

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN



Thành viên nhóm

20120606	Võ Tú Trình
20120113	Lê Nguyên Khang
20120611	Lâm Nhựt Trường

ĐỒ ÁN 1

CÁC THUẬT TOÁN

TÌM KIẾM

Giáo viên hướng dẫn:

Lê Hoài Bắc

Nguyễn Duy Khánh

Nguyễn Ngọc Băng Tâm

Học phần: Cơ sở trí tuệ nhân tạo

THÀNH PHỐ HỒ CHÍ MINH – 10/2022

MỤC LỤC

Contents

MỤC LỤC	2
1. Thông tin thành viên	3
2. Mức độ hoàn thành đồ án	3
3. Mô tả cách cài đặt các thuật toán	4
3.1. Depth first search	4
3.2. Breath first search	4
3.3. Uniform cost search	5
3.4 Greedy best first search	6
3.5 A star search	7
3.6 Thuật toán điểm thưởng	9
Chạy chương trình 3 bản đồ có điểm thưởng	16
4. Các bản đồ không có điểm thưởng	20
Bản đồ 1	20
Bản đồ 2	25
Bản đồ 3	29
Bản đồ 4	33
Bản đồ 5	34
5. Kịch bản nâng cấp	36
6. Thông tin bài nộp:	40
TÀI LIỆU THAM KHẢO	41

1. Thông tin thành viên

Võ Tú Trình	20120606	20120606@student.hcmus.edu.vn
Lê Nguyên Khang	20120113	20120113@student.hcmus.edu.vn
Lâm Nhựt Trường	20120611	20120611@student.hcmus.edu.vn

2. Mức độ hoàn thành đồ án

	Công việc	Mức độ hoàn thành
Mức 1a	Thiết kế đủ 5 bản đồ và hoàn thành 3 thuật toán tìm kiếm không có thông tin (bao gồm cài đặt thuật toán, báo cáo, hình vẽ minh họa)	100%
Mức 1b	Hoàn thành tiếp 2 thuật toán tìm kiếm có thông tin.	100%
Mức 2a	Đề xuất thuật toán và các hàm heuristic phù hợp để giải quyết, thiết kế đủ 3 bản đồ và mô phỏng cách thuật toán chạy trên các bản đồ	100%
Mức 2b	Cài đặt thành công thuật toán(đảm bảo kết quả giống với mô phỏng ở mức 2a)	100%
Điểm cộng	Bản đồ có các cánh cửa để dịch chuyển bất kỳ từ điểm (i, j) sang điểm (i', j') (giống teleportation - dịch chuyển tức thời). Các bạn tìm lời giải tối ưu nhất có thể cho kịch bản đó, sau đó tự thiết kế bản đồ (tối thiểu 3 bản đồ) và nhận xét.	100%

3. Mô tả cách cài đặt các thuật toán

3.1. Depth first search

Ý tưởng

Dùng thuật toán Depth-first Search - DFS (tìm kiếm theo chiều sâu) như một cách duyệt qua các đỉnh của đồ thị cho đến khi gặp được lối ra khỏi mê cung thì dừng hoặc duyệt qua hết các ô được phép đi từ “S” của mê cung thì dừng.

Cách cài đặt

- Thay vì duyệt trên một danh sách kề các đỉnh thì trong bài này duyệt qua ma trận, các cạnh kề là 4 hướng được phép đi và tất nhiên thỏa điều kiện không đi lên ô “x” và không được vượt các biên.
- Chọn đỉnh bắt đầu duyệt là ô “S”.
- Dùng vòng for qua bốn hướng đi tại ô đang xét, kiểm tra xem ô nào chưa đi và là ô được phép đi, không vượt các biên thì tiến hành DFS cho ô đó.
- Trong quá trình DFS không quên lưu lại đường đi để sau này dùng cho việc vẽ hình.
- Liên tục làm các bước trên cho đến khi gặp đích đến là “S” thì dừng lại .

Đánh giá thuật toán

- Tính đầy đủ : có nếu không gian hữu hạn
- Tính tối ưu: không (Tốt nhất khi ô “S” nằm trên lần đào sâu đầu tiên).
- Độ phức tạp thời gian: $O(b^m)$ và tệ nếu m lớn hơn nhiều so với b
- Độ phức tạp không gian: $O(bm)$ kích thước không gian tuyến tính!

3.2. Breath first search

Ý tưởng

Dùng thuật toán Breadth-first Search (tìm kiếm theo chiều rộng - BFS) loan ra cho đến khi gặp được lối ra khỏi mê cung thì dừng hoặc duyệt qua hết các ô được phép đi từ “S” của mê cung thì dừng.

Cách cài đặt

- Thay vì BFS trên một danh sách kề các đỉnh thì trong bài này BFS qua ma trận, các cạnh kề là 4 hướng được phép đi và tất nhiên thỏa điều kiện không đi lên ô “x” và không được vượt các biên.
- Chọn đỉnh bắt đầu duyệt là ô “S” push tọa độ vào queue và tiến hành BFS.
- Khi nào queue còn khác rỗng lấy đỉnh từ queue ra và xét đỉnh đó.
- Dùng vòng for qua bốn hướng đi tại ô đang xét, kiểm tra xem ô nào chưa đi và là ô được phép đi, không vượt các biên thì tiến hành push tọa độ ô đấy vào queue. Dùng trace đánh dấu đường đi ô vừa push vào.
- Tiếp tục khi queue còn khác rỗng.

Đánh giá thuật toán

- Tính đầy đủ: có (nếu b hữu hạn)
- Tính tối ưu: có vì chi phí di chuyển như nhau
- Độ phức tạp thời gian: $1 + b^1 + b^2 + b^3 + \dots + b^d = O(b^d)$
- Độ phức tạp không gian: $O(b^{(d-1)})$ cho tập mở và $O(b^d)$ cho biên

3.3.Uniform cost search

Ý tưởng

Uniform cost search là một thuật toán tìm kiếm mù(uninformed search) không sử dụng tri thức nào khác ngoài định nghĩa bài toán, thuật toán sẽ tìm kiếm trên một cấu trúc cây hoặc đồ thị có trọng số, việc tìm kiếm bắt đầu từ nút bắt đầu(nút gốc) và tiếp tục mở các nút tiếp theo với chi phí thấp nhất tính từ gốc(tương tự thuật toán Dijkstra).

Cách cài đặt

Bước 1: Tạo danh sách các **điểm đã mở**(closed list) và danh sách các **điểm biên** (open list)

Bước 2: Đưa điểm bắt đầu vào open list

Bước 3: Chọn mở điểm có chi phí(từ điểm bắt đầu đến nó) thấp nhất trong open list, và đưa nó vào closed list

Bước 4: Kiểm tra điểm vừa mở có phải là điểm exit

- Nếu điểm vừa mở là điểm exit thì dừng và trả về đường đi, đường đi được lấy dựa trên tập cha (parent).

Bước 5: Kiểm tra các successor của điểm vừa mở (**curPoint**):

- Nếu điểm này đã được mở, nằm trong **close** thì xét đến các điểm kế tiếp
- Trong trường hợp điểm đó chưa nằm trong **open**, thì sẽ đưa nó vào với điểm cha là **curPoint** và $g(n) = g(n-1) + 1$.
- Trường hợp nó đã nằm trong **open**, xem xét chi phí của nó hiện tại với chi phí trước, nếu nhỏ hơn sẽ cập nhật lại chi phí và điểm cha, nếu không thì xét đến các successor tiếp theo.

Bước 6: Quay lại bước 3 đến khi nào tìm được đường đi hoặc tập mở là rỗng.

Đánh giá thuật toán

- Tính đầy đủ: có
- Tính tối ưu: có(vì nó luôn chọn các nút với chi phí bé nhất)
- Độ phức tạp theo thời gian: $O(b^{1+[C^*/\epsilon]})$
- Độ phức tạp theo không gian: $O(b^{1+[C^*/\epsilon]})$
 - Với ϵ là chi phí di chuyển thấp nhất
 - C^* là chi phí lời giải tối ưu

Trong bài toán tìm đường đi ma trận này thì thuật toán UCS trở thành thuật toán BFS do chi phí di chuyển như nhau.

3.4 Greedy best first search

Ý tưởng

Thuật toán GBFS(greedy best first search) là một chiến lược tìm kiếm với tri thức bổ sung ngoài việc sử dụng định nghĩa bài toán(informed search). Thuật toán sẽ sử dụng 1 hàm đánh giá là hàm heuristic $h(n)$ để đánh giá chi phí di chuyển từ nút hiện tại đến nút đích. GBFS sẽ cố gắng mở các nút có ước lượng heuristic nhỏ nhất, nghĩa là các nút “có thể” gần với đích nhất.

Cách cài đặt

Bước 1: Đầu tiên ta tạo danh sách các **điểm đã mở**(closed list) và danh sách các **điểm biên** (open list)

Bước 2: Đặt điểm bắt đầu vào open list

Bước 3: Chọn ra điểm có ước lượng heuristic nhỏ nhất trong open list và mở điểm đó

Bước 4: Đặt điểm vừa mở vào closed list

Bước 5: Kiểm tra điểm vừa mở có phải là điểm exit

- Nếu là điểm exit thì dừng và sẽ trả về đường đi, đường đi được lấy dựa trên tập cha (**parent**) được xét ngược đi từ đích

Bước 6: Kiểm tra các successor của điểm vừa mở

- Nếu không nằm trong closed list, đưa vào open list, cập nhật $h(n)$ và điểm cha của nó

Bước 7: quay lại bước 3 đến khi nào tập mở rỗng hoặc đã tìm ra được đường đi đến đích

Đánh giá thuật toán

- Tính đầy đủ: có(vì nó không mở lại nút đã mở nên nó sẽ không rơi vào vòng lặp vô hạn)
- Tính tối ưu: không
- Độ phức tạp thời gian: $O(b^m)$
- Độ phức tạp không gian: $O(b^m)$
- Độ phức tạp giống với thuật toán DFS trong trường hợp tệ nhất

Hàm heuristic mà chúng em chọn là khoảng cách manhattan giữa 2 điểm

Với hai điểm $x(a, b)$ và $y(c, d)$ thì khoảng cách **manhattan** giữa chúng là:

$$d = |a - c| + |b - d|$$

3.5 A star search

Thuật toán A* là thuật toán tìm kiếm trong đồ thị, thuật toán sẽ tìm đường đi từ một nút ban đầu đến một nút đích cho trước sao cho chi phí tốt nhất và số bước duyệt là ít nhất. A* là thuật toán cải thiện hiệu năng từ thuật toán greedy best first search và thuộc lớp các thuật toán tìm kiếm với thông tin cho trước (informed search)

Ý tưởng

Khác với các thuật toán tìm kiếm mù, trong quá trình tìm kiếm A* tích hợp heuristic vào quá trình tìm kiếm để ước tính khoảng cách từ nút đó đến đích. Việc làm này giúp xác định được nút nào gần đích nhất và đưa ra sự lựa chọn cho các ô kế tiếp.

Lựa chọn Heuristic

Hàm heuristic được chọn để ước lượng khoảng cách từ điểm n đến đích là hàm tính khoảng cách :

- Khoảng cách **Euclid** (L^2): $h_2(n) = \sqrt{(x_n - x_{Goal})^2 + (y_n - y_{Goal})^2}$
- Khoảng cách **Manhattan** (L^1): $h_1(n) = |x_n - x_{Goal}| + |y_n - y_{Goal}|$
- Khoảng cách **Chebyshev** (L^∞): $h_\infty(n) = \max(|x_n - x_{Goal}|, |y_n - y_{Goal}|)$

Hàm được đánh giá là tối ưu trong bài toán là hàm **Manhattan**.

Mô tả thuật toán

- Thuật toán A* mở đầu bằng việc khởi tạo bằng tập các điểm mở (**Open**) và tập đóng là tập các điểm đã được mở (**visited**). Những điểm nào ở trong tập đóng thì sẽ không được xét tới nữa.
- Ở mỗi bước tập mở sẽ lấy một ô để xét đường đi. Việc chọn điểm dựa trên ước lượng chi phí đến đích: $f(n) = g(n) + h(n)$. Trong đó, $g(n)$: chi phí đường đi đến n, $h(n)$: ước tính khoảng cách đến đích (heuristic), $f(n)$: ước tính chi phí đến đích. Trong tập mở sẽ chọn điểm có ước tính chi phí là bé nhất.
- Đầu tiên đưa vào tập mở với điểm đầu tiên là điểm bắt đầu (**start**), và điểm đang được xét (**curPoint**) cũng là **start**.
- Xem xét **curPoint** có phải là đích hay không:
 - Nếu có thì dừng thuật toán và trả về đường đi, đường đi sẽ được lấy dựa trên tập cha (**parent**) được xét ngược lại từ đích.
 - Nếu **curPoint** không phải đích sẽ đi đến tìm bước đi tiếp theo bằng việc tìm các điểm lân cận (**findNearPoint**):
 - Nếu điểm lân cận nào đã được xét, có trong **visited** sẽ không được đưa vào **open** nữa.
 - Trường hợp điểm đó chưa có trong **open** thì sẽ đưa nó vào với điểm cha là **curPoint** và ước tính chi phí của nó

$$f(n) = g(n - 1) + 1 + h(n)$$
 (vì chi phí giữa 2 điểm lân cận là 1).

- Trường hợp điểm lân cận đã có trong **open** thì phải xét $f(n)$ hiện tại có nhỏ hơn $f(n)$ trước không, nếu có sẽ cập nhật lại cả điểm cha và $f(n)$, nếu không thì bỏ qua và thực hiện với điểm lân cận tiếp theo.
- Tiếp theo ta sẽ đưa **curPoint** vào **visited** và xóa khỏi **open**, tiếp tục tìm **curPoint** bằng việc tìm điểm có $f(n)$ bé nhất trong tập mở, cứ tiếp tục như vậy đến khi nào tìm được đích hoặc tập mở (**open**) rỗng (trường hợp không tìm được đường đi).

Đánh giá thuật toán

- Với chi phí di chuyển thấp nhất là ϵ , C^* là chi phí lời giải tối ưu
- Tính đầy đủ: có nếu $\epsilon > 0$ và không gian trạng thái hữu hạn
- Tính tối ưu: có nếu heuristic hợp lý và nhất quán
- Độ phức tạp thời gian: cấp số mũ
- Độ phức tạp không gian: cấp số mũ

Lưu ý: Trong bài toán sử dụng heuristic để ước lượng đường đi, chọn điểm có ước tính nhỏ nhất, tuy nhiên trong trường hợp có nhiều điểm nhỏ nhất bằng nhau thì cả thuật toán gbfs và A^* đều chọn điểm được đưa vào tập mở đầu tiên.

3.6 Thuật toán điểm thưởng

Đây là bài toán mở rộng từ bài toán tìm kiếm đường đi thông thường, thay vì tìm kiếm đường đi ngắn nhất để đi từ điểm bắt đầu đến đích, thì có thể đi qua các điểm thưởng, khi đi qua các điểm này, chi phí sẽ được trừ đi một lượng nhất định.

Ý tưởng của thuật toán:

Thông thường thì ta sẽ đi tìm con đường để đi thẳng đến điểm đích, tuy nhiên khi xuất hiện điểm thưởng thì có một câu hỏi được đặt ra là “*Liệu khi ăn điểm thưởng này có lợi hay không?*”. Nếu có lợi thì sẽ đi qua điểm thưởng, còn không sẽ đi thẳng đến đích.

Do đó, trước khi thực hiện việc tìm kiếm thì thuật toán sẽ ước tính xem đi đến điểm thưởng có lợi hay không, sau đó sẽ thực hiện việc tìm kiếm đường đi đến đích hoặc điểm thưởng bằng thuật toán A^* .

Heuristic được sử dụng:

- Heuristic ước tính khoảng cách giữa 2 điểm:

$$\text{Khoảng cách Manhattan } (L^1): h(n) = |x_n - x_{Goal}| + |y_n - y_{Goal}|$$

- Ước tính điểm thưởng có lợi:

- Dựa trên ước tính khoảng cách, ước tính khoảng cách từ điểm đầu đi thẳng đến đích (**cost_start**) và tổng khoảng cách từ điểm đầu đến điểm thưởng với từ điểm thưởng đến điểm đích (**cost_bonus**). Một điểm thưởng được cho là có lợi khi **cost_start** > **cost_bonus** + điểm thưởng (điểm thưởng là một số âm)
- Khoảng cách được ước tính thì dựa trên hàm tính khoảng cách cơ bản, tương tự như các hàm heuristic được chọn. Hàm tính khoảng cách được sử dụng trong thuật toán này là hàm khoảng cách **Manhattan**.

Mô phỏng thuật toán:

- 2 điểm thưởng:

+ Đầu tiên, từ start sẽ ước tính các điểm có lợi:

.Ước tính khoảng cách từ start đến đích
 $d(S, G) = 1 + 23 = 24$

. Ước tính tổng khoảng cách từ S đến điểm thưởng (3,9) và từ (3,9) đến G:

$$\begin{aligned} D1 &= d(S, (3,9)) + d((3,9), G) \\ &= 12 + 10 = 22 \end{aligned}$$

+ Tương tự với điểm thưởng (5,12):

$$D2 = 28$$

⇒ Chi phí khi đi qua điểm thưởng (3,5)

$$= 22 - 5 = 17 < d(S, G) \Rightarrow (3,9)$$

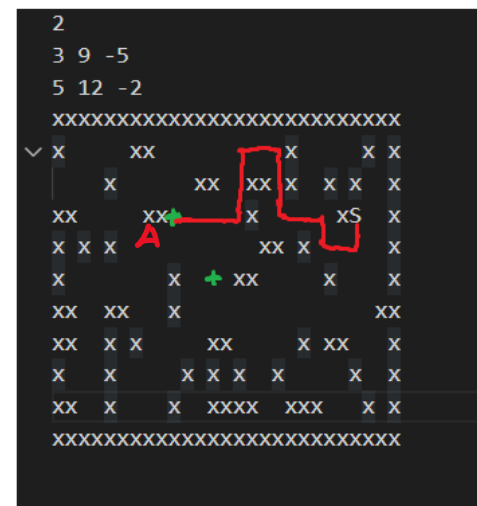
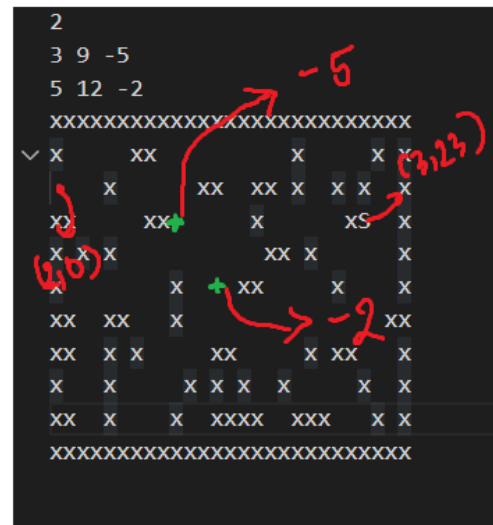
có lợi

⇒ Chi phí khi đi qua điểm thưởng (5,12)

$$= 28 - 2 = 26 > d(S, G) \Rightarrow (5,12)$$

không có lợi.

⇒ (3,9) được chọn để xem xét đường đi: ta thấy từ S đến (3,9) có đường đi và từ (3,9)



đến đích cũng có đường đi mà không cắt vào đường đi trước

⇒ (3,9) được chọn để đi đến

- Xét tiếp từ điểm $A = (3,9) \rightarrow G: d(A, G) = 10$

$$\begin{aligned} d1 &= d(A, (5,12)) + d((5,12), G) \\ &= 5 + 14 = 19 \end{aligned}$$

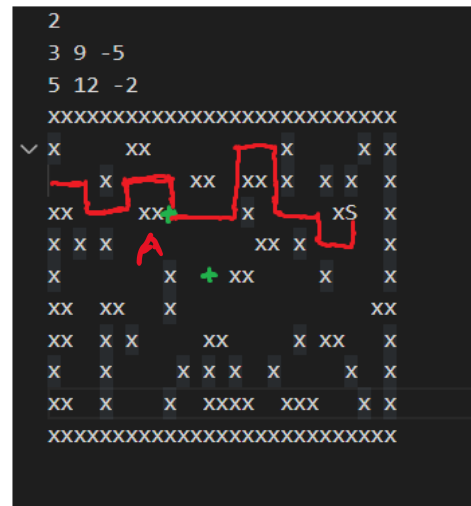
⇒ $d < d1 - 2 \Rightarrow$ điểm (5,12) không có lợi

⇒ Xét không còn điểm thưởng nào

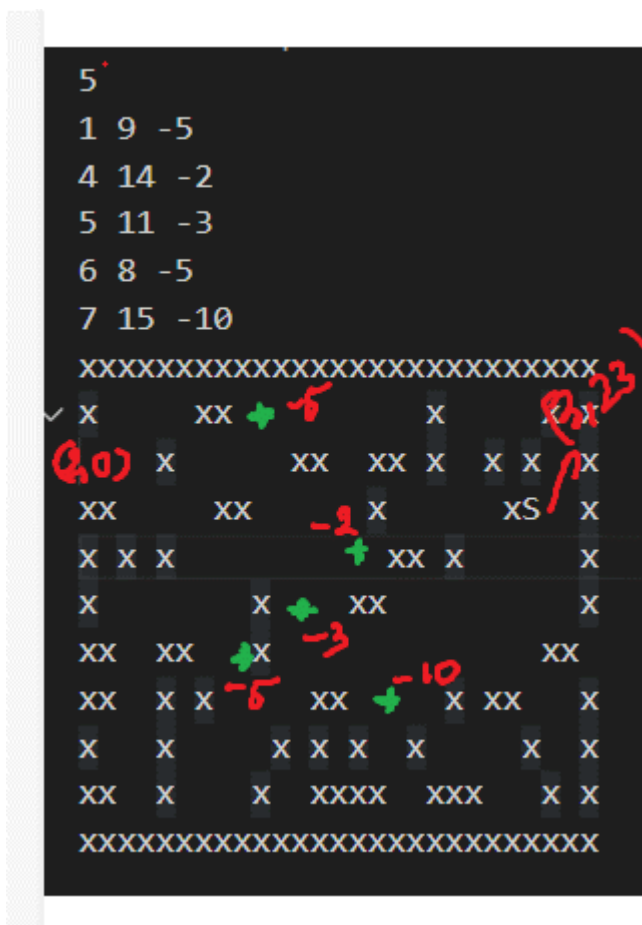
⇒ Đi thẳng từ A đến G:

⇒ Hoàn thành đường đi với chi phí là

$$10 + 17 = 27$$



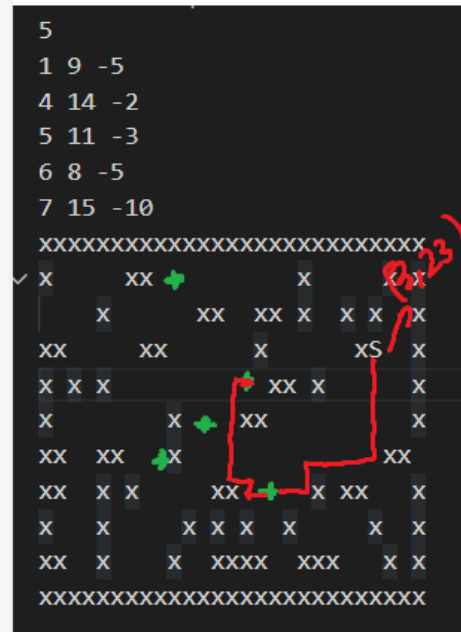
- 5 điểm thưởng



Xét từ S: $d(S, G) = 24$

Điểm thưởng	Khoảng cách ước tính $d(S, A) + d(A, G)$	Tổng chi phí	Có lợi
(1,9)	26	21	Có
(4,14)	26	24	Không
(5,11)	28	25	Không
(6,8)	20	25	Không
(7,15)	32	22	Có

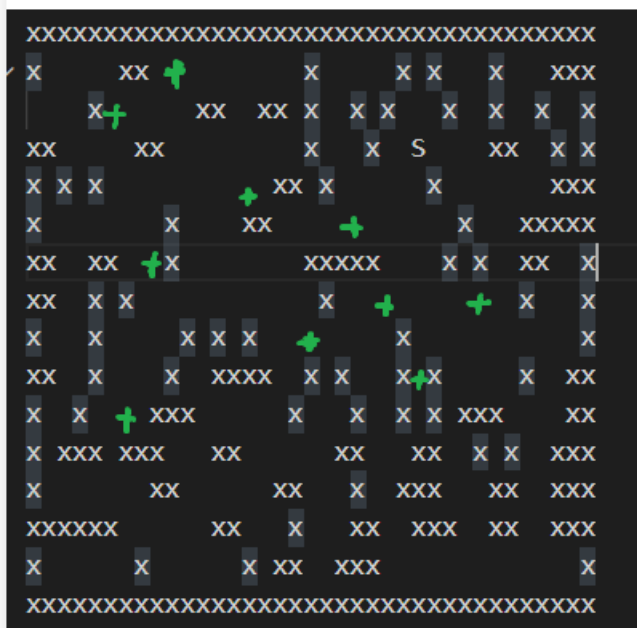
- Xét điểm (7,15) : $d((7,15), G) = 20$
 - ⇒ Các điểm thưởng còn lại có lợi là
(1,9), (4,14), (5,11), (6,8)
 - ⇒ Chọn điểm xa đích hơn: (4,14). Tồn tại đường đi
 - ⇒ Đi đến (4,14) .



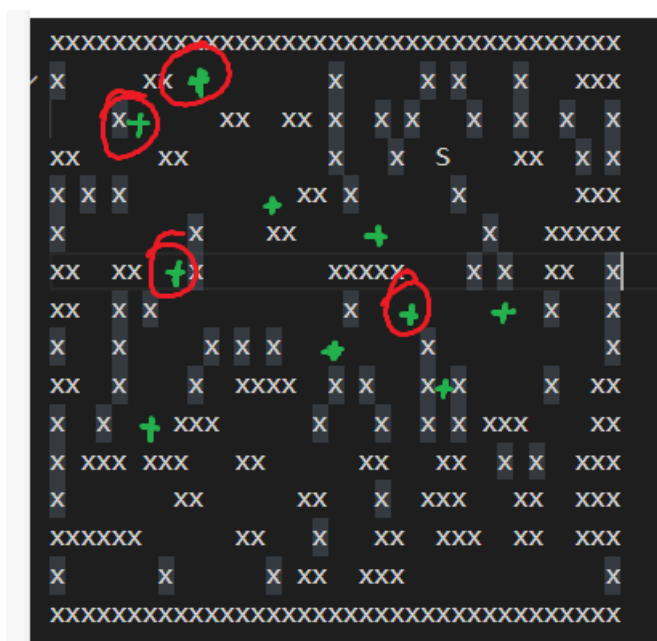
- Tương tự chọn được các điểm tiếp theo là (5,11) → (6,8) → (1,9) → G



- ⇒ Hoàn thành đường đi
- **10 điểm thưởng:**



- Từ S có các điểm có lợi là (1,9), (6,8), (7,23), (2,5)



⇒ Chọn (7,23)

- Tương tự chọn được các điểm tiếp theo và đến đích :



- ⇒ Các điểm không được chọn là những điểm được ước tính không có lợi hoặc không có đường đi phù hợp
- ⇒ Hướng di chuyển giữa 2 điểm được vẽ bằng thuật toán A*

Mô tả thuật toán:

- Đầu tiên thuật toán xem xét thử rằng có đường đi từ điểm bắt đầu (**start**) đến điểm đích (**goal**) hay không, nếu không sẽ trả về đường đi là tập rỗng, nếu có sẽ thực hiện các bước tiếp theo.
- Điểm đầu tiên được xét (**curPoint**) là điểm bắt đầu, **curPoint** là điểm xuất phát của từng chặng đường.
- Tiếp đến , thuật toán sẽ đi tìm các điểm thưởng được cho là có lợi (**set_bonus**) dựa trên việc ước tính điểm thưởng từ **curPoint** đến đích
- Trong số những điểm thưởng có lợi vừa được tìm thì điểm được chọn (**Goal_bonus**) để đi đến là điểm nằm xa điểm đích nhất (ước lượng bằng **Manhattan**). Điểm đó sẽ được lấy ra khỏi tập điểm thưởng ban đầu. Trường hợp không còn điểm để chọn thì **Goal_bonus = goal**.

- Sau khi đã xác định điểm để đi đến, thuật toán lại đi đến xác định rằng khi ăn điểm thưởng đó thì có còn đường nào để đến đích hay không. Thuật toán sẽ đi tìm 2 đường đi **path_1** là đường đi từ **curPoint** đến **Goal_bonus** bằng A* với tập đóng (**visited**) là đường đã được tìm ở các chặng trước (**path**), **path_2** là đường đi từ **Goal_bonus** đến goal với tập đóng là (**path + path1**) :
 - o Nếu **path_2** là tập rỗng tức là không có đường đi từ điểm thưởng đến đích, khi đó sẽ xét đến các điểm thưởng tiếp theo.
 - o Ngược lại thì sẽ **path_1** được chọn và được thêm vào tập đường đi, **curPoint** lúc này sẽ là điểm thưởng vừa được xét,
- Cứ thực hiện như vậy đến khi đi đến đích (tức **curPoint = goal**).

Chạy chương trình 3 bản đồ có điểm thưởng

Trường hợp 2 điểm thưởng:

- Trường hợp không có điểm thưởng thì đường đi có dạng:

```

*****
*          **  ▾◀◀◀◀◀◀◀◀*          *
EXIT◀◀  *▾◀◀◀◀◀◀◀◀*  **◀*  *  *
**◀◀◀◀◀  **          *  ◀◀◀◀◀  *★  *
*  *  *          **  *  *◀◀◀◀◀  *
*          *          **          *
**  **  *          **          **
**  *  *          **          *  **
*  *          *  *  *  *  *          *
**  *          *  ****  ****  *
*****
  
```

- Kết quả đường đi khi có 2 điểm thưởng tại (1,9) = -5 và (5,12) = -2 :

```

*****
*          **  ▾◀◀◀◀◀◀◀◀*          *
EXIT◀◀  *▾◀◀◀◀◀◀◀◀*  **◀*  *  *
**◀◀◀◀◀  **  +◀◀◀◀◀◀◀◀*  ◀◀◀◀◀  *★  *
*  *  *          *  +  *          *
*          *          **          *
**  **  *          **          **
**  *  *          **          *  **
*  *          *  *  *  *  *          *
**  *          *  ****  ****  *
*****
  
```


- Nhận xét:

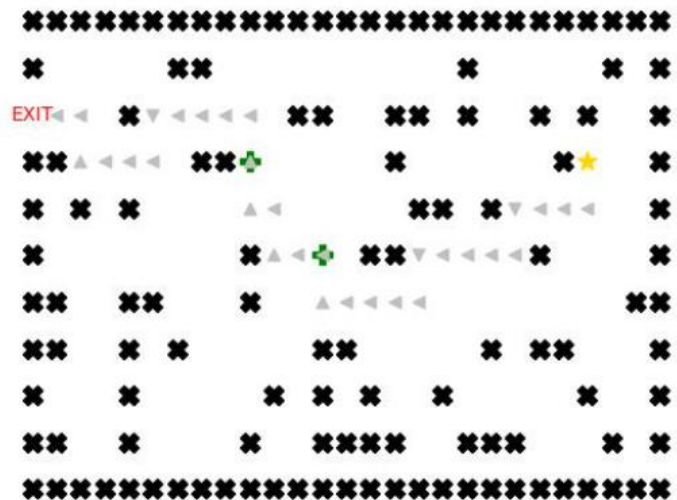
+ Trong cùng một bản đồ, trong 2 trường hợp có điểm thưởng và không có điểm thưởng là hoàn toàn khác nhau, và có thể thấy rằng đường đi đã có thể đi qua các điểm thưởng.

+ Ở bản đồ trên, có 2 điểm thưởng là (1,9) và (5,12) tuy nhiên đường đi chỉ đi qua điểm (1,9), nguyên nhân là do khi ước lượng điểm thưởng thì điểm (5,12) được cho rằng không có lợi nên không được xét đến .

- Nếu thử tăng giá trị điểm thưởng ở (5,12) lên để chắc chắn rằng có lợi :

⇒ Ta có thể thấy rằng chương trình lập tức đi qua điểm (5,12)

⇒ Thuật toán đã giải quyết được ước lượng điểm thưởng có lợi.

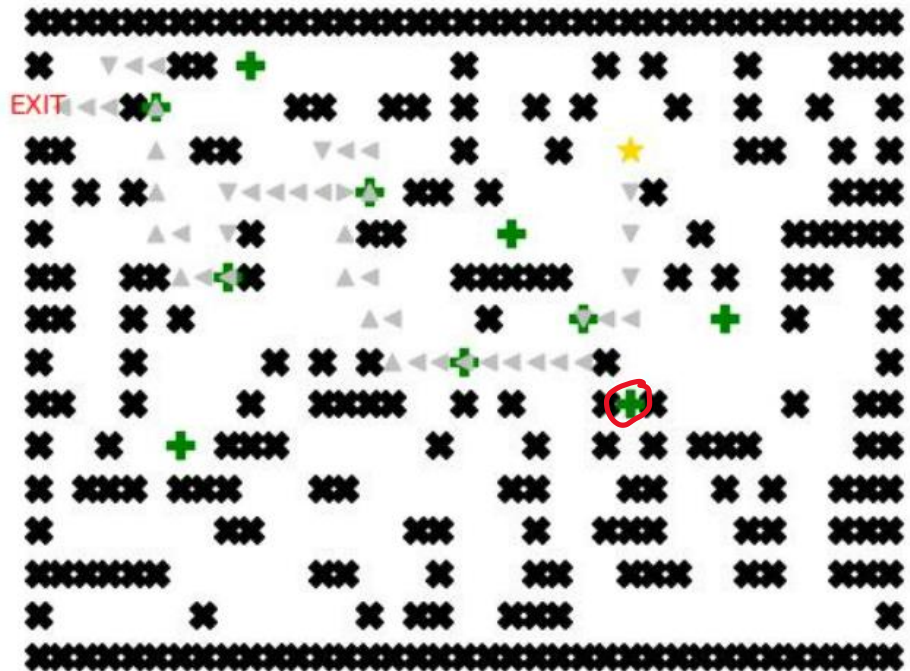


- So sánh chi phí của đường đi trường hợp có điểm thưởng và không có điểm thưởng:

Không điểm thưởng	Có điểm thưởng
<p>Chi phí là: 30</p>	<p>Chi phí là : 27</p>

Trường hợp 10 điểm thưởng:

10
1 9 -5
4 14 -2
5 20 -3
6 8 -8
9 25 -10
7 23 -15
10 6 -4
7 29 -5
8 18 -10
2 5 -4


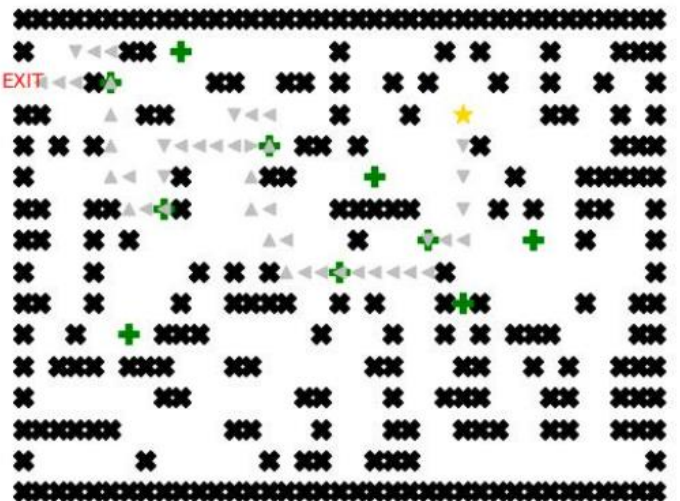
**- Nhận xét:**

+ Ở trường hợp này đường đi không đi qua hết các điểm thưởng.

+ Có những điểm chỉ có mặc dù có lợi, tuy nhiên lại không được đi đến, Lý do là vì những điểm này không có đường vào hoặc có đường vào nhưng không có đường để đi đến các điểm tiếp theo. Ví dụ như điểm (9,25) (điểm được đánh dấu màu đỏ).

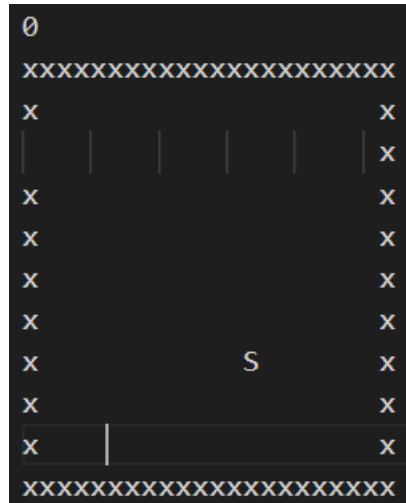
+ Từ trường hợp này ta có thể đánh giá rằng thuật toán đã giải quyết được vấn đề về ước lượng điểm có lợi, ước tính đường đi trước khi bắt đầu thực hiện việc tìm kiếm đường đi.

- So sánh chi phí của đường đi trường hợp có điểm thưởng và không có điểm thưởng:

<p>Không điểm thưởng:</p>		<p>Chi phí là 32</p>
<p>Có điểm thưởng</p>		<p>Chi phí là 7</p>

4. Các bản đồ không có điểm thưởng

Bản đồ 1



Bản đồ đầu tiên chúng em chọn là bản đồ không có chướng ngại vật, mục đích là để xem thử tự và cách mở các điểm của các thuật toán.

Depth first search

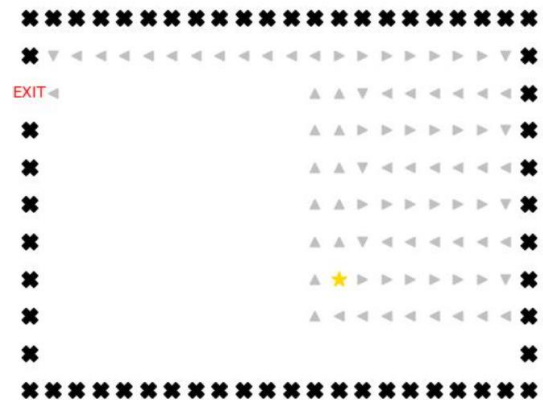
Tính đầy đủ: có

Tính tối ưu: không

Chi phí di chuyển: 84

Độ mở các nút: mở 181 nút

Thời gian chạy: 0.2966067791



Breadth first search

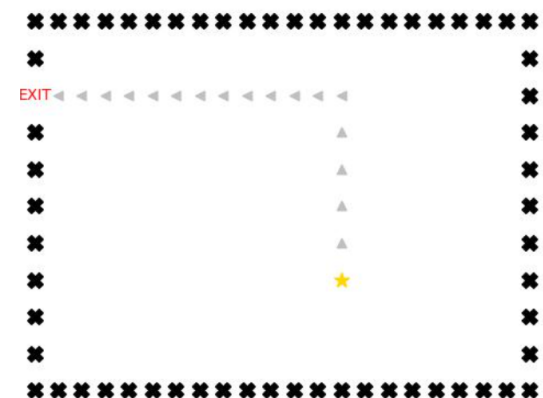
Tính đầy đủ: có

Tính tối ưu: có

Chi phí di chuyển: 18

Độ mở các nút: mở 181 nút

Thời gian chạy: 0.2530603409



Uniform cost search

Tính đầy đủ: có

Tính tối ưu: có

Chi phí di chuyển: 18

Độ mở các nút: mở 180 nút

Thời gian chạy: 0.2024087906

**Greedy best first search**

Tính đầy đủ: có

Tính tối ưu: không

Heuristic: khoảng cách Manhattan

Chi phí di chuyển: 18

Độ mở các nút: mở 19 nút

Thời gian chạy: 0.2032821178



Heuristic: khoảng cách Euclid

Chi phí di chuyển: 18

Độ mở các nút: mở 19 nút

Thời gian chạy: 0.2044551373



Heuristic: khoảng cách Chebyshev

Chi phí di chuyển: 18

Độ mở các nút: mở 19 nút

Thời gian chạy: 0.2019240856

**A star search**

Tính đầy đủ: có

Tính tối ưu: có

Heuristic: khoảng cách Manhattan

Chi phí di chuyển: 18

Độ mở các nút: mở 78 nút

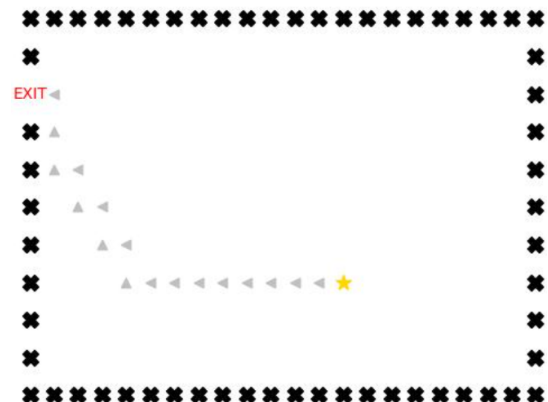
Thời gian chạy: 0.2009544373

**Heuristic: khoảng cách Euclid**

Chi phí di chuyển: 18

Độ mở các nút: mở 101 nút

Thời gian chạy: 0.2286980152

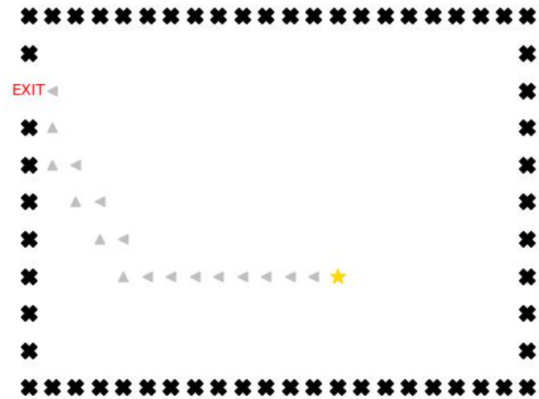


Heuristic: khoảng cách Chebyshev

Chi phí di chuyển: 18

Độ mở các nút: mở 109 nút

Thời gian chạy: 0.2004804611

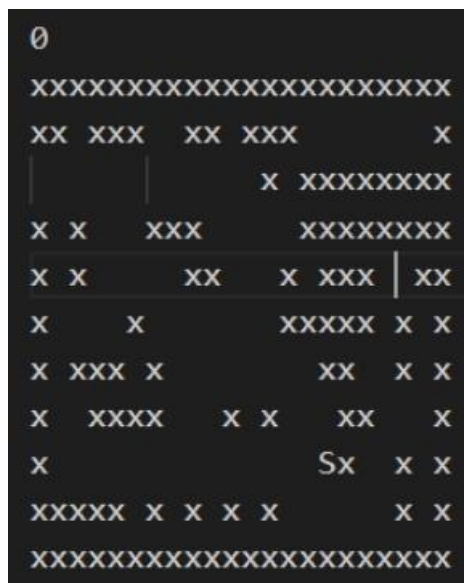


=> Trong thuật toán A* và GBFS, bằng 3 heuristic khác nhau ta có thể thấy sự khác nhau về đường đi giữa 3 heuristic. Về chi phí di chuyển, cả 3 heuristic đều cho ra cùng 1 chi phí tuy nhiên về độ mở và thời gian chạy thì Euclid và Chebyshev lớn hơn nhiều so với Manhattan. Bởi vì Manhattan có cách tính giống như là cách di chuyển của đường đi, là đi ngang và đi dọc.

=> Heuristic khoảng cách Manhattan tối ưu hơn các heuristic còn lại.

Nhận xét

- Các thuật toán đều cho đường đi ngắn nhất còn DFS thì không.
- UCS và BFS mở cùng số nút, do bản đồ có chi phí bằng nhau nên UCS sẽ trở thành BFS trong TH này.
- UCS có thời gian chạy lâu hơn BFS do UCS phải tốn thêm thời gian để duyệt tập các điểm biên.
- BFS, DFS và UCS mở nhiều nút hơn hẳn các thuật toán có thông tin
=> các thuật toán có thông tin tiết kiệm chi phí lưu trữ.
- Về độ mở, GBFS có số nút mở ít hơn và những điểm được mở là những điểm nằm trên đường đi, nguyên nhân là do gbfs có xu hướng mở những điểm gần đích hơn và trong trường hợp này, bản đồ không có vật cản nên gbfs dễ dàng đi tới đích. Còn các thuật toán khác, việc mở điểm còn phụ thuộc vào nhiều yếu tố khác nên số nút mở là nhiều hơn so với gbfs.
- GBFS chạy tốt hơn A* trong bản đồ này.

Bản đồ 2*Bản đồ ngẫu nhiên***Breadth first search**

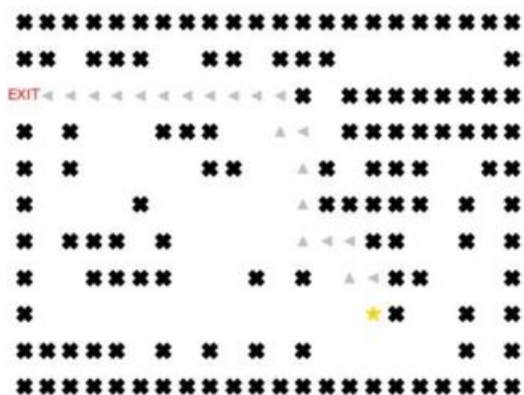
Tính đầy đủ: có

Tính tối ưu: có

Chi phí di chuyển: 21

Độ mở các nút: mở 103 nút

Thời gian chạy: 0.1979420185



Depth first search

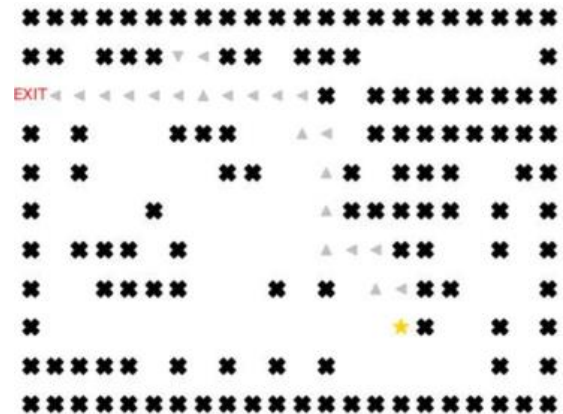
Tính đầy đủ: có

Tính tối ưu: không

Chi phí di chuyển: 23

Độ mở các nút: mở 28 nút

Thời gian chạy: 0.2053375244

**Uniform cost search**

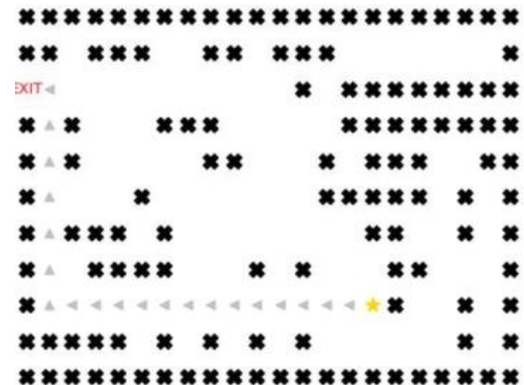
Tính đầy đủ: có

Tính tối ưu: có

Chi phí di chuyển: 21

Độ mở các nút: mở 103 nút

Thời gian chạy: 0.2054078579

**Greedy best first search**

Heuristic: Manhattan

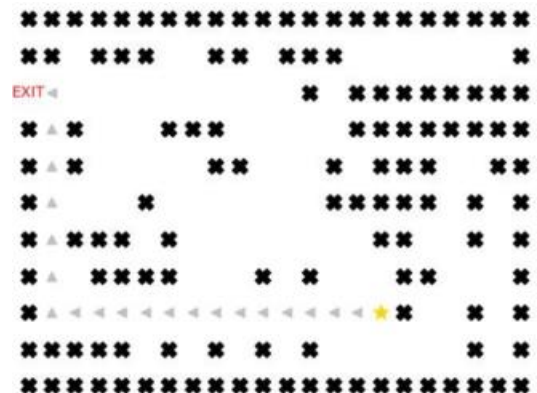
Tính đầy đủ: có

Tính tối ưu: có

Chi phí di chuyển: 21

Độ mở các nút: mở 22 nút

Thời gian chạy: 0.2029225826



Heuristic: Euclid

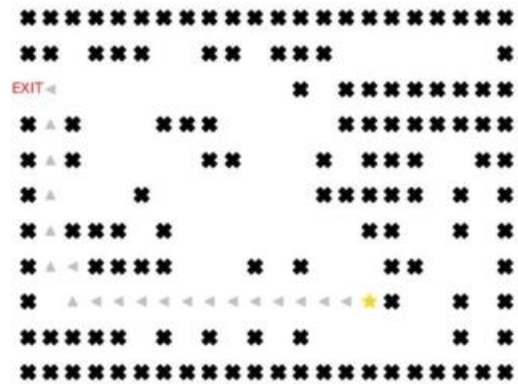
Tính đầy đủ: có

Tính tối ưu: có

Chi phí di chuyển: 21

Độ mở các nút: mở 22 nút

Thời gian chạy: 0.2003166676

**Heuristic: Chebyshev**

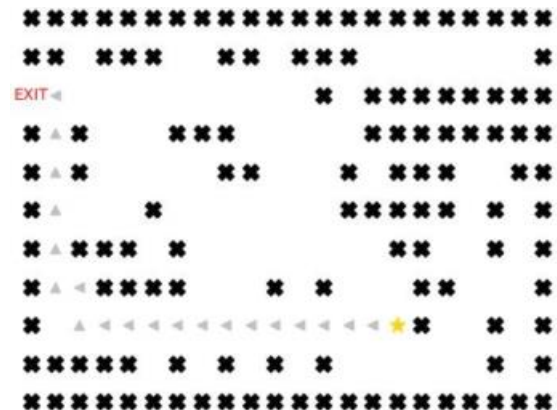
Tính đầy đủ: có

Tính tối ưu: có

Chi phí di chuyển: 21

Độ mở các nút: mở 22 nút

Thời gian chạy: 0.2004499435

**A star search**

Tính đầy đủ: có

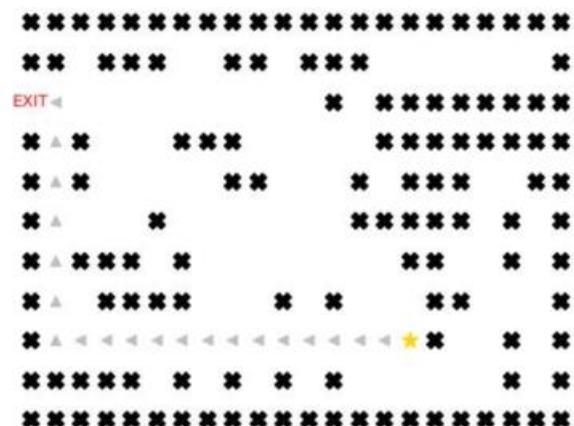
Tính tối ưu: có

Heuristic Manhattan

Chi phí di chuyển: 21

Độ mở các nút: mở 69 nút

Thời gian chạy: 0.2051281929

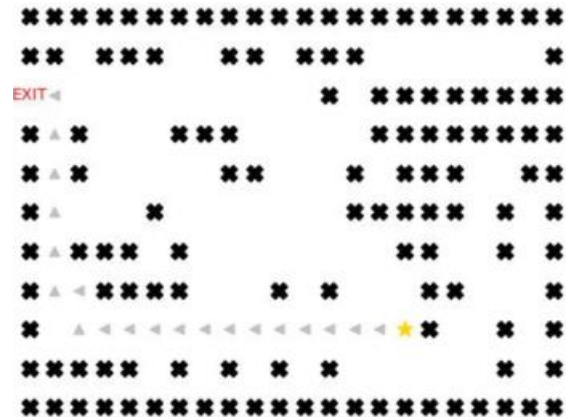


Heuristic Euclid

Chi phí di chuyển: 21

Độ mở các nút: mở 76 nút

Thời gian chạy: 0.2074711323

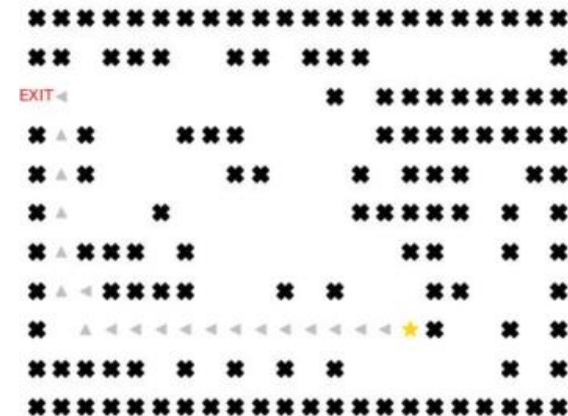


Heuristic Chebyshev

Chi phí di chuyển: 21

Độ mở các nút: mở 80 nút

Thời gian chạy: 0.2213435173

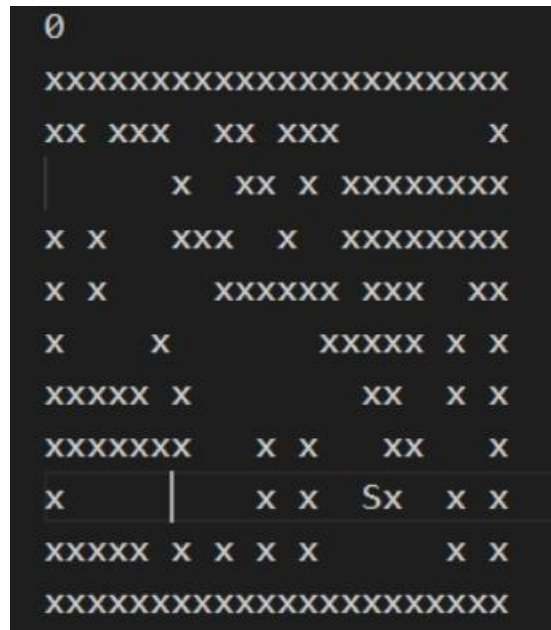


=> Với một bản đồ được tạo ngẫu nhiên, xét về đường đi thì cả 3 heuristic gần như là giống nhau. Việc giảm đi phạm vi hoạt động đã khiến cho thời gian chạy cũng như độ mở của 3 heuristic cũng tương đối nhau với từng thuật toán. Tuy nhiên heuristic Manhattan vẫn được đánh giá cao hơn về tính tối ưu.

Nhận xét

- Ở bản đồ được tạo ngẫu nhiên, tất cả các thuật toán đều tìm được đường đi đến đích, trong các thuật toán thì dfs có đường đi khác nhiều so với các thuật toán còn lại, chi phí có phần lớn hơn
- Về thời gian, trong trường hợp này các thuật toán tương đương nhau, tuy nhiên đây vẫn là trường hợp có lợi hơn đối với các thuật toán không có heuristic, trong những trường hợp khác thì các thuật toán có heuristic vẫn tối ưu hơn
- A* mở nhiều nút hơn hẳn so với GBFS bởi vì GBFS chỉ cố gắng mở các nút có ước lượng gần với đích nhất, còn A* thì xét thêm chi phí đến điểm đang xét ngoài heuristic.

Bản đồ 3



Bản đồ có đường đi của BFS trùng với DFS

Breadth first search

Tính đầy đủ: có

Tính tối ưu: có

Chi phí di chuyển: 21

Độ mở các nút: mở 80 nút

Thời gian chạy: 0.3090693951



Depth first search

Tính đầy đủ: có

Tính tối ưu: có

Chi phí di chuyển: 21

Độ mở các nút: mở 23 nút

Thời gian chạy: 0.2065792084

**Uniform cost search**

Tính đầy đủ: có

Tính tối ưu: có

Chi phí di chuyển: 21

Độ mở các nút: mở 80 nút

Thời gian chạy: 0.1985180378

**Greedy best first search****Heuristic Chebyshev**

Tính đầy đủ: có

Tính tối ưu: có

Chi phí di chuyển: 21

Độ mở các nút: mở 26 nút

Thời gian chạy: 0.2142219543



Heuristic Manhattan

Tính đầy đủ: có

Tính tối ưu: có

Chi phí di chuyển: 21

Độ mở các nút: mở 22 nút

Thời gian chạy: 0.2073311806

**Heuristic Euclid**

Tính đầy đủ: có

Tính tối ưu: có

Chi phí di chuyển: 21

Độ mở các nút: mở 22 nút

Thời gian chạy: 0.2207839489

**A star search****Heuristic Manhattan**

Tính đầy đủ: có

Tính tối ưu: có

Chi phí di chuyển: 21

Độ mở các nút: mở 34 nút

Thời gian chạy: 0.2009818554



Heuristic Euclid

Tính đầy đủ: có

Tính tối ưu: có

Chi phí di chuyển: 21

Độ mở các nút: mở 46 nút

Thời gian chạy: 0.1972701550

**Heuristic Chebyshev**

Tính đầy đủ: có

Tính tối ưu: có

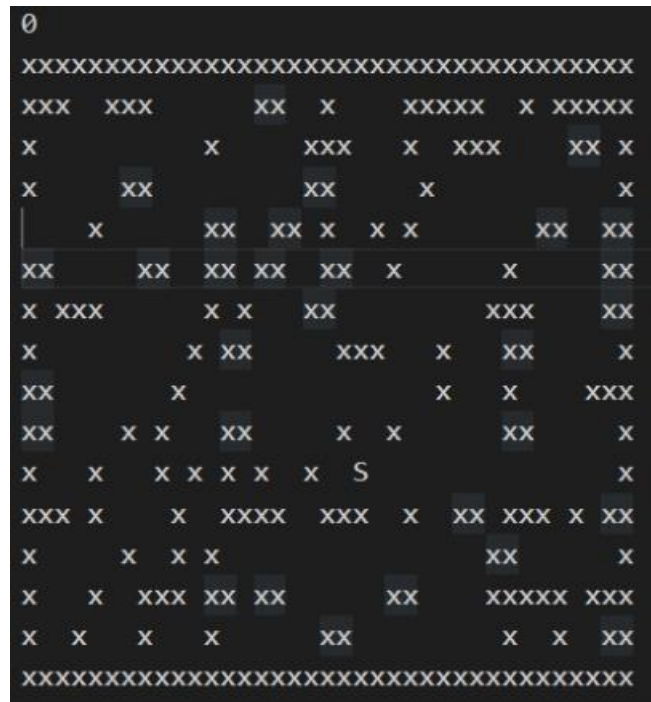
Chi phí di chuyển: 21

Độ mở các nút: mở 57 nút

Thời gian chạy: 0.2095861435

**Nhận xét**

- DFS và BFS trùng đường đi vì thứ tự mở các điểm của DFS được cài đặt là trên phải trái dưới.
- Đường đi của tất cả các thuật toán đều tối ưu, nhưng có sự khác nhau giữa việc chọn đường đi giữa một bên là thuật toán tìm kiếm mù và một bên là thuật toán tìm kiếm có thông tin.
- Trong test này DFS có đích nằm trên đường đào sâu đầu tiên nên trùng với đường đi ngắn nhất đến đích như hình vẽ.
- Trong lần đào đầu tiên DFS tìm được đích nên tập mở nhỏ bằng chi phí đường đi.
- Do lần duyệt đầu tiên của DFS đã tìm thấy đường đi đến đích nên thời gian thoát khỏi mê cung TH này là nhanh nhất.

Bản đồ 4*Bản đồ không có đường đi đến đích*

- Tất cả các thuật toán đều mở 226 nút

Breadth first search	Depth first search	Uniform cost search
Thời gian: :0.1725165653	Thời gian: 0.1609799576	Thời gian: : 0.1857323742

Greedy best first search

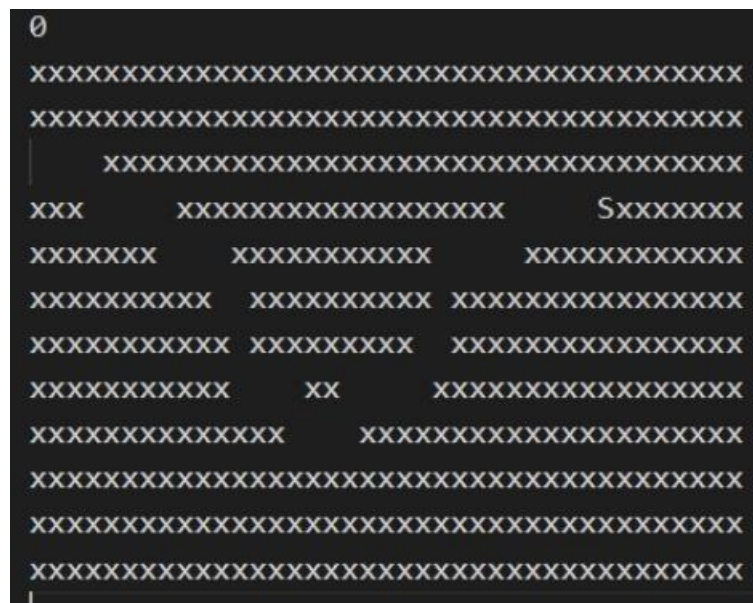
Heuristic Euclid	Heuristic Manhattan	Heuristic Chebyshev
Thời gian: : 0.1801345444	Thời gian: 0.1881144142	Thời gian: : 0.1843018627

A star search:

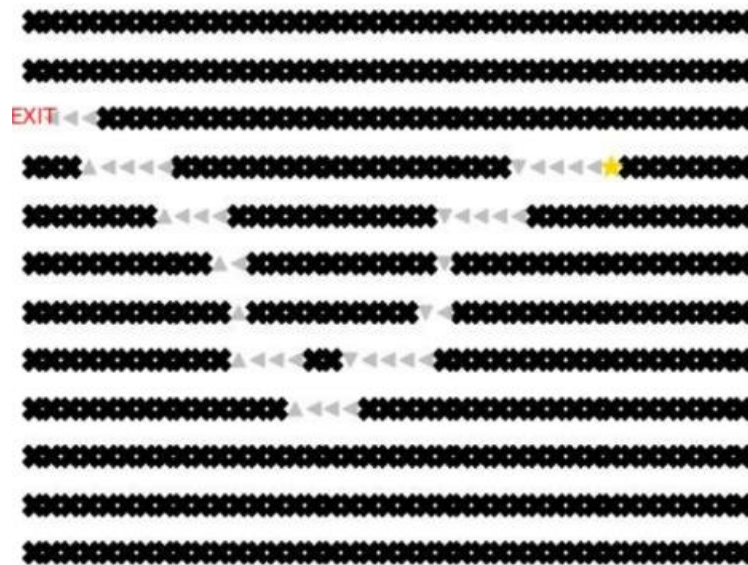
Heuristic Euclid	Heuristic Manhattan	Heuristic Chebyshev
Thời gian: 0.1850497818	Thời gian: 0.1827466583	Thời gian: : 0.1856155491

Nhận xét

- Tất cả các thuật toán đều duyệt qua hết tất cả các nút có thể (226 nút) trước khi trả ra không có đường đi.
- DFS và BFS duyệt nhanh hơn vì không phải cân nhắc heuristic như A* và GBFS hay chọn điểm có chi phí đến nó nhỏ nhất như UCS.
- Giữa các thuật toán tìm kiếm có thông tin không có nhiều sự khác biệt về thời gian chạy.

Bản đồ 5*Bản đồ chỉ có 1 đường đi*

- Các thuật toán đều có đường đi giống nhau



Breadth first search	Depth first search	Uniform cost search
Tính đầy đủ: có	Tính đầy đủ: có	Tính đầy đủ: có
Tính tối ưu: có	Tính tối ưu: có	Tính tối ưu: có
Chi phí di chuyển: 42	Chi phí di chuyển: 42	Chi phí di chuyển: 42
Độ mở các nút: mở 43 nút	Độ mở các nút: mở 43 nút	Độ mở các nút: mở 43 nút
Thời gian: 0.2392811775	Thời gian: 0.2330975533	Thời gian: 0.2561805248

Greedy best first search

Tính đầy đủ: có

Tính tối ưu: có

Chi phí di chuyển: 42

Độ mở các nút: mở 43 nút

Heuristic Euclid	Heuristic Manhattan	Heuristic Chebyshev
Thời gian: 0.2422416306	Thời gian: 0.2433736324	Thời gian: 0.242855787

A star search

Tính đầy đủ: có

Tính tối ưu: có

Chi phí di chuyển: 42

Độ mở các nút: mở 43 nút

Heuristic Euclid	Heuristic Manhattan	Heuristic Chebyshev
Thời gian: 0.2474510040	Thời gian: 0.2581579876	Thời gian: 0.2464489937

Nhận xét:

- BFS và DFS có thời gian chạy nhanh hơn các thuật toán còn lại, do các A* và GBFS phải tính các hàm heuristic còn UCS phải duyệt qua tập mở để lấy điểm có chi phí đến nó thấp nhất.
=> không phải lúc nào các thuật toán tìm kiếm có thông tin cũng tốt hơn các thuật toán tìm kiếm không có thông tin
- Thời gian chạy của GBFS nhanh hơn A*

5. Kịch bản nâng cấp

Kịch bản và Input đầu vào

- Kịch bản: ma trận tương tự như đề bài mô tả (không có điểm thưởng) và có các cánh cửa để dịch chuyển bất kỳ từ điểm (i, j) sang điểm (i', j') (giống teleportation - dịch chuyển tức thời). Nếu ô là cổng dịch chuyển thì khi bước vào buộc phải dịch chuyển sang cổng bên kia và cổng bên kia thì ngược lại.

Việc dịch chuyển giữa cái cổng không mất chi phí. Nếu tìm được đường đi thì xuất ra chi phí, còn không thì xuất NO.

- Input:

+ Dòng đầu tiên 1 số nguyên n mô tả số cánh cửa có thể dịch chuyển qua lại cho nhau.

+ N dòng tiếp mỗi dòng gồm 4 số nguyên a, b, c, d với (a, b) là tọa độ cổng 1, (c, d) là tọa độ cổng 2, hai cổng trên có thể dịch chuyển qua lại cho nhau mà không mất chi phí.

+ Các dòng tiếp theo mà ma trận gồm các kí hiệu như đề đã mô tả trên và có thêm kí hiệu “o” đại diện cho cổng dịch chuyển(số lượng kí hiệu o tất nhiên phải là chẵn vì sẽ gồm các cặp cổng dịch chuyển).

Thuật toán tối ưu

- Ý tưởng: kế thừa thuật toán UCS(cài hàng đợi ưu tiên) và bổ sung việc xử lý các trường hợp cổng dịch chuyển để thuật toán chạy đúng và tối ưu hơn.

- Mô tả cài đặt thuật toán:

+ Dùng mảng gate[][][] để lưu lại các cổng có thể dịch chuyển qua lại cho nhau.

+ UCS_ADVANCE:

- Gồm mảng d[][]: lưu trọng số đường đi của mỗi tọa độ đỉnh, trace[]: lưu vết đường đi dùng khi truy xuất đường đi, heap pq(a, b, c, d) với a là trọng số đường đi đi đến tọa độ (b, c) và d nhận tập giá trị {0, 1} đại diện cho nếu đi đến tọa độ (b, c) là một cổng dịch chuyển từ 1 cổng dịch chuyển khác thì nó là 1 và các trường hợp còn lại là 0.
- Một trong số các thao tác bổ sung cho UCS là khi lấy một đỉnh từ heap ra xét thì nếu có là một cổng dịch chuyển được đi đến từ 1 ô thường thì sẽ cho qua không được xét để đi ngược lại thì đi bình thường.
- Trong quá trình duyệt các đỉnh kể nếu đỉnh k là một cổng dịch chuyển ta thực hiện 2 thao tác. Một là đối với cổng vào: cập nhật trọng số đường đi, lưu vết vào trace và push vào heap với check = 0 đại diện cho việc đi đến cổng dịch chuyển từ một ô thường. Hai là đối với cổng đích: cập nhật trọng số đường đi với tọa độ cổng đích, lưu vết vào trace là hướng đi từ cổng dịch chuyển và push vào heap với check = 1 đại diện cho việc đi đến cổng dịch chuyển từ một cổng dịch chuyển khác.

- Đánh giá:

+ ADVANCE được cài như trên đảm bảo tìm được đường đi đến đích nếu như kịch bản được tạo tồn tại đường đi đến đích => tính đầy đủ: có.

- + Việc tìm được đường đi dựa vào UCS cũng đảm bảo đường đi đến đích khi tìm được cũng sẽ là ngắn nhất => tính tối ưu: có.
- + ADVANCE sẽ có thể tìm kiếm được đường đi đến đích trong một số trường hợp UCS không đi qua cổng dịch chuyển sẽ không có đường đi đến đích.
- + Độ phức tạp ADVANCE tuy có xử lý một số trường hợp nhiều hơn so với UCS nhưng tùy vào việc dịch chuyển mà có thể tìm đến đích nhanh hay chậm hơn nhưng nhìn chung thì ~ độ phức tạp thuật toán UCS.

Chạy bản đồ thiết kế: output với mỗi cặp cổng sẽ được tô một màu

- Bản đồ 1

+ Input – output:

- Trường hợp không có chướng ngại vật “X”



+ Nhận xét:

- Tìm được đường đi ngắn nhất về đến đích với chi phí di chuyển là 9 ngắn hơn hẳn so với các đường đi không qua cổng.
- Việc đi thông qua các cổng dịch chuyển cho đường đi ngắn hơn so với các đường đi khác nên ADVANCE chọn đi thông qua hết tất cả các cổng rồi về đích (Đi qua cặp cổng tím -> xanh -> vàng -> về đích).

- Bản đồ 2

+ Input – output:

- Trường hợp có chướng ngại vật và các cổng dịch chuyển.

```

3
1 1 7 9
1 21 5 24
5 18 9 21
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
XO   XX           X XO  X X
      X      XX  XX X  X X  X
XX    XX  XX  X  XX  X  X
X XXX           XX  S    X
X      XXX XX   OXXX  O X
XX  XX  X      X      XX
XX  X X  O  XX   X  XX  X
X  X      X X X  X      X X
XX  X      X XXXX  XXXO  X X
XXXXXXXXXXXXXXXXXXXXXXXXXXXX

```

```

XXXXXXXXXXXXXXXXXXXXXXXXXXXX
X  1  XX           X  2  X  X
EXIT  X      XX  XX  X  X  X
XX    XX  XX  X  XX  X  X
X  XXXX           XX  S    X
X      XXX XX   OXXX  O X
XX  XX  X      X      XX
XX  X X  O  XX   X  XX  X
X  X      X X X  X      X X
XX  X      X XXXX  XXXO  X X
XXXXXXXXXXXXXXXXXXXXXXXXXXXX

```

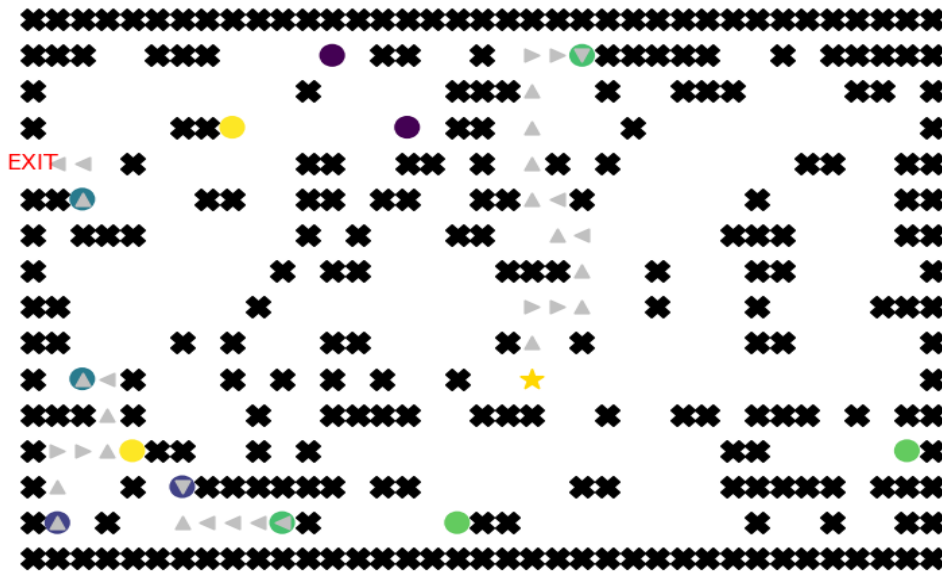
+ Nhận xét:

- Tìm được đường đi ngắn nhất về đến đích với chi phí di chuyển là 21 ngắn hơn so với các đường đi không qua cổng.
- ADVANCE đã chọn đường đi khác đến cổng màu tím để né việc đi vào cổng màu xanh cho nó di chuyển đến vị trí bất lợi hơn.

- Bản đồ 3

+ Input – output:

- Trường hợp nếu không đi qua cổng dịch chuyển sẽ không thể về đích.



```

6
1 22 14 10
13 6 14 1
14 17 12 35
1 12 3 15
12 4 3 8
10 2 5 2
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXX  XXX  O XX  X  OXXXXX  X XXXXX
X          X   XXX  X  XXX  XX  X
X   XXO    O XX   X          X
      X   XX  XX X  X X      XX  XX
XXO   XX  XX XX  XX  X      X   XX
X XXX   X X   XX      XXX  XX
X       X XX   XXX  X  XX   X
XX      X          X  X   XXX
XX   X X   XX   X X      XX   X
X O X   X X X X  X  S          X
XXX X   X  XXXX  XXX  X  XX XXX X XX
X  OXX  X X          XX   OX
X  X OXXXXXX XX      XX  XXXXX XXX
XO X      OX   OXX      X  X  XX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

```

+ Nhận xét :

- ADVANCE tìm được đường đi để thoát ra khỏi mê cung với chi phí khá lớn.
- Nếu không đi vào các cổng dịch chuyển sẽ không thể thoát khỏi mê cung trên.
- ADVANCE đã thông qua cổng dịch chuyển để thoát khỏi hai mê cung cụt không có lối ra.
- Việc xuất hiện các cổng trên đường đi tuy nhiên nếu không tối ưu ADVANCE sẽ không chọn và chọn đi vào cổng cho đường đi thoát khỏi mê cung tối ưu nhất.

6. Thông tin bài nộp:

- Python version: 3.10.8

- Input:

- + level_1: bao gồm 5 bản đồ (input1.txt, input2.txt, input3.txt, input4.txt, input5.txt)
- + level_2: bao gồm 3 bản đồ có điểm thưởng (input1.txt, input2.txt, input3.txt)
- + advance : bao gồm 3 bản đồ có công dịch chuyển (input1.txt, input2.txt, input3.txt)

TÀI LIỆU THAM KHẢO

- [1] [Các thuật toán cơ bản trong AI - Phân biệt Best First Search và Uniform Cost Search \(UCS\) \(viblo.asia\)](#)
- [2] [thuật toán tìm kiếm greedy best-first search – Blog Lập Trình \(wordpress.com\)](#)
- [3] <https://www.youtube.com/watch?v=6TsL96NAZCo>
- [4] <https://www.youtube.com/watch?v=h1RYvCfuon4>
- [5] <https://www.youtube.com/watch?v=dRMvK76xQJI&t=490s>
- [6] <https://www.youtube.com/watch?v=dv1m3L6QXWs>
- [7] [Slide bài giảng của thầy.](#)