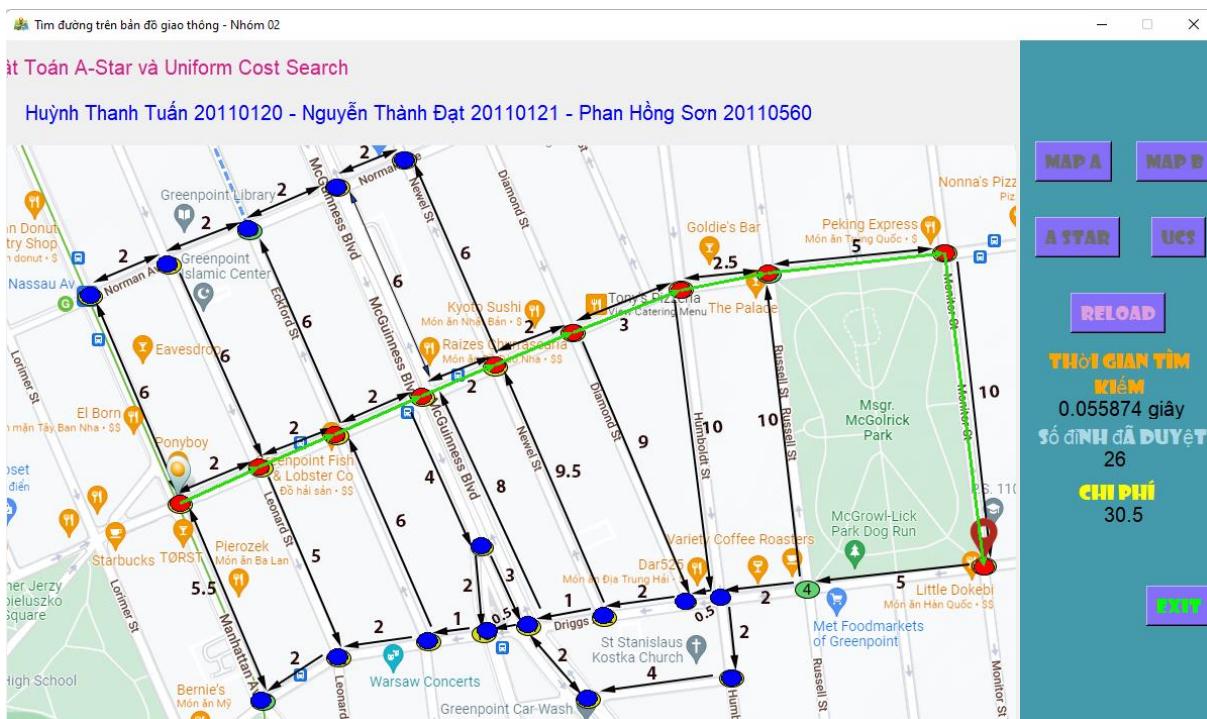


Hình 15: Lần 4 thuật toán UCS

Lần 5: Đi từ đỉnh 26 đến 1



Hình 16: Lần 5 thuật toán A Star



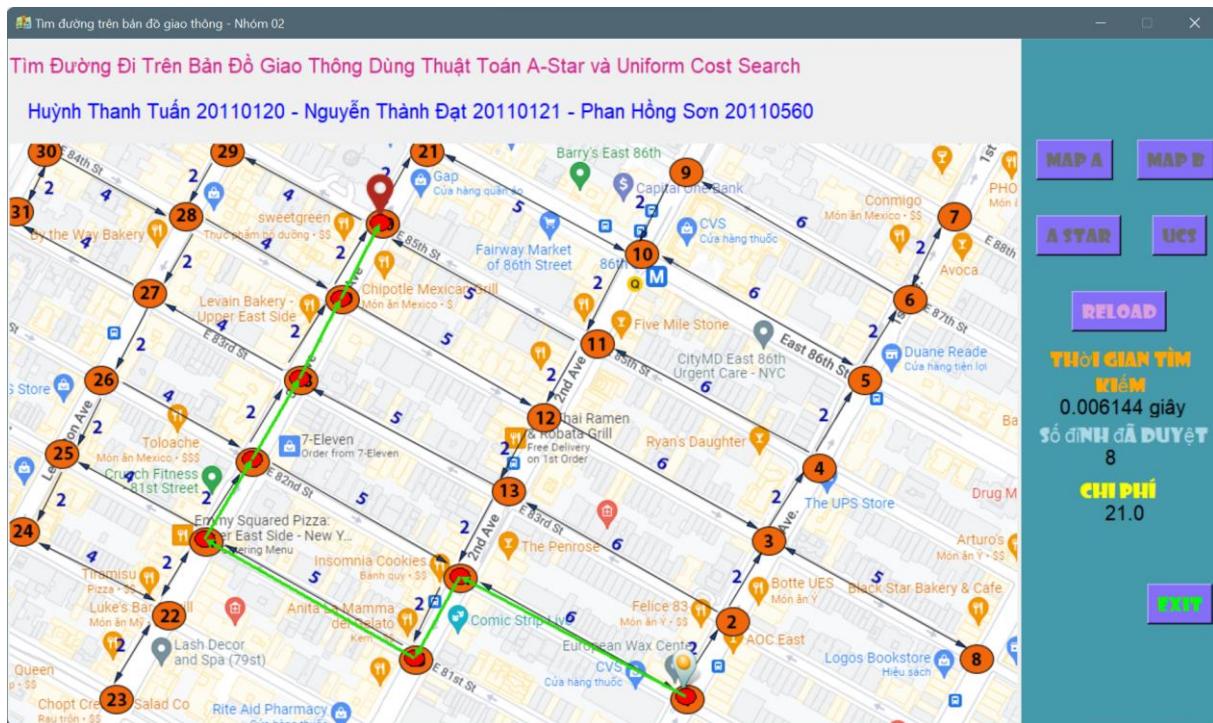
Hình 17: Lần 5 thuật toán UCS

Lần	A Star			UCS		
	Thời gian(s)	Số đỉnh duyệt	Chi phí	Thời gian(s)	Số đỉnh duyệt	Chi phí
1	0.02751	9	32.5	0.044685	23	24
2	0.03829	11	36.5	0.04953	27	27.5
3	0.040763	21	41.5	0.046005	27	41.5
4	0.018981	12	27	0.028125	24	27
5	0.038005	19	30.5	0.055874	26	30.5
Trung bình	0.0327098	14.4	33.6	0.0448438	25.4	30.1

Hình 18: Bảng tổng hợp kết quả thử nghiệm MAP A

4.2.2. Map B

Lần 1: Đi từ đỉnh 1 đến đỉnh 20

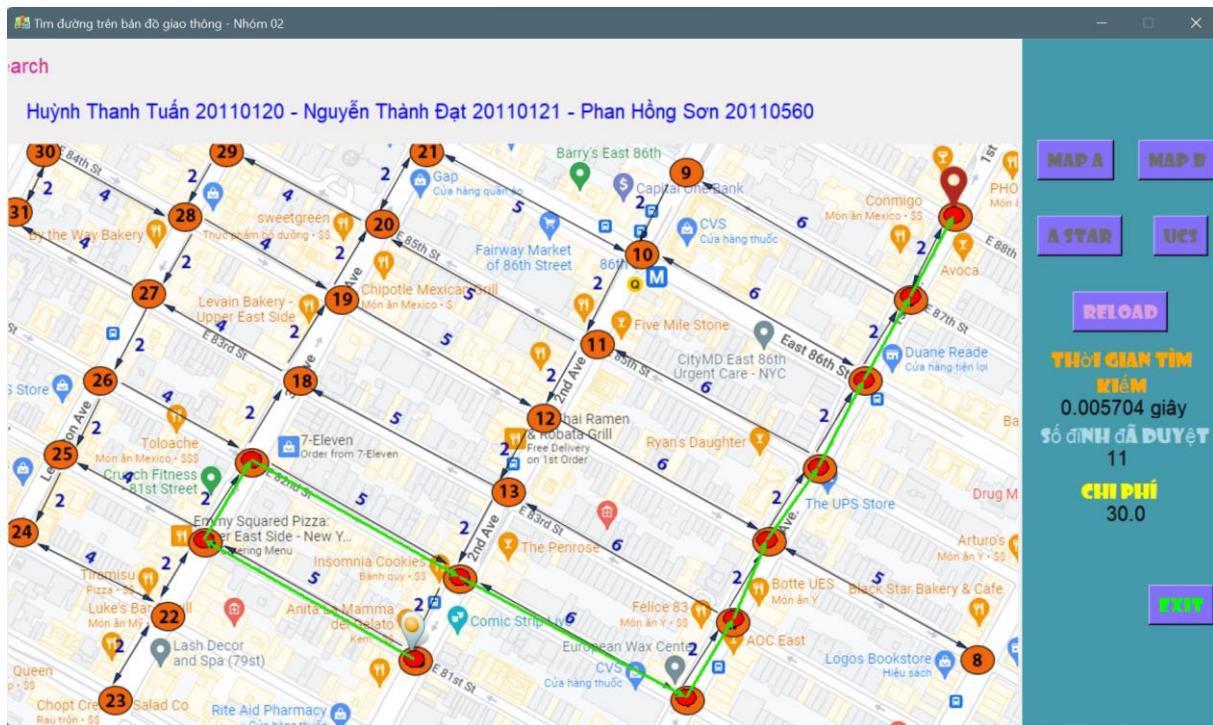


Hình 19. Lần 1 thuật toán A Star

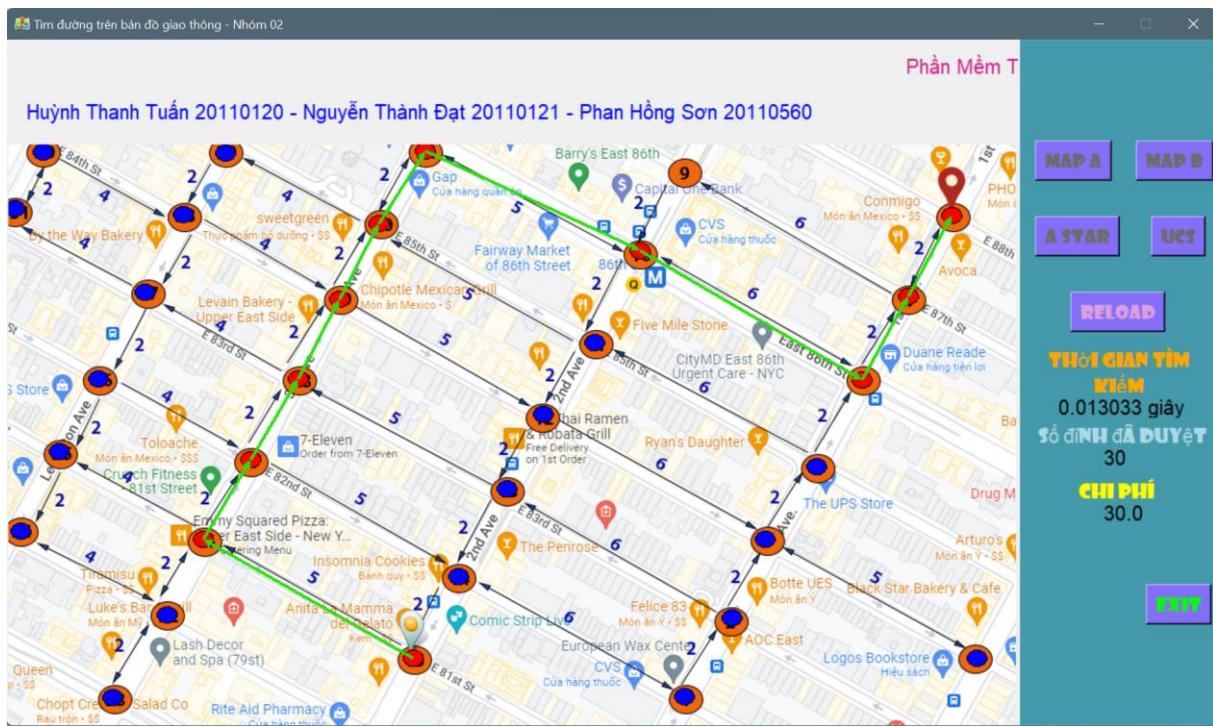


Hình 20. Lần 1 thuật toán UCS

Lần 2: Đi từ đỉnh 15 đến đỉnh 7



Hình 21. Lần 2 thuật toán A Star

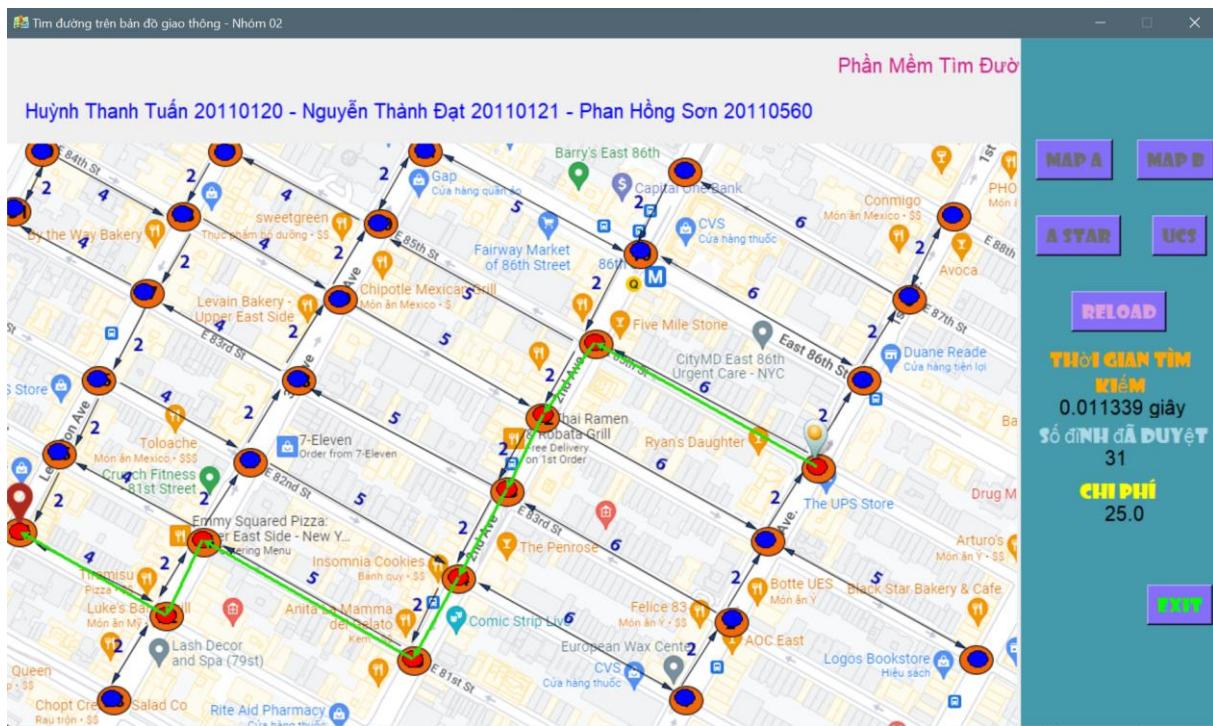


Hình 22. Lần 2 thuật toán UCS

Lần 3: Đi từ đỉnh 4 đến đỉnh 24



Hình 23. Lần 3 thuật toán A Star

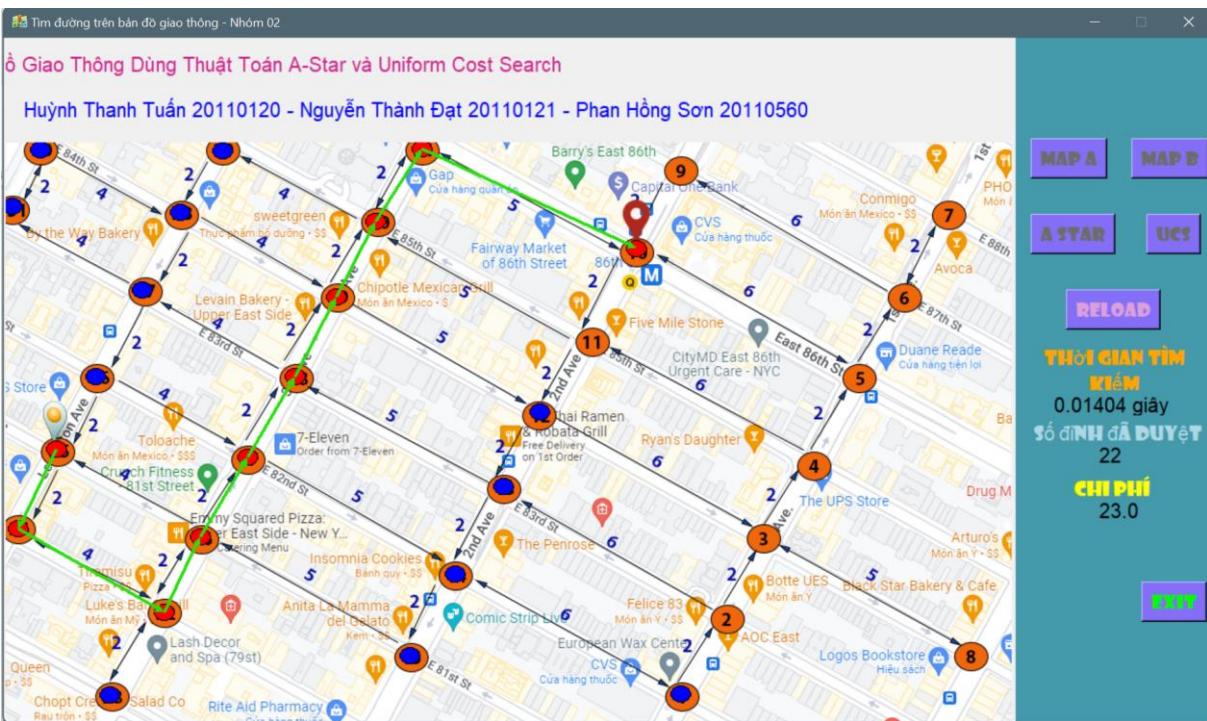


Hình 24. Lần 3 thuật toán UCS

Lần 4: Đi từ đỉnh 25 đến đỉnh 10

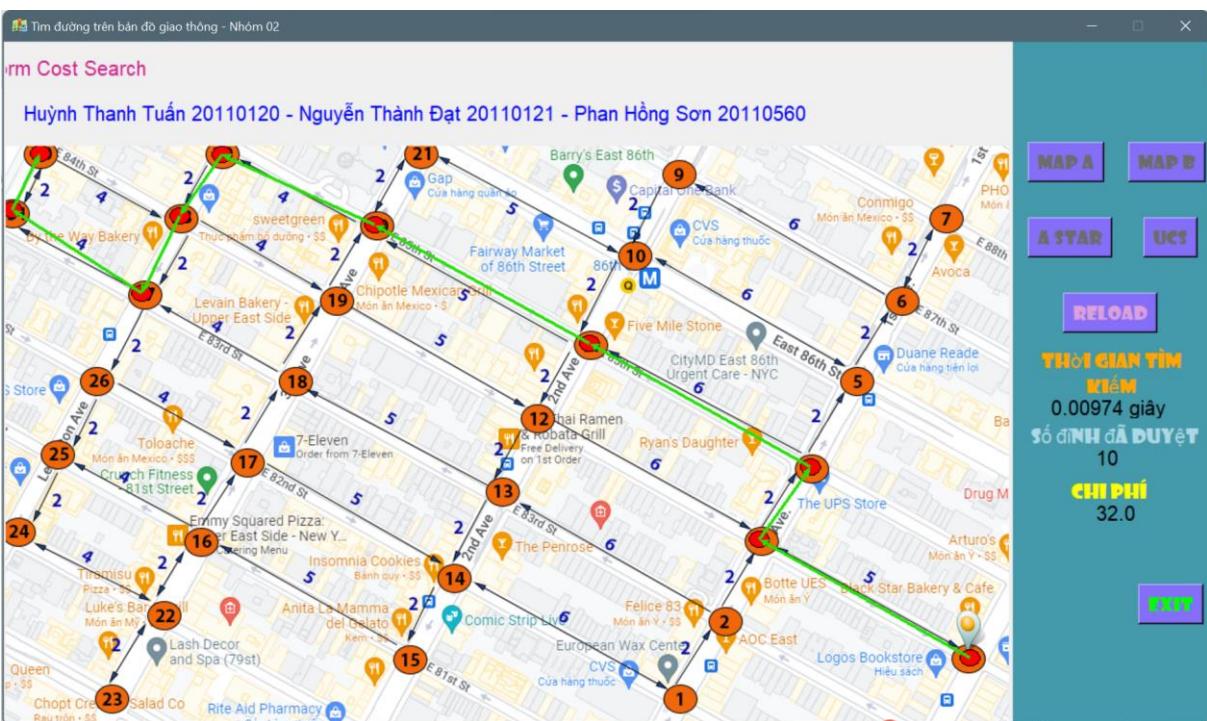


Hình 25. Lần 4 thuật toán A Star

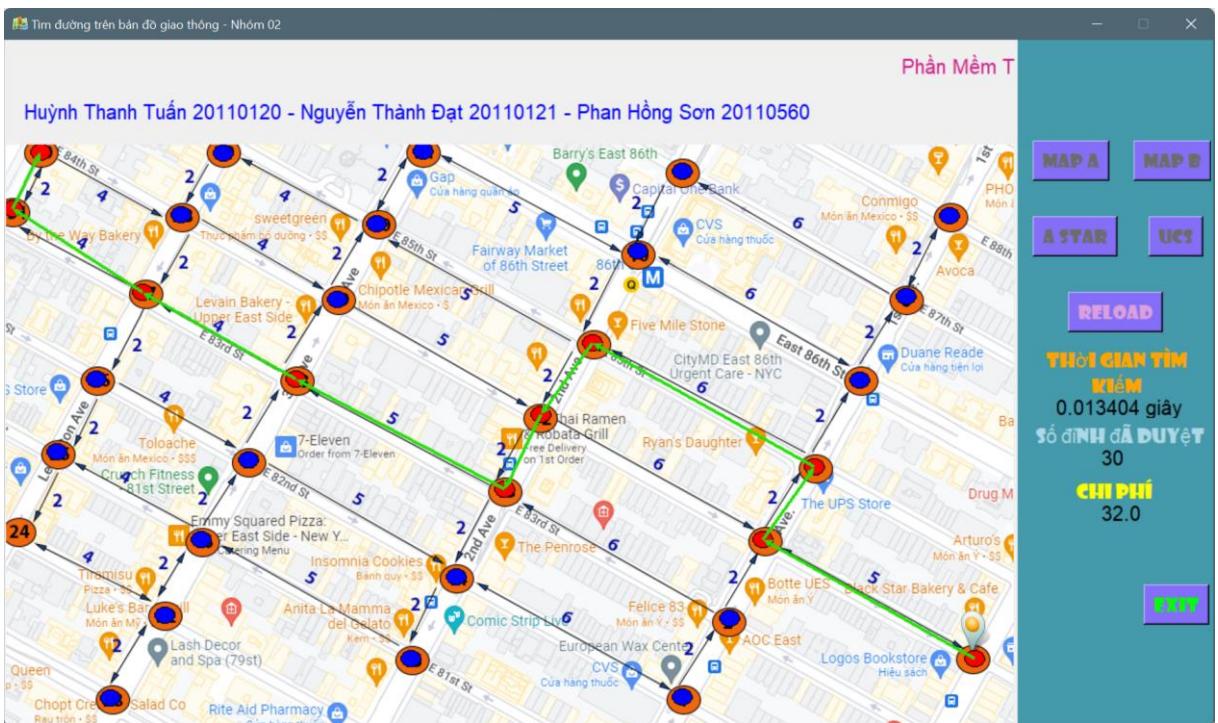


Hình 26. Lần 4 thuật toán UCS

Lần 5: Đi từ đỉnh 8 đến 30



Hình 27: Lần 5 thuật toán A Star



Hình 28. Lần 5 thuật toán UCS

Lần	A Star			UCS		
	Thời gian(s)	Số đỉnh duyệt	Chi phí	Thời gian(s)	Số đỉnh duyệt	Chi phí
1	0.006144	8	21	0.010509	23	17
2	0.005704	11	30	0.013033	30	30
3	0.007968	9	25	0.011339	31	25
4	0.005119	11	23	0.01404	22	23
5	0.00974	10	32	0.013404	30	32
Trung bình	0.006935	9.8	26.2	0.012465	27.2	25.4

Hình 29. Bảng tổng hợp kết quả thử nghiệm MAP B

Nhận xét: Thuật toán A* do có hàm $h(x)$ giúp định hướng khoảng cách đến điểm kết thúc nên số lượng đỉnh duyệt qua là ít hơn dẫn đến thời gian cho ra kết quả là nhanh hơn so với thuật toán UCS số lượng đỉnh duyệt qua nhiều nên thời gian chạy chậm hơn. Thuật toán A* không đảm bảo cho ra chi phí là thấp nhất so với UCS cụ thể là trong lần thứ 1 và 2 của map A, lần thứ 1 của Map B. Mặc dù A* cho ra kết quả nhanh hơn UCS nhưng UCS lại có chi phí tốt hơn.

4.3 Giải thích, hướng dẫn, thực thi phần mềm

a) Giải thích

- Điểm bắt đầu: Ghim bắt đầu viên màu trắng chính giữa màu cam.



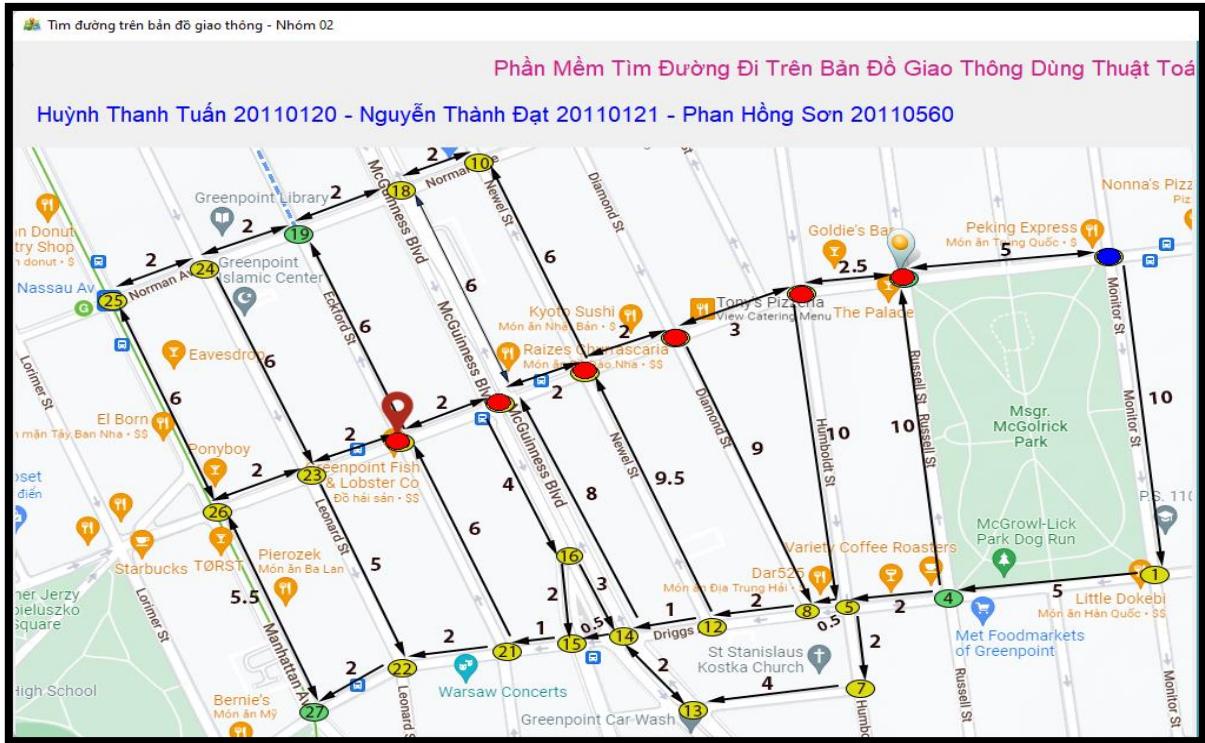
Hình 30: Điểm xuất phát

- Điểm kết thúc: Ghim màu đỏ.



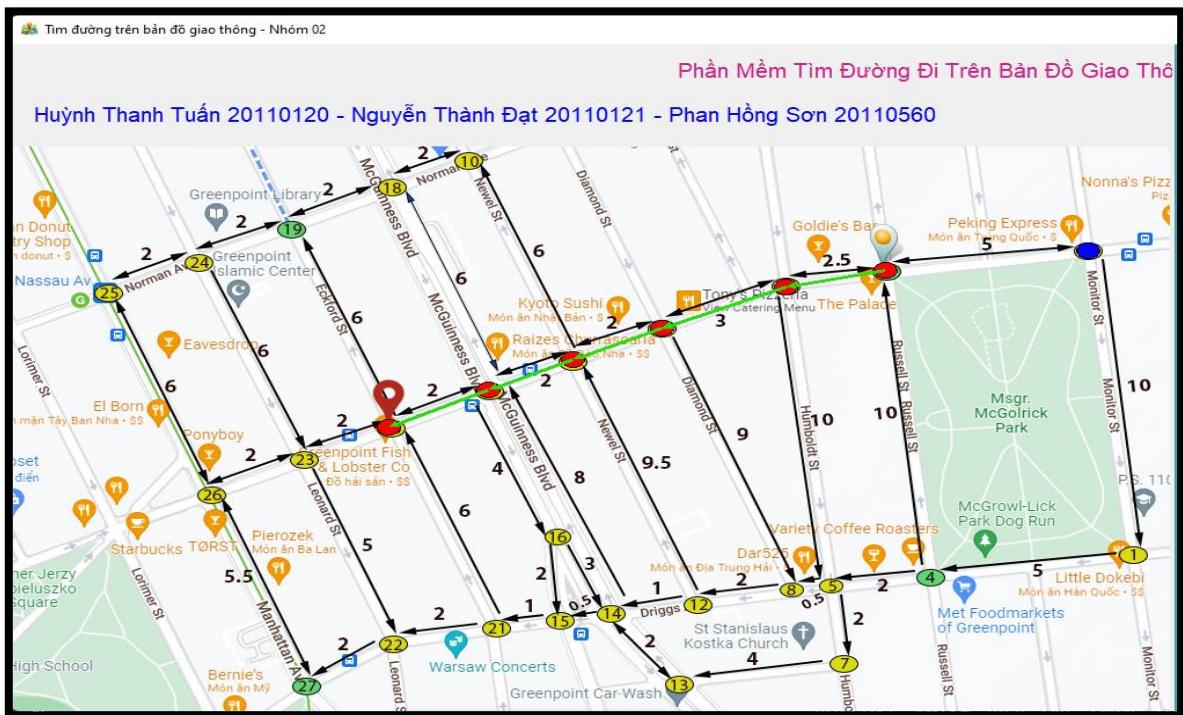
Hình 31: Điểm đến

- Các vị trí duyệt qua: chấm hình oval màu xanh và màu đỏ. Trong đó các hình oval màu đỏ sẽ là các đỉnh của đường đi tìm được.



Hình 32: Các đỉnh duyệt qua

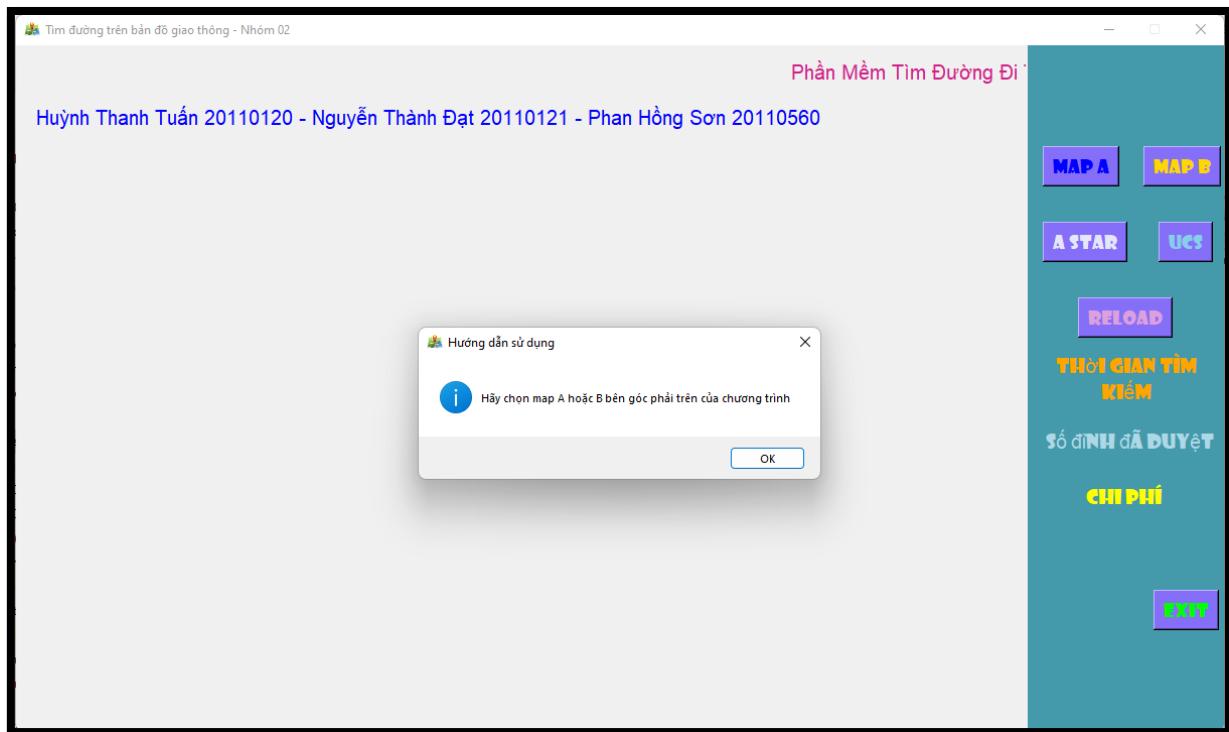
- Kết quả khi tìm được đường đi.



Hình 33: Kết quả tìm đường đi

b) Hướng dẫn thực thi phần mềm

Bước 1: Khi chạy file tên “GiaoDien.py” sẽ xuất hiện giao diện như hình bên dưới.



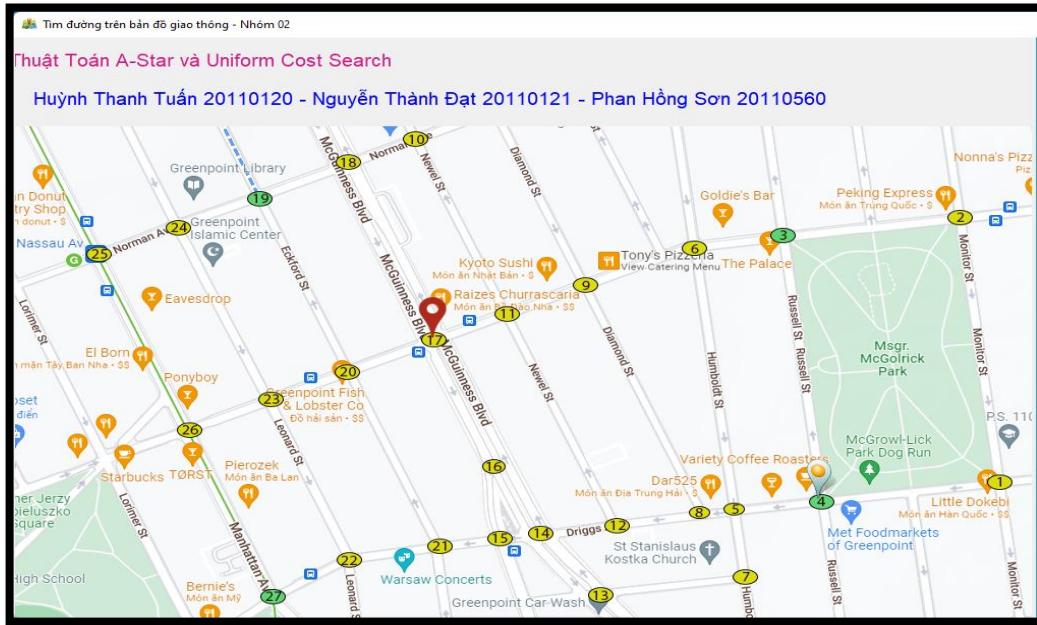
Hình 34: Giao diện khởi động

Bước 2: Người dùng click chuột phải chọn Map ở góc bên phải



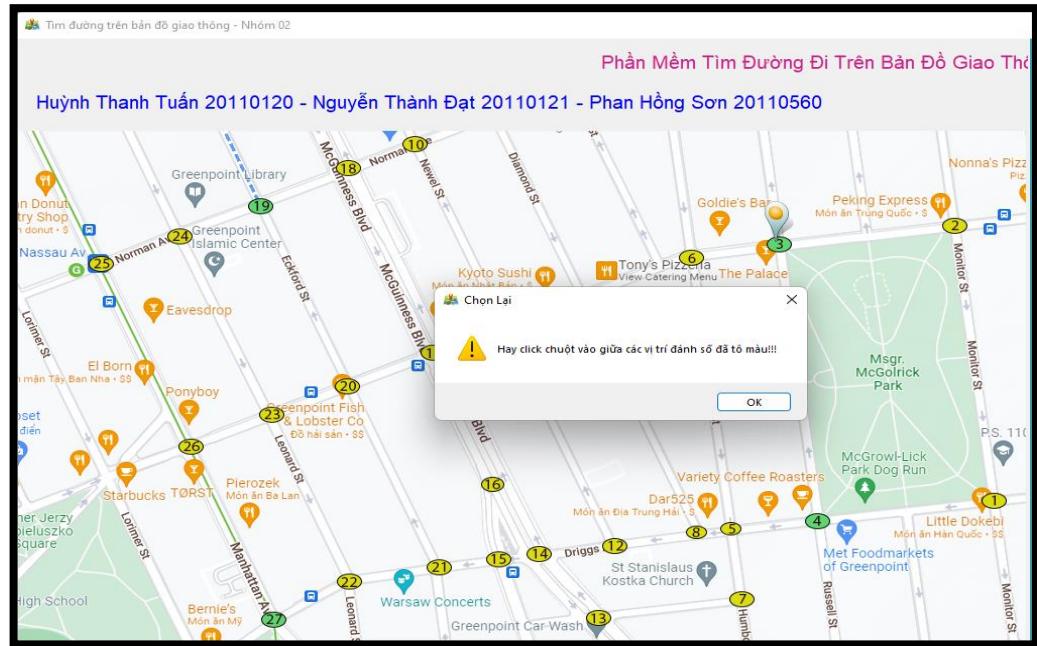
Hình 35: Kết quả chọn Map A

Bước 3: Người dùng click chuột phải chọn địa điểm đầu và cuối là các ô đã được tô màu và đánh số.



Hình 36: Chọn điểm đầu

- ❖ Trường hợp người dùng chọn ngoài ô đã được tô màu và đánh số sẽ xuất hiện hộp thoại thông báo



Hình 37: Hộp thoại thông báo

Bước 4: Sau khi chọn điểm đầu và cuối xong. Người dùng click chuột phải để chọn thuật toán chạy “A STAR” hoặc “UCS” để tìm đường đi kết quả tìm kiếm sẽ hiện thị trên giao diện. Người dùng có thể click chuột phải vào nút “RELOAD” để đặt lại chương trình như lúc mới khởi động Hoặc nhấn nút “EXIT” để thoát chương trình.



Hình 38: Kết quả chạy

PHẦN 5: KẾT LUẬN

5.1 Đánh giá những kết quả đã thực hiện được

Đã cài đặt thành công các thuật toán A star và Uniform Cost Search cho bài toán tìm đường trên bản đồ giao thông. Tạo giao diện đồ họa giúp người dùng dễ sử dụng và thao tác thông qua thư viện tkinter. Phân tích và so sánh được sự khác nhau giữa thuật toán tìm kiếm có thông tin với thuật toán A* và thuật toán tìm kiếm mù với thuật toán UCS về thời gian thực thi cho ra kết quả tìm đường đi, số lượng đỉnh duyệt qua và chi phí đường đi

5.2 Định hướng phát triển

Mở rộng bản đồ trên một khu vực rộng lớn hơn.

Làm giao diện thân thiện và bắt mắt hơn.

Thêm một vài thuật toán để xem sự khác nhau.

Làm phần mềm tìm kiếm đường đi được sử dụng rộng rãi ngoài thực tế.

TÀI LIỆU THAM KHẢO:

1. Bài giảng chương 6: Informed Search Algorithms của thầy PGS.TS. Hoàng Văn Dũng. Vị trí tham khảo trang số 23
2. SHICHIKI LÊ, 08/08/2020, Thuật Giải A*, Vị trí tham khảo mã giải Link: <https://www.stdio.vn/giai-thuat-lap-trinh/thuat-giai-a-DVnHj>
3. Wikipedia, 23/08/2020, Tìm kiếm chi phí đều, Vị trí tham khảo mã giải Link:https://vi.wikipedia.org/wiki/T%C3%ACm_ki%C3%A9m_chi_ph%C3%AD_deu
4. Education 4u, 30/08/2019, A Star algorithm | Example | Informed search | Artificial intelligence | Lec-21 | Bhanu Priya
Link: <https://www.youtube.com/watch?v=PzEWHH2v3TE>
5. Phở Code, 30/01/2016, Tkinter – Giới thiệu về Tkinter
Link: <https://phocode.com/python/tkinter/tkinter-gioi-thieu-ve-tkinter/>