

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN



MÔN HỌC: CƠ SỞ TRÍ TUỆ NHÂN TẠO
ĐỒ ÁN THỰC HÀNH 1 - ROBOT TÌM ĐƯỜNG

THÀNH VIÊN

MSSV	Họ và tên
22127076	Lương Hoàng Dung
22127249	Trần Thanh Long
22127374	Lê Thanh Tâm
22127410	Lưu Thanh Thúy

Giáo viên hướng dẫn

Thầy Bùi Duy Đăng

Thầy Lê Nguyễn Nhựt Trường

Thành phố Hồ Chí Minh, tháng 3 năm 2024

LỜI CẢM ƠN

Lời đầu tiên cho phép chúng em được cảm ơn chân thành và sâu sắc nhất đến các thầy, cô Trường Đại học Khoa Học Tự Nhiên, Đại học Quốc gia Thành phố Hồ Chí Minh, đặc biệt là các thầy, cô khoa Công nghệ thông tin của trường đã tạo điều kiện để chúng em có thể hoàn thành tốt đồ án này. Hơn hết, chúng em muốn gửi đến thầy Bùi Duy Đăng và thầy Lê Nguyễn Nhựt Trường lời cảm ơn sâu sắc vì đã nhiệt tình hướng dẫn, chỉ bảo chúng em.

Quá trình tìm hiểu các công cụ mới này đã cho chúng em được mở rộng thêm rất nhiều kiến thức về cơ sở trí tuệ nhân tạo và các thuật toán tìm đường liên quan. Không chỉ vậy, đồ án lần này còn giúp chúng em cải thiện hơn trong việc làm nhóm. Những lời góp ý của giảng viên đã giúp nhóm cải thiện, nâng cao điểm mạnh của các thành viên và cuối cùng là tạo ra sản phẩm tốt nhất.

Trong thời gian làm bản cáo, chúng em còn gặp nhiều sai sót do thiếu kinh nghiệm, mong các giảng viên bỏ qua và chỉ bảo thêm. Bản báo cáo dựa trên những gì mà chúng em đã thu thập, trải nghiệm trong quá trình làm đồ án vừa qua. Chúng em rất mong nhận được những ý kiến, đóng góp của các giảng viên để chúng em có thể học hỏi và hoàn thiện hơn.

Chúng em xin chân thành cảm ơn.

MỤC LỤC

I. GIỚI THIỆU CHUNG.....	4
1. Thông tin nhóm và phân công.....	4
2. Tóm tắt bài toán và các phần đã hoàn thành.....	4
Tóm tắt yêu cầu bài toán.....	4
Báo cáo các phần đã hoàn thành.....	5
II. QUÁ TRÌNH THỰC HIỆN.....	6
1. Tổ chức thư mục.....	6
2. Hướng dẫn sử dụng.....	6
3. Một số lưu ý.....	6
4. Giao diện đồ thị.....	7
a. Chú thích.....	8
b. Tính năng.....	9
c. Các hàm quan trọng được sử dụng.....	9
III. BÁO CÁO CÁC MỨC HOÀN THÀNH.....	9
1. Mức 1.....	9
a. Yêu cầu.....	9
b. Thực hiện.....	10
c. Quá trình chạy thử.....	10
2. Mức 2.....	13
a. Yêu cầu.....	13
b. Thực hiện.....	13
i. Thuật toán BFS.....	13
ii. Thuật toán A*.....	14
iii. Thuật toán IDA*.....	14
iv. Nhận xét sự khác nhau giữa 3 thuật toán.....	15
3. Mức 3.....	21
a. Yêu cầu.....	21
b. Thực hiện.....	21
c. Quá trình chạy thử.....	22
4. Mức 4.....	24
a. Yêu cầu.....	24
b. Chú ý.....	25
c. Thực hiện.....	25

I. GIỚI THIỆU CHUNG

1. Thông tin nhóm và phân công

Họ tên	MSSV	Phân công	Đánh giá
Lương Hoàng Dung	22127076	Cài đặt thuật toán BFS và GBFS.	100%
Trần Thanh Long	22127249	Thiết kế đồ họa, biểu diễn đường đi của robot và vật chướng ngại (mức 4).	100%
Lê Thanh Tâm	22127374	Cài đặt thuật toán A* và IDA*.	100%
Lưu Thanh Thúy	22127410	Cài đặt thuật toán A* (trường hợp có điểm đón: mức 3).	100%
Cả nhóm		Đưa ra bộ test và chỉnh sửa chương trình tối ưu nhất. Hoàn thành báo cáo đồ án.	100%

2. Tóm tắt bài toán và các phần đã hoàn thành

Tóm tắt yêu cầu bài toán:

Cho một bản đồ phẳng xOy (góc phần tư I), trên đó người ta đặt một điểm bắt đầu $S(x_s, y_s)$ và một điểm đích đến $G(x_G, y_G)$. Đồng thời đặt các chướng ngại vật là các hình đa giác lồi sao cho các đa giác không được đặt chồng lên nhau hay có điểm chung, đồng thời không đè lên hay chứa điểm đầu, điểm đích và các điểm đón.

Chọn và cài đặt các thuật toán để tìm kiếm đường đi ngắn nhất từ S đến G sao cho đường đi không được cắt xuyên qua các đa giác. Biểu diễn đồ họa có thể ở mức đơn giản nhất để người sử dụng thấy được các đa giác và đường đi.

Báo cáo các phần đã hoàn thành:

	Chi tiết	Đánh giá
Mức 1	Thành công chạy thuật toán GBFS và hiển thị đường đi robot với các dữ liệu đọc được từ file input.	100%
Mức 2	Chạy thử thành công các thuật toán BFS, A* và IDA* và hiển thị thành công đường đi robot với các dữ liệu đọc được từ file input.	100%
Mức 3	Chạy thử thành công và hiển thị được đường đi ngắn nhất qua tất cả điểm đón cho trước thông qua thuật toán A*.	100%
Mức 4	Với thuật toán BFS, thành công hiển thị đường đi tìm được đến đích với các hình vật cản chuyển động với thông tin từ file input.	100%

II. QUÁ TRÌNH THỰC HIỆN

1. Tổ chức thư mục:

Các tập tin được tổ chức thành 4 folder chính:

- **Level1_2:** chứa các file chương trình và input của mức 1 và 2.
Trong đó:
 - + 5 file text ứng với 5 file input của 5 testcase.
 - + File *Graph.py* là file hỗ trợ đọc file, và hiển thị đồ thị.
 - + Các file *GBFS.py*, *BFS.py*, *A_Star.py* và *IDA_Star.py* là các file thực hiện chạy thuật toán tương ứng.
- **Level3:** chứa các file chương trình và input của mức 3. Trong đó:
 - + 5 file text ứng với 5 file input của 5 testcase.
 - + File *Graph.py* là file hỗ trợ đọc file, và hiển thị đồ thị.
 - + File *Main.py* là file thực hiện chương trình.
- **Level4:** chứa các file chương trình và input của mức 4. Trong đó:
 - + 5 file text ứng với 5 file input của 5 testcase.
 - + File *Graph.py* là file hỗ trợ đọc file, và hiển thị đồ thị.
 - + File *Main.py* là file thực hiện chương trình.
- **Test Result:** chứa các animation kết quả cho các testcase. Gồm 3 thư mục, mỗi thư mục ứng với mỗi **Level** (1 và 2; 3; 4).

2. Hướng dẫn sử dụng:

- Đầu tiên, chương trình xin đường dẫn tới file input của người dùng. Nếu file input không có lỗi, chương trình tiếp tục xin tên file output (không có định dạng, vì chương trình chỉ hỗ trợ định dạng *.gif*).
- Nếu mọi thứ đã ổn, chương trình sẽ output ra màn hình thông tin về testcase đó, cộng với file animation với tên *{output}.gif*.

3. Một số lưu ý:

- Robot di chuyển theo 4 hướng: lên, xuống, trái và phải; không được phép đi chéo, đi ra ngoài bản đồ hoặc đi lên vật cản.
- Mỗi một bước di chuyển của robot đều tốn 1 đơn vị thời gian.
- Quy tắc vẽ cạnh của vật cản: với 2 điểm cạnh nhau của vật cản, ta biểu diễn cạnh đó dưới dạng đường thẳng $y = ax + b$ hoặc $x = ay + b$ sao cho $a \leq 1$. Sau đó, ta đi từng giá trị x hoặc y rồi tính giá trị còn lại rồi làm tròn, ta được một cặp x, y .

- Ví dụ: cạnh $(1, 1); (5; 4)$ được biểu diễn thành đường thẳng $y = \frac{3}{4}x + \frac{1}{4}$. Với $x = 1$, ta được $y = 1$, từ đó ta được điểm $(1, 1)$. Với $x = 2$, ta được $y = \frac{7}{4}$, từ đó ta được điểm $(2, 2)$ (vì $\frac{7}{4}$ làm tròn thành 2). Tương tự, ta được các điểm $(3, 3); (4, 3)$ và $(5, 4)$.
- Hàm **heuristic** được sử dụng là *khoảng cách Manhattan* từ điểm được tính đến điểm đích.
- Việc sử dụng *khoảng cách Manhattan* để ước lượng khoảng cách giúp hàm **heuristic** trở thành một **consistent heuristic**.
Giải thích: Xét 2 điểm A, B kề nhau. Khi đó khoảng cách từ A tới B là $AB = 1$. Gọi $h(A)$ là khoảng cách Manhattan từ điểm A đến điểm đích.
 - Với mọi điểm A, B kề nhau, ta có: $|h(A) - h(B)| = 1$.
 - Nếu $h(A) - h(B) = 1$, ta có:

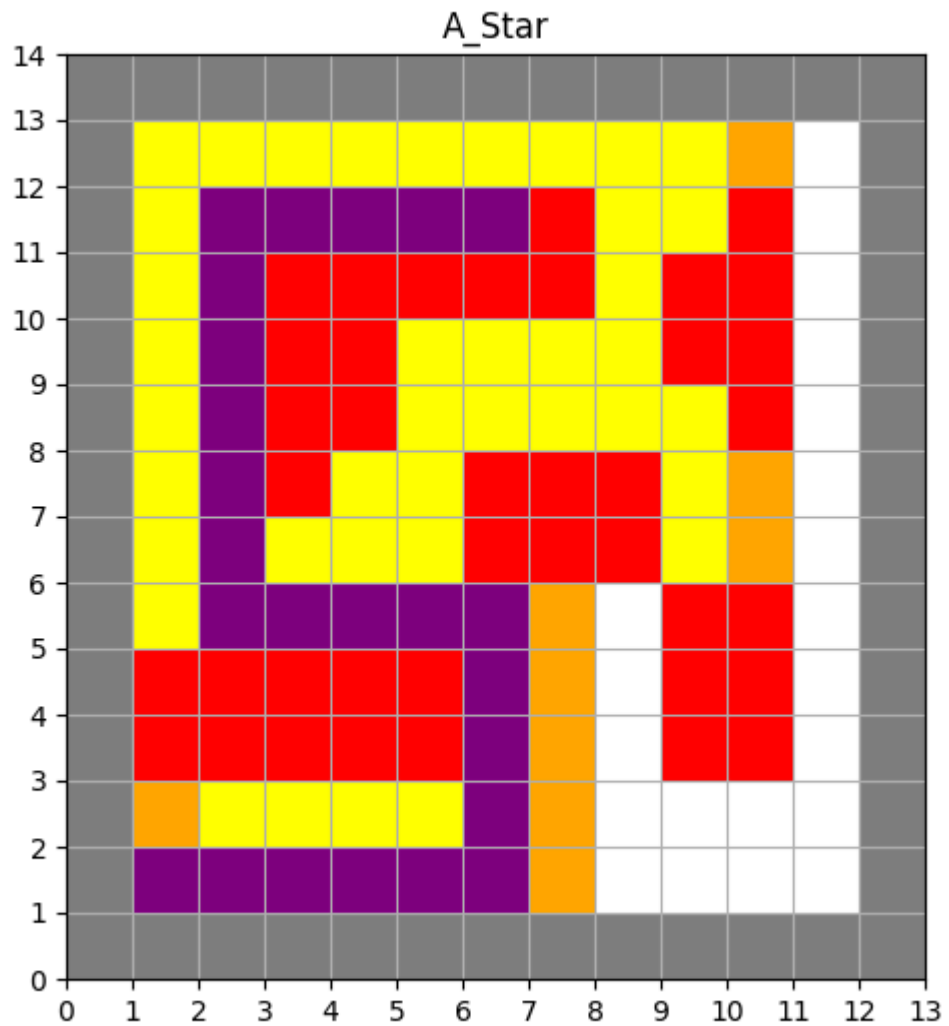
$$h(A) = 1 + h(B) \Rightarrow h(A) \leq AB + h(B).$$
 - Tương tự, nếu $h(B) - h(A) = 1$, ta có:

$$h(A) = h(B) - 1 < h(B) + 1 \Rightarrow h(A) \leq AB + h(B).$$
 → Vậy, với mọi điểm A, B kề nhau bất kì, $h(A) \leq AB + h(B)$, và do đó **heuristic** này là **consistent**.
- Vì hàm **heuristic** là **consistent**, các thuật toán duyệt đồ thị không tham lam sử dụng **heuristic này** (cụ thể trong phần thực hiện này là **A*** và **IDA***), sẽ luôn cho ra đường đi tối ưu nhất.

4. Giao diện đồ thị:

Đầu tiên, do nhóm có sử dụng thư viện **matplotlib** nên trước hết, người dùng cần tải về bằng lệnh **pip install matplotlib** qua **terminal** hay **command prompt**.

Mỗi mức có một lớp **Graph** khác nhau, tùy thuộc vào yêu cầu của đề bài, nhưng đều có các đặc điểm chung:



a. Chú thích:

Bản đồ được biểu diễn dưới dạng một lưới, với mỗi điểm biểu diễn bởi 1 màu chứa thông tin của điểm đó.

Chú thích các màu được sử dụng:

- Xám: tường, đại diện cho giới hạn của bản đồ.
- Trắng: các điểm chưa được thăm.
- Xanh dương: điểm bắt đầu.
- Xanh lá: điểm đích.
- Đỏ: viền ngoài của vật cản.
- Vàng: các điểm đã được mở.
- Cam: các điểm **frontier**, sẵn sàng được mở.
- Tím: đường đi tìm được.
- Nâu: các điểm đón (mức 3).

Ở mức 3, các điểm mở và **frontier** sẽ không được hiển thị. Màu cam lúc này là vị trí hiện tại của robot.

Ở mức 3, một điểm có thể được đi lại nhiều lần.

Ở mức 4, đường đi không được hiển thị như một con đường mà chỉ hiển thị vị trí hiện tại của robot.

b. Tính năng:

- Khởi tạo: Lớp được khởi tạo từ một tập tin đầu vào bao gồm kích thước lưới, vị trí bắt đầu và điểm đích, và các chướng ngại vật (ở mức 3 có thêm vị trí các điểm đón, mức 4 có thêm số ô di chuyển của chướng ngại vật).
- Tìm đường: Cung cấp các phương thức để tìm đường từ điểm bắt đầu đến điểm kết thúc sử dụng các thuật toán tìm đường khác nhau.
- Hiển thị kết quả: Cho phép hiển thị kết quả tìm đường thông qua **animation** và lưu thành **file GIF**.
- Kiểm tra trạng thái: Cung cấp các phương thức để kiểm tra trạng thái của một vị trí trên lưới, như **EXPLORED** hoặc **UNEXPLORED**.
- Tính toán **Heuristic**: Tính giá trị **heuristic** (ở đây là *khoảng cách Manhattan*) từ một vị trí đến điểm đích.

c. Các hàm quan trọng được sử dụng:

- **expand()**: Mở rộng các nút trên đồ thị cho đến khi đạt được điểm kết thúc.
- **set_parent()**: Thiết lập vị trí cha của một vị trí cụ thể trên đồ thị.
- **get_start()** và **get_goal()**: Trả về điểm bắt đầu và điểm đích của đồ thị.
- **is_explored()**: Kiểm tra xem một vị trí đã được khám phá chưa.
- **heuristic()**: Tính toán giá trị **heuristic** (*khoảng cách Manhattan* đến điểm đích) cho một vị trí cụ thể.
- **give_up()**: Hiển thị thông báo khi không tìm thấy đường đi.

III. BÁO CÁO CÁC MỨC HOÀN THÀNH

1. Mức 1:

a. Yêu cầu:

Cài đặt thành công 1 thuật toán để tìm đường đi từ S tới G. Báo cáo lại thuật toán và quá trình chạy thử.

Lưu ý, chạy thử trường hợp không có đường đi.

b. Thực hiện:

Ở mức 1, nhóm sử dụng thuật toán **Greedy Best-first Search** (gọi tắt là **GBFS**).

- *Khởi tạo:*

- + Người dùng nhập tên của tệp dữ liệu đầu vào.
- + Tạo đối tượng **Graph** với thuật toán là "**GBFS**".
- + Sử dụng biến boolean **found** để theo dõi việc tìm thấy đường đi.

- *Tìm đường đi:* Bắt đầu từ điểm xuất phát. Thêm điểm xuất phát vào hàng đợi ưu tiên (**priority queue**).

- Duyệt đồ thị theo độ ưu tiên: lặp lại các bước sau cho đến khi hàng đợi ưu tiên trống:

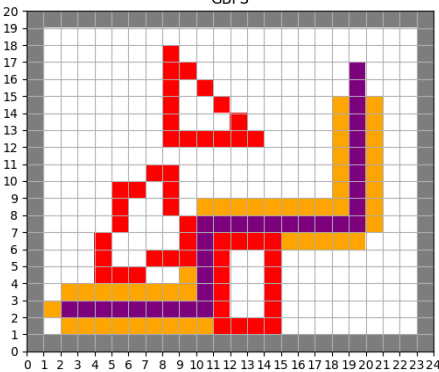
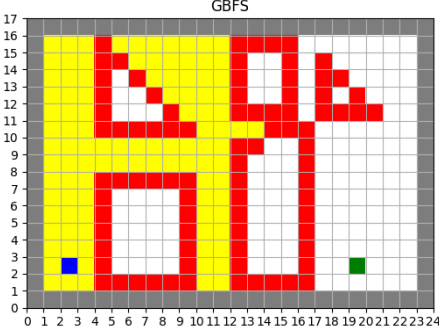
- + Lấy phần tử được ưu tiên nhất (**heuristic** nhỏ nhất) ra khỏi hàng đợi ưu tiên.
- + Nếu điểm được lấy ra là điểm đích, ta thông báo đường đi tìm được và kết thúc thuật toán.
- + Nếu không, duyệt qua tất cả các đỉnh kề của đỉnh đó và thêm chúng vào hàng đợi (nếu chưa được duyệt).
- + Đánh dấu các đỉnh đã duyệt để tránh lặp lại.

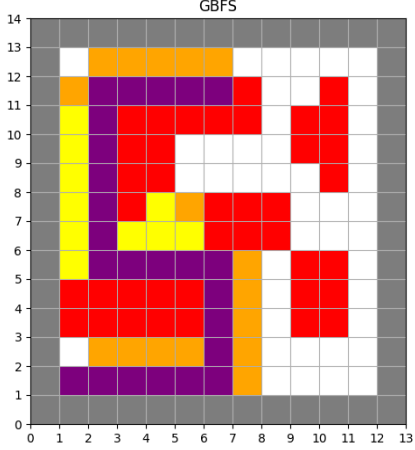
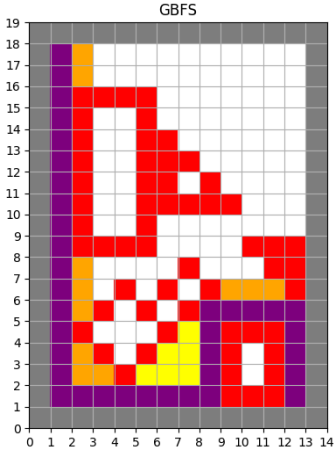
- *Kiểm tra kết quả:*

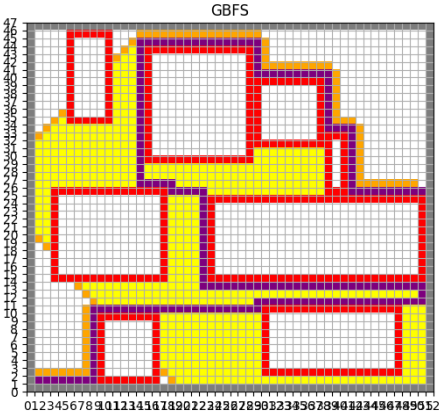
- + Nếu tìm thấy đường đi, dừng quá trình.
- + Trong trường hợp không tìm thấy đường đi, gọi hàm **give_up** để thông báo.

→ **Thuật toán GBFS** chọn nút tiếp theo dựa trên giá trị **heuristic** (ở đây là *giá trị Manhattan* đến điểm đích), tức là chọn nút được dự đoán gần đích nhất, nhưng không đảm bảo đường đi là ngắn nhất.

c. Quá trình chạy thử:

	<p>#testcase1: input1.txt</p> <p>22,18 2,2,19,16 3 4,4,5,9,8,10,9,5 8,12,8,17,13,12 11,1,11,6,14,6,14,1</p>
<p>Sau khi chạy #testcase1, ta có màn hình output:</p> <ul style="list-style-type: none"> - Path length: 31 - Visited: 79 - Opened: 31 - Frontier: 47 	
	<p>#testcase2: input2.txt</p> <p>22,15 2,2,19,2 5 4,10,9,10,4,15 4,1,9,1,9,7,4,7 12,1,16,1,16,10,12,9 17,11,20,11,17,14 12,11,15,11,15,15,12,15</p>
<p>Sau khi chạy #testcase2, ta có màn hình output:</p> <ul style="list-style-type: none"> - Không có đường đi - Opened: 104 	

 <p>A 14x14 grid showing a search space. The start cell is at (1,1) in yellow. The goal cell is at (11,12) in red. The path is highlighted in orange. The grid is labeled 'GBFS' at the top.</p>	<p>#testcase3: input3.txt</p> <p>11,12 6,11,1,1 6 1,3,1,4,5,4,5,3 9,3,9,5,10,5,10,3 6,6,6,7,8,7,8,6 3,7,3,10,5,10 6,10,7,10,7,11 9,9,9,10,10,11,10,8</p>
<p>Sau khi chạy #testcase3, ta có màn hình output:</p> <ul style="list-style-type: none"> - Path length: 23 - Visited: 50 - Opened: 44 - Frontier:16 	
 <p>A 19x14 grid showing a search space. The start cell is at (1,1) in yellow. The goal cell is at (12,17) in red. The path is highlighted in orange. The grid is labeled 'GBFS' at the top.</p>	<p>#testcase4: input4.txt</p> <p>12,17 1,17,12,1 6 4,2,2,4,4,6,6,4 7,5,6,6,7,7,8,6 9,1,9,4,11,4,11,1 10,8,12,8,12,6 6,10,6,13,9,10 2,8,2,15,5,15,5,8</p>
<p>Sau khi chạy #testcase4, ta có màn hình output:</p> <ul style="list-style-type: none"> - Path length: 35 - Visited: 53 - Opened: 41 - Frontier: 11 	

	<p>#testcase5: input5.txt</p> <p>50,45 1,1,50,25 8 9,1,9,9,16,9,16,1 30,2,47,2,47,10,30,10 3,14,17,14,17,25,3,25 23,14,50,14,50,24,23,24 15,29,28,29,28,43,15,43 5,34,10,34,10,45,5,45 29,31,29,39,37,39,37,31 38,25,40,25,40,32,38,32</p>
<p>Sau khi chạy #testcase5, ta có màn hình output:</p> <ul style="list-style-type: none"> - Path length: 183 - Visited: 788 - Opened: 706 - Frontier: 81 	

2. Mức 2:

a. Yêu cầu:

Cài đặt ít nhất 3 thuật toán khác nhau (ví dụ tìm kiếm mù, tham lam, heuristic, ...). Báo cáo nhận xét sự khác nhau khi chạy thử 3 thuật toán.

b. Thực hiện:

Ở đây, nhóm sử dụng lần lượt 3 thuật toán khác nhau: **BFS**, **A*** và **IDA***.

i. Thuật toán **BFS**:

- Bắt đầu từ điểm xuất phát. Thêm điểm xuất phát vào hàng đợi (**queue**).
- Duyệt đồ thị theo chiều rộng: Lặp lại các bước sau cho đến khi hàng đợi trống:
 - + Lấy điểm đầu tiên ra khỏi hàng đợi.
 - + Nếu điểm được lấy ra là điểm đích, ta thông báo đường đi tìm được và kết thúc thuật toán.

- + Nếu không, duyệt qua tất cả các đỉnh kề của đỉnh đó và thêm chúng vào hàng đợi (nếu chưa được duyệt).
 - + Đánh dấu các đỉnh đã duyệt để tránh lặp lại.
- **BFS** giúp tìm kiếm đường đi ngắn nhất từ điểm đích đến tất cả các đỉnh khác trong đồ thị.

ii. Thuật toán A^* :

- **Hàm Heuristic (h):** là *khoảng cách Manhattan* tới điểm đích. Như đã trình bày ở trên, đây là một **consistent heuristic**, do đó, thuật toán chắc chắn sẽ tìm được đường đi tối ưu (nếu tồn tại).
- **Giá trị $f(x)$:** Để xác định độ ưu tiên của một điểm, ta sử dụng giá trị $f(x)$. Trong đó:
 - + $g(x)$ là khoảng cách từ điểm xuất phát đến đỉnh hiện tại x .
 - + $h(x)$ là giá trị **heuristic** từ đỉnh hiện tại x đến đích.
- **Thuật toán A^* :**
 - + Bắt đầu từ đỉnh xuất phát, duyệt qua các đỉnh và xây dựng cây tìm kiếm.
 - + Duyệt đồ thị theo độ ưu tiên: lặp lại các bước sau cho đến khi hàng đợi ưu tiên trống:
 - So sánh giá trị $f(x) = g(x) + h(x)$ của các điểm đã được mở, chọn đỉnh có thứ tự ưu tiên cao hơn (cụ thể là giá trị $f(x)$ nhỏ nhất) rồi lấy ra khỏi hàng đợi ưu tiên (**priority queue**).
 - Nếu điểm được lấy ra là điểm đích, ta thông báo đường đi tìm được và kết thúc thuật toán.
 - Nếu không, duyệt qua tất cả các đỉnh kề của đỉnh đó và thêm chúng vào hàng đợi (nếu chưa được duyệt).
 - Đánh dấu các đỉnh đã duyệt để tránh lặp lại.
 - + Đường đi ngắn nhất từ điểm xuất phát đến đỉnh đích đã được tìm thấy (nếu tồn tại). Nếu không, gọi hàm **give_up** để thông báo.

iii. Thuật toán IDA^* :

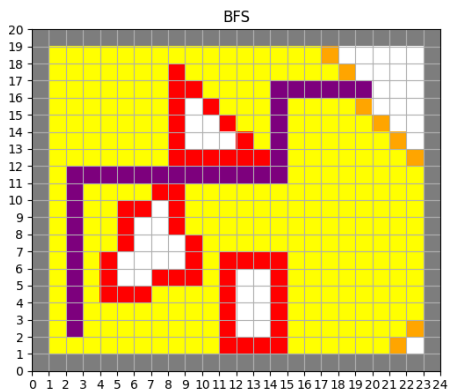
- Là một biến thể của thuật toán A^* được sử dụng để tìm đường đi ngắn nhất trong đồ thị.

- Xác định độ ưu tiên của đỉnh: tương tự như thuật toán A^* , độ ưu tiên của một đỉnh được xác định bằng giá trị $f(x) = g(x) + h(x)$, trong đó:
 - + $g(x)$ là khoảng cách từ điểm xuất phát đến đỉnh hiện tại x .
 - + $h(x)$ là giá trị **heuristic** từ đỉnh hiện tại x đến đỉnh đích.
- Giống với thuật toán A^* , hàm **heuristic** được sử dụng là *khoảng cách Manhattan* tới điểm đích. Điều này giúp thuật toán luôn tìm được đường đi tối ưu (nếu tồn tại).
- Bắt đầu từ điểm xuất phát. Thêm điểm xuất phát vào hàng đợi ưu tiên (**priority queue**) theo f .
- **Duyệt đồ thị theo độ sâu, duyệt đến khi hàng đợi ưu tiên trống:**
 - + Lấy phần tử được ưu tiên nhất (giá trị f nhỏ nhất) ra khỏi hàng đợi ưu tiên.
 - + Sử dụng thuật toán **GBFS** (Duyệt tham lam theo **heuristic**), bắt đầu tại điểm đó. Điểm khác biệt là khi gặp một điểm có f lớn hơn ngưỡng quy định, ta thêm điểm đó vào hàng đợi ưu tiên.
 - + Ngưỡng bắt đầu từ ước lượng chi phí tại trạng thái ban đầu (*khoảng cách Manhattan* từ điểm đầu đến điểm đích) và tăng dần sau mỗi lần lặp.
- **Cập nhật ngưỡng:** Tại mỗi lần lặp, ngưỡng sử dụng cho lần lặp tiếp theo là giá trị f tối thiểu của tất cả các trạng thái vượt quá ngưỡng hiện tại.

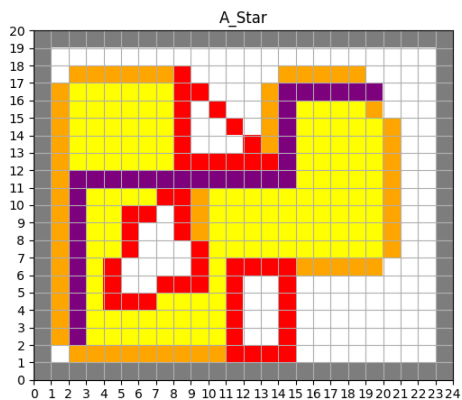
iv. Nhận xét sự khác nhau giữa 3 thuật toán:

Các bộ test case: Giống mức 1

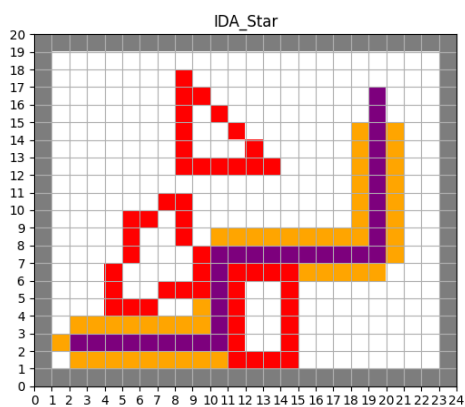
#testcase1: input1.txt



- Path length: 31
- Visited: 302
- Opened: 293
- Frontier: 8

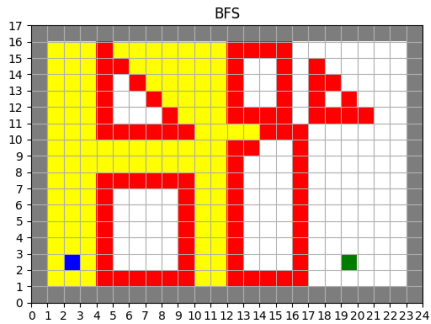


- Path length: 31
- Visited: 217
- Opened: 161
- Frontier: 56

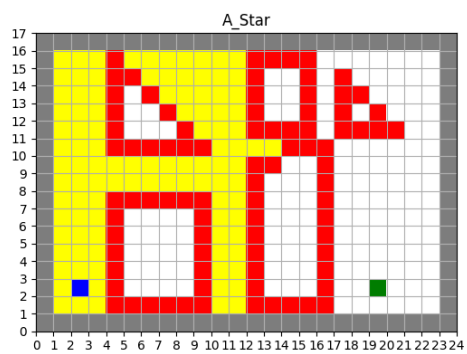


- Path length: 31
- Visited: 79
- Opened: 31
- Frontier: 47

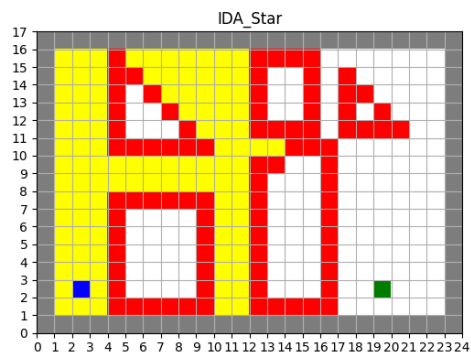
#testcase2: input2.txt



- Không có đường đi
- Opened: 104

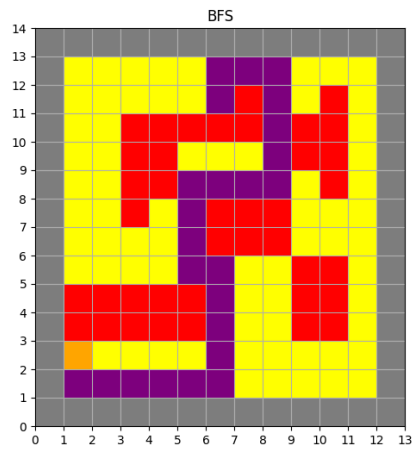


- Không có đường đi
- Opened: 104

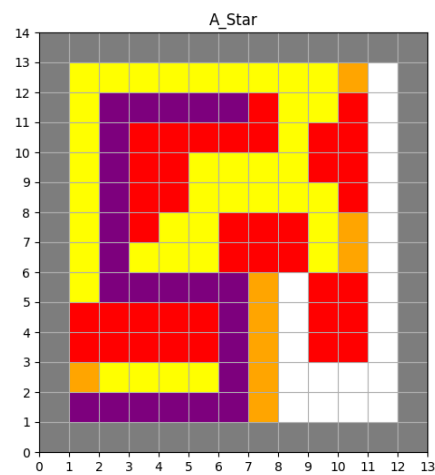


- Không có đường đi
- Opened: 104

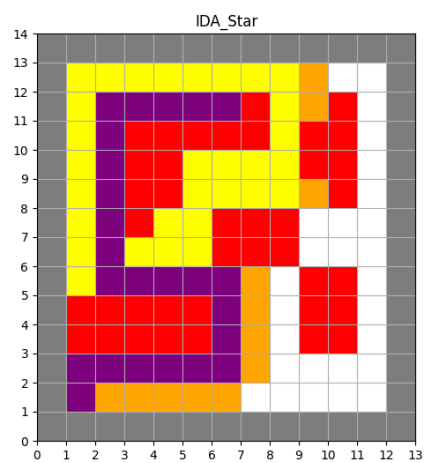
#testcase3: input3.txt



- Path length: 23
- Visited: 93
- Opened: 91
- Frontier: 1

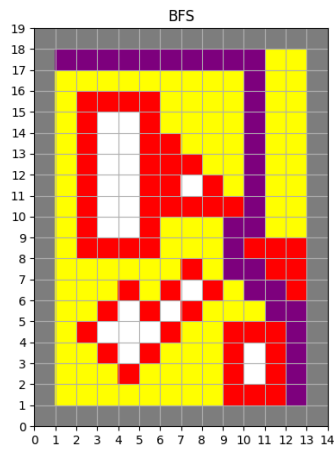


- Path length: 23
- Visited: 72
- Opened: 63
- Frontier: 9

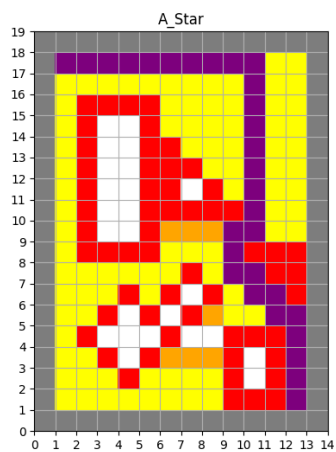


- Path length: 23
- Visited: 66
- Opened: 53
- Frontier: 12

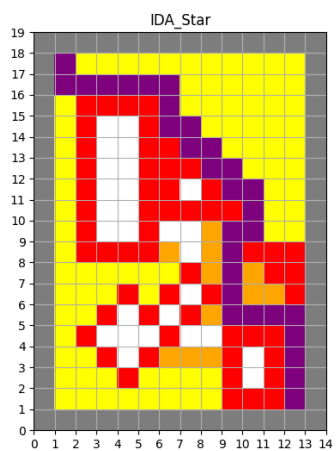
#testcase4: input4.txt



- Path length: 29
- Visited: 125
- Opened: 124
- Frontier: 0

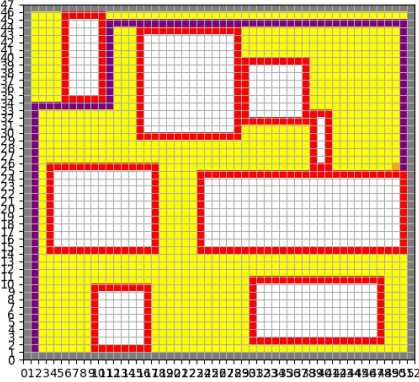
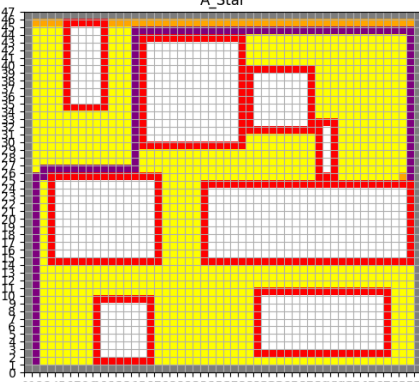
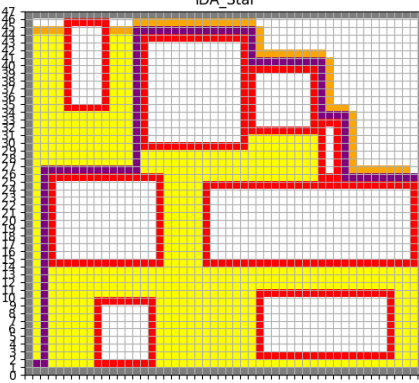


- Path length: 29
- Visited: 123
- Opened: 116
- Frontier: 7



- Path length: 29
- Visited: 120
- Opened: 108
- Frontier: 11

#testcase5: input5.txt

<p>BFS</p> 	<ul style="list-style-type: none">- Path length: 111- Visited: 1141- Opened: 1139- Frontier: 1
<p>A_Star</p> 	<ul style="list-style-type: none">- Path length: 111- Visited: 1141- Opened: 1096- Frontier: 45
<p>IDA_Star</p> 	<ul style="list-style-type: none">- Path length: 111- Visited: 909- Opened: 849- Frontier: 59

Nhận xét chung:

- **Thuật toán GBFS** luôn có số đỉnh mở nhỏ hơn 3 thuật toán còn lại, tuy nhiên đường đi tìm được lại thường không tối ưu. Đó là vì **GBFS** là thuật toán tìm kiếm tham lam, và nó chỉ chọn những điểm theo **heuristic** mà không quan tâm đến khoảng cách thật sự.
- **Thuật toán BFS** luôn có số đỉnh được mở (opened) lớn hơn so với hai thuật toán còn lại là **A*** và **IDA***, vì **BFS** là thuật toán tìm mù (*uninformed search*) trong khi 2 thuật còn lại là thuật toán tìm có thông tin (*informed search*).
- **Thuật toán A*** thường có ít **frontier** hơn thuật toán **IDA*** (vì bản chất của **IDA*** là **GBFS**), trong khi **A*** không có cài đặt tham lam và do đó phải mở tất cả các điểm có giá trị **f** bằng nhau trước khi đến giá trị **f** tiếp theo.

3. Mức 3:

a. Yêu cầu:

Trên bản đồ sẽ xuất hiện thêm một số điểm khác được gọi là điểm đón. Xuất phát từ S, sau đó đi đón tất cả các điểm này rồi đến trạng thái G. Thứ tự các điểm đón không quan trọng. Mục tiêu là tìm ra cách để tổng đường đi là nhỏ nhất. Báo cáo thuật toán đã áp dụng và quá trình chạy thử.

b. Thực hiện:

Ở mức 3, nhóm sử dụng **thuật toán A***. Thuật toán này tương tự như đã làm ở mức 2, nhưng được thêm vào các điểm đón mà robot phải đi qua trên đường tới đỉnh đích.

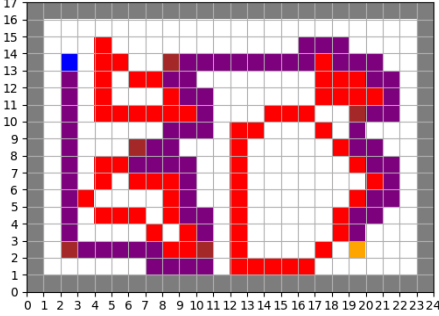
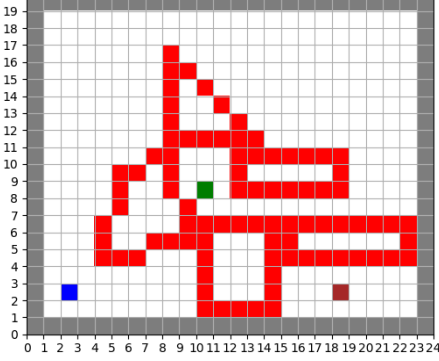
- Lớp **Graph** được cài đặt như mức 1 và mức 2, nhưng trong phần khởi tạo có thêm biến **stops[]** là một **list** gồm các điểm đón.
- **Hàm Heuristic (h)**: là *khoảng cách Manhattan* tới điểm đích. Như đã trình bày ở trên, đây là một **consistent heuristic**, do đó, thuật toán chắc chắn sẽ tìm được đường đi tối ưu (nếu tồn tại).
- **Giá trị f(x)**: Để xác định độ ưu tiên của một điểm, ta sử dụng giá trị **f**. Trong đó:
 - + **g(x)** là khoảng cách từ điểm xuất phát đến đỉnh hiện tại x.
 - + **h(x)** là giá trị **heuristic** từ đỉnh hiện tại x đến đỉnh đích.

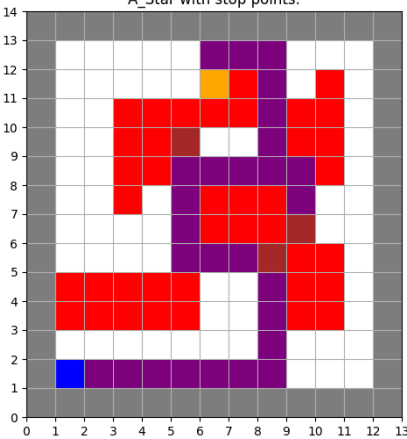
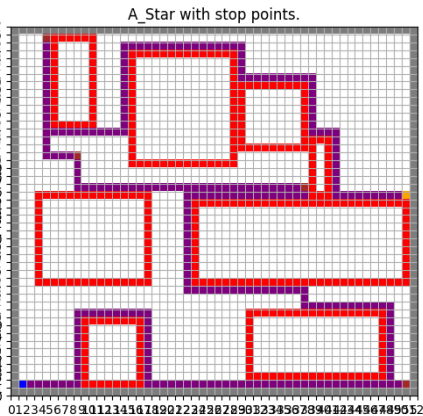
- **Ý tưởng thuật toán:**

- + Bắt đầu từ đỉnh xuất phát, duyệt qua các đỉnh và xây dựng cây tìm kiếm.
- + Tạo một mảng gồm hoán vị các điểm đón (dùng thư viện **itertools** có sẵn trong Python).
- + Dùng **thuật toán A*** (như đã mô tả ở mức 2) để tìm đường đi đến từng cặp đỉnh kề nhau theo thứ tự trong hoán vị.
- + Sau khi duyệt qua tất cả hoán vị, trả về đường đi ngắn nhất. Nếu không tồn tại đường đi, gọi hàm **give_up()** để thông báo.

c. Quá trình chạy thử:

<p>A_Star with stop points.</p>	<p>#testcase1: input1.txt</p> <p>22,18</p> <p>2,2,19,16,10,2,17,12,3,17</p> <p>3</p> <p>4,4,5,9,8,10,9,5</p> <p>8,12,8,17,13,12</p> <p>11,1,11,6,14,6,14,1</p>
<p>Sau khi chạy #testcase1, ta có màn hình output:</p> <ul style="list-style-type: none"> - Found path: Start $\rightarrow p_1(10, 2) \rightarrow p_3(3, 17) \rightarrow p_2(17, 2) \rightarrow$ Goal - Path length: 57 	

	<p>#testcase2: input2.txt</p> <p>22,15 2,13,19,2,2,2,10,2,8,13,19,10,6,8 4 4,10,4,14,9,10 3,5,4,7,8,6,9,2 12,1,12,9,16,10,20,6,16,1 17,11,17,13,20,11</p>
<p>Sau khi chạy #testcase2, ta có màn hình output:</p> <ul style="list-style-type: none"> - Found path: Start $\rightarrow p_1(2, 2) \rightarrow p_2(10, 2) \rightarrow p_5(6, 8) \rightarrow p_3(8, 13) \rightarrow p_4(19,10) \rightarrow$ Goal - Path length: 74 	
	<p>#testcase3: input3.txt</p> <p>22,18 2,2,10,8,18,2 5 4,4,5,9,8,10,9,5 8,11,8,16,13,11 10,1,10,6,14,6,14,1 12,8,12,10,18,10,18,8 15,4,15,6,22,6,22,4</p>
<p>Sau khi chạy #testcase3: Không tìm thấy đường đi.</p>	

 <p>A Star with stop points.</p>	<p>#testcase4: input4.txt</p> <p>11,12 1,1,6,11,8,5,9,6,5,9 6 1,3,1,4,5,4,5,3 9,3,9,5,10,5,10,3 6,6,6,7,8,7,8,6 3,7,3,10,5,10 6,10,7,10,7,11 9,9,9,10,10,11,10,8</p>
<p>Sau khi chạy #testcase4, ta có màn hình output:</p> <ul style="list-style-type: none"> - Found path: Start $\rightarrow p_1(8, 5) \rightarrow p_3(5, 9) \rightarrow p_2(9, 6) \rightarrow$ Goal - Path length: 35 	
 <p>A Star with stop points.</p>	<p>#testcase5: input5.txt</p> <p>50,45 1,1,50,25,4,45,50,1,37,26,8,30 8 9,1,9,9,16,9,16,1 30,2,47,2,47,10,30,10 3,14,17,14,17,25,3,25 23,14,50,14,50,24,23,24 15,29,28,29,28,43,15,43 5,34,10,34,10,45,5,45 29,31,29,39,37,39,37,31 38,25,40,25,40,32,38,32</p>
<p>Sau khi chạy #testcase5, ta có màn hình output:</p> <ul style="list-style-type: none"> - Found path: Start $\rightarrow p_2(50, 1) \rightarrow p_3(37, 26) \rightarrow p_4(8, 30) \rightarrow p_1(4, 45) \rightarrow$ Goal - Path length: 275 	

4. Mức 4:

a. Yêu cầu:

Các hình đa giác có thể di động được với tốc độ h tọa độ/s.

Cách thức di động có thể ở mức đơn giản nhất là tới lui một khoảng nhỏ để đảm bảo không đè lên đa giác khác. Chạy ít nhất 1 thuật toán trên đó. Quay video và đính kèm trực tiếp/link vào báo cáo.

b. Chú ý:

- Để biểu diễn sự di chuyển của các vật cản, ta thêm N cặp số vào cuối file, mỗi cặp số đại diện cho sự di chuyển của vật cản tương ứng.
- Trong mỗi cặp số, số đầu tiên đại diện cho số ô vật cản di chuyển về phía bên phải (số âm thể hiện việc vật cản di chuyển về phía bên trái). Trong khi số thứ hai đại diện cho số ô vật cản di chuyển lên trên (số âm thể hiện việc vật cản di chuyển xuống dưới).
- Vật cản di chuyển theo quy tắc như quy tắc vẽ cạnh cho vật cản. Mỗi bước, vật cản di chuyển theo như được quy định trong file input, sau đó về lại vị trí ban đầu, rồi lặp lại.

c. Thực hiện:

- Ta tạo bảng các ô nằm trong đường đi của vật cản. Gọi là **movement matrix**.
- Tạo 2 lưới, một lưới chứa vị trí các vật cản ở thời gian chẵn, lưới kia chứa vị trí các vật cản ở thời gian lẻ.
- Tạo mảng các vị trí cần mở (**frontier**). Ban đầu mảng gồm một phần tử là điểm xuất phát.
- Tạo mảng các vị trí nằm trong **movement matrix**, gọi là **discard**.
- Tạo biến thời gian hiện tại **t**, khởi tạo bằng 0.
- **Duyệt theo thời gian, duyệt cho đến khi còn phần tử trong **frontier** và **discard**:**
 - + Tạo mảng **new frontier** là mảng các **frontier** mở thêm được trong lần chạy này.
 - + Lấy tất cả các phần tử trong **discard**, rồi mở các điểm đó. Chỉ lấy các điểm thỏa mãn (không đụng tường) không thuộc **movement matrix**. Cho các điểm đó vào **new frontier**. Sau đó ta clear mảng này.
 - + Sau đó, lấy các phần tử trong **frontier**, rồi mở các điểm đó. Lấy các điểm thỏa mãn (không đụng tường và vật cản tại thời gian **t + 1**).
 - + Nếu điểm đó nằm thuộc **movement matrix**, ta cho vào mảng **discard**. Nếu không, ta cho nó vào mảng **frontier**.

- + Cuối cùng, ta cập nhật mảng **frontier** bằng **new frontier**, và cộng biến thời gian **t** bởi 1.
- + Nếu trong khi mở ta tìm được điểm đích thì ta thông báo đường đi và kết thúc chương trình. Còn nếu ta duyệt hết tất cả các **frontier** và không tìm được đường đi, ta thông báo là không có đường đi tồn tại.