

# ARTIFICIAL INTELLIGENCE REPORT

AI

AI - SEARCH

## MEMBERS:

20120264 - Trần Hải Đăng

20120138 - Lê Thành Nam

20120340 - Trần Nhật Nguyên

## **ĐÁNH GIÁ MỨC ĐỘ HOÀN THÀNH**

**1. Bản đồ không có điểm thưởng:**

Mức 1a: 100%

Mức 1b: 100%

**2. Bản đồ có điểm thưởng**

Mức 2a: 100%

Mức 2b: 100%

**3. Điểm cộng: 0%**



# TABLE OF CONTENTS

## SECTION 1 (BASIC MATRIX)

<b>Chú thích:</b>	<b>03</b>
<b>So sánh các map</b> (hình ảnh và nhận xét)	<b>04</b>
<b>So sánh heuristic</b> (vẽ biểu đồ và nhận xét)	<b>25</b>
<b>So sánh các thuật toán</b> (vẽ biểu đồ và nhận xét)	<b>26</b>
<b>So sánh các thuật toán</b> (vẽ biểu đồ và nhận xét)	<b>26</b>

## SECTION 2

(MATRIX WITH BONUS POINTS)

<b>Thuật toán 1:</b>	<b>28</b>
<b>Minh họa thuật toán 1</b> (với 3 bản đồ)	<b>29</b>
<b>Thuật toán 2:</b>	<b>32</b>
<b>Minh họa thuật toán 2</b> (với 3 bản đồ)	<b>33</b>
<b>Nguồn tham khảo</b>	<b>36</b>
<b>Lời cảm ơn</b>	<b>37</b>

# CHÚ THÍCH

GBFS

$f(n) = h(n)$  với  $h(n)$  là hàm heuristic ước lượng chi phí từ 1 trạng thái nhất định đến đích

A \*

$f(n) = g(n) + h(n)$  với  $g(n)$  là khoảng cách tới 1 trạng thái nhất định và  $h(n)$  định nghĩa ở trên

MANHATTAN

```
def Manhattan(cell1, cell2):
```

```
    x1, y1 = cell1
```

```
    x2, y2 = cell2
```

```
    return (abs(x1 - x2) + abs(y1 - y2))
```

EUCLID

```
def Euclid(cell1, cell2):
```

```
    x1, y1 = cell1
```

```
    x2, y2 = cell2
```

```
    return math.sqrt((x1-x2)**2 + (y1-y2)**2)
```

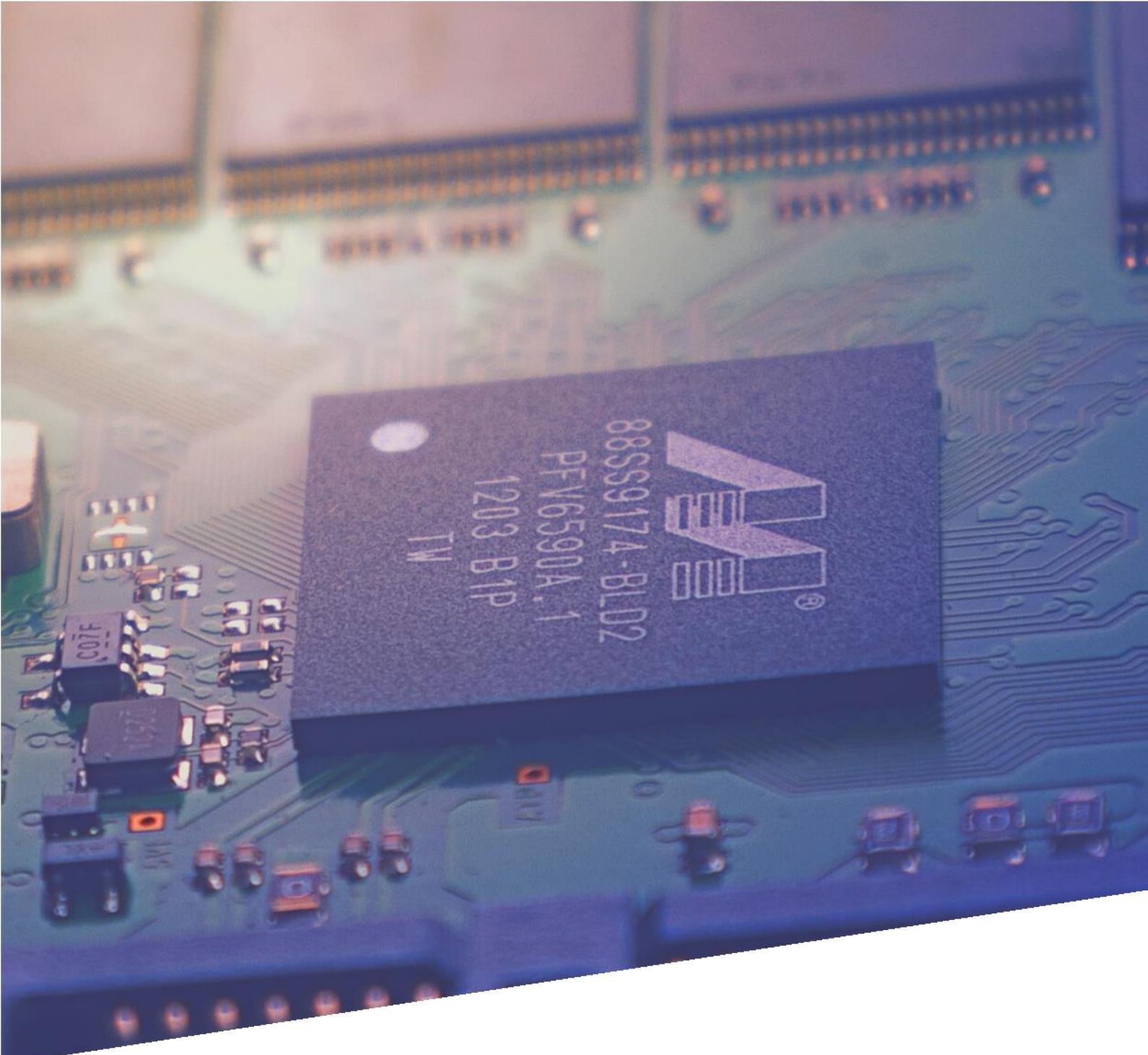
OCTILE

```
def Octile(cell1, cell2):
```

```
    dx = abs(cell1[0] - cell2[0])
```

```
    dy = abs(cell1[1] - cell2[1])
```

```
    return (dx + dy) + (math.sqrt(2)) * min(dx, dy)
```



ARTIFICIAL INTELLIGENCE

# SO SÁNH MAP 1 2 3 4 5

---

FIND PATH TO WAY OUT MATRIX

# MAP 1

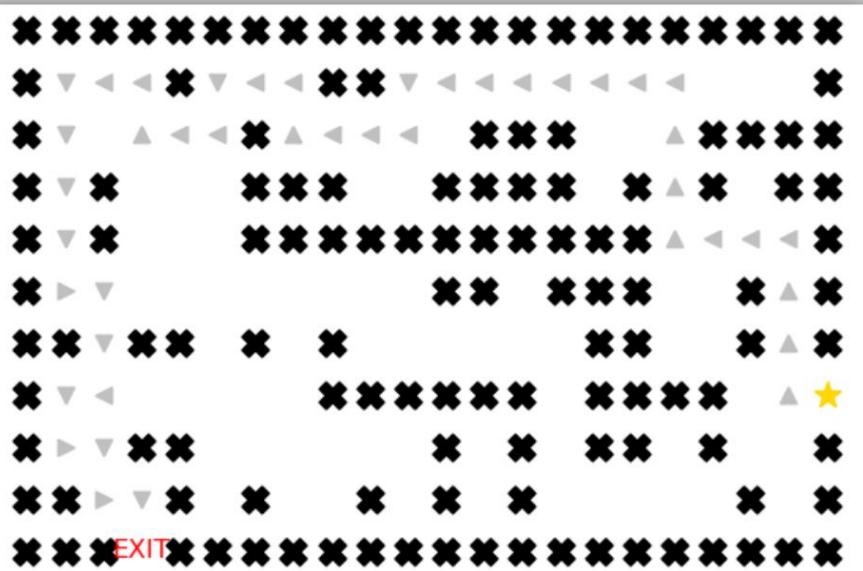
Ta sử dụng đồ với nhiều điểm uốn lượn kiểm tra xem các thuật toán sẽ xử lý khác nhau như thế nào

**DFS**

**Độ dài final path:** 44

**Độ dài search path:** 44

**Run time:** 1

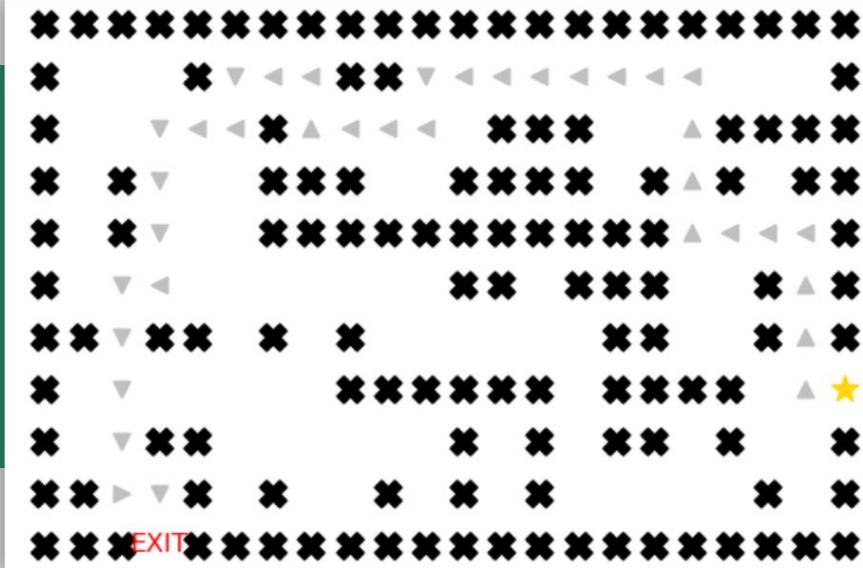


**BFS**

**Độ dài final path:** 38

**Độ dài search path:** 98

**Run time:** 1.5





## Độ dài final path: 40

## Độ dài search path: 44

# Run time: 1

A 10x10 grid puzzle. The board contains the following symbols:

- x**: Black cross symbol.
- o**: Gray circle symbol.
- \***: Gray diamond symbol.
- >**: Gray right-pointing triangle symbol.
- <**: Gray left-pointing triangle symbol.
- : Gray square symbol.
- ★**: Yellow star symbol.

The objective is to move from the top-left corner to the bottom-right corner (marked with a yellow star) using the available symbols as movement instructions. A green vertical bar is located on the far left edge of the grid.

# GBFS

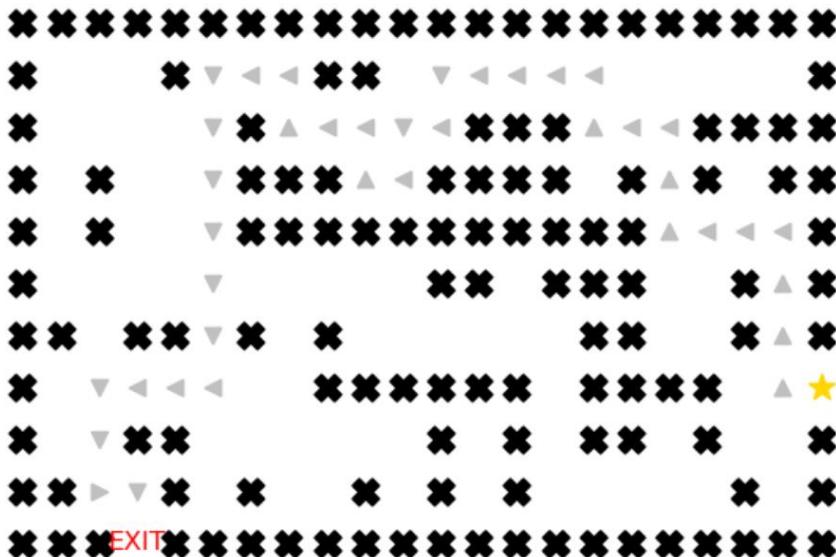
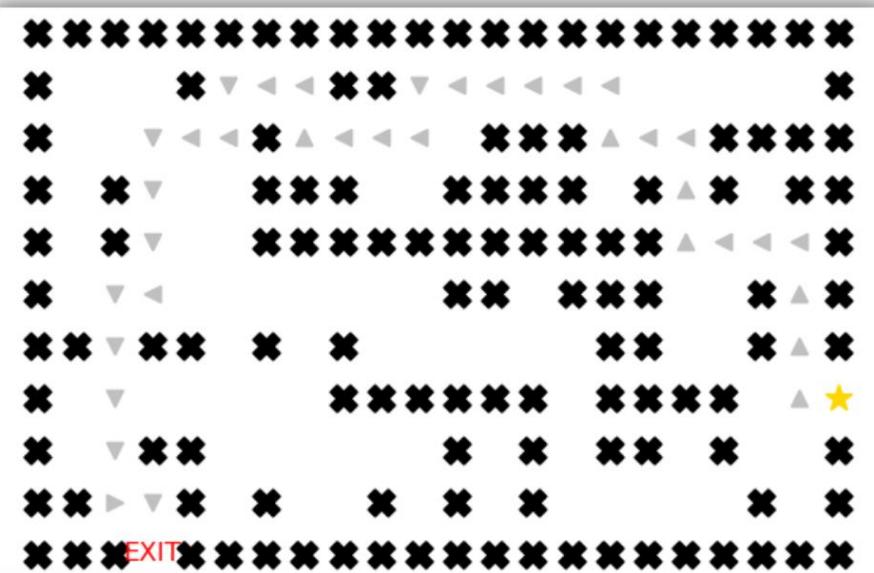
## Các hàm heuristic được sử dụng: Euclid, Manhattan, Octile

**Manhattan:**

**Độ dài final path: 38**

**Độ dài search path: 62**

**Run time: 0.99**



**Octile**

**Độ dài final path: 44**

**Độ dài search path: 80**

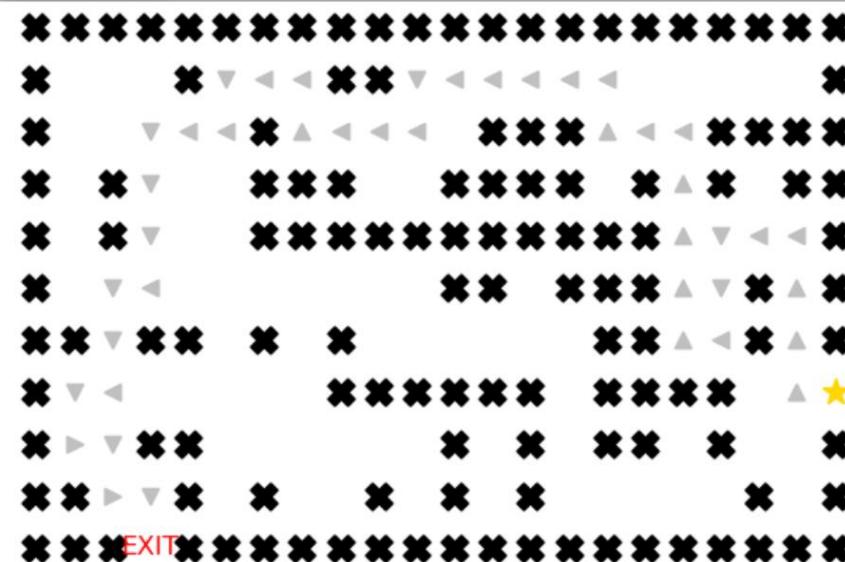
**Run time: 1.998**

**Euclid**

**Độ dài final path: 40**

**Độ dài search path: 64**

**Run time: 0.999**



A\*

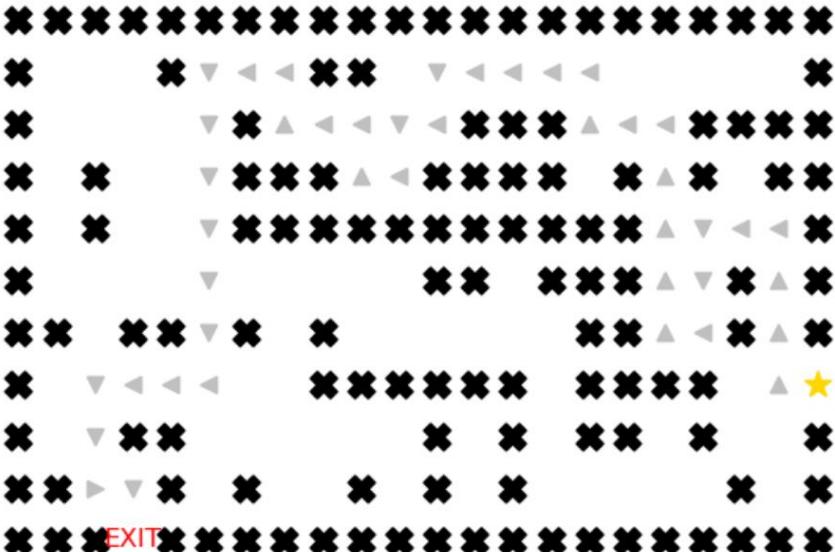
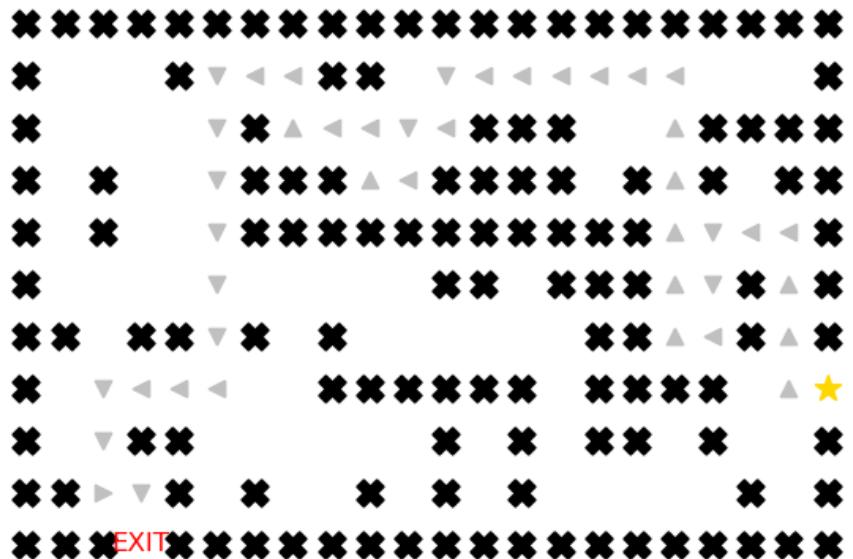
## Các hàm heuristic được sử dụng: Euclid, Manhattan, Octile

**Manhattan:**

Độ dài final path: 44

Độ dài search path: 73

Run time: 1.996



**Octile**

Độ dài final path: 44

Độ dài search path: 76

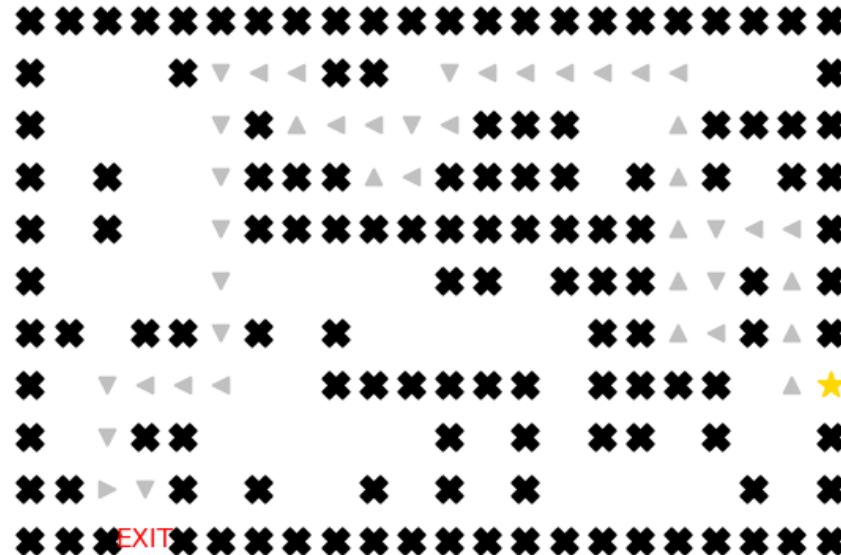
Run time: 1.999

**Euclid**

Độ dài final path: 44

Độ dài search path: 95

Run time: 2



# NHẬN XÉT

ARTIFICIAL INTELLIGENCE

## DFS



Độ dài đường đi thuật toán tìm kiếm bằng với đường đi cuối cùng và cũng gần bằng với đường đi ngắn nhất

## BFS

Tìm được đường đi tối ưu tuy nhiên không gian tìm kiếm gần như là mọi ô trống trên ma trận. Với những bản đồ nhỏ thì thời gian tìm kiếm của BFS và DFS không có quá nhiều khác biệt

## UCS

Tương tự với BFS nhưng do sử dụng chi phí để tìm đường đi nên có ít Search Path hơn.

## GBFS

Ưu tiên theo hàm heuristic là khoảng cách đến đích nên sẽ tìm đường đi gần như tối ưu với thời gian và không gian tìm kiếm ở mức khá nhỏ.

## A STAR

A\* bên cạnh heuristic còn xét thêm khoảng cách từ điểm bắt đầu nên dù vẫn tìm đường đi gần như tối ưu tuy nhiên không gian tìm kiếm lớn hơn GBFS, DFS và chỉ ít hơn BFS

**LƯU Ý: ĐỐI VỚI GBFS VÀ A\* TA SẼ XÉT HÀM HEURISTIC LÀ KHOẢNG CÁCH EUCLID ĐỂ SO SÁNH 4 THUẬT TOÁN**

# MAP 2

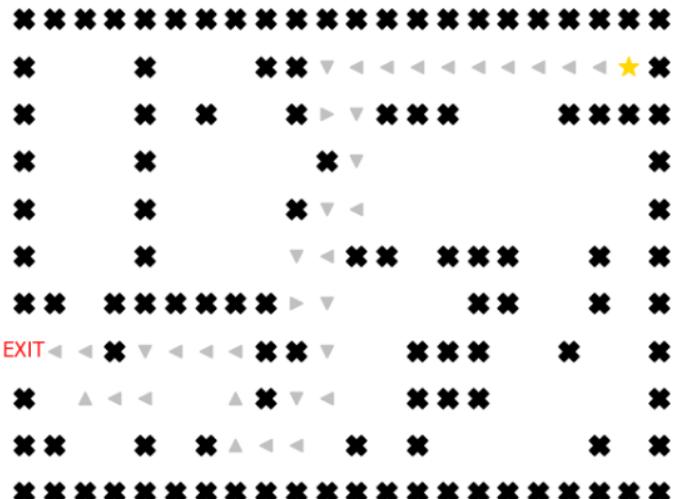
Với map này, việc sử dụng bản đồ với nhiều điểm gấp khúc và bố trí các điểm đầu cuối phức tạp hơn khi thử tự ưu tiên đường đi là up, left, right, down.

**DFS**

Độ dài final path: 104

Độ dài search path: 188

Run time: 4.996

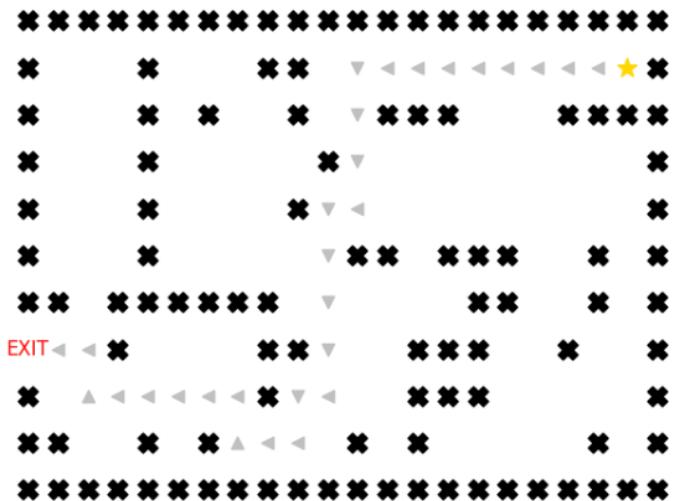


**BFS**

Độ dài final path: 40

Độ dài search path: 236

Run time: 1.997

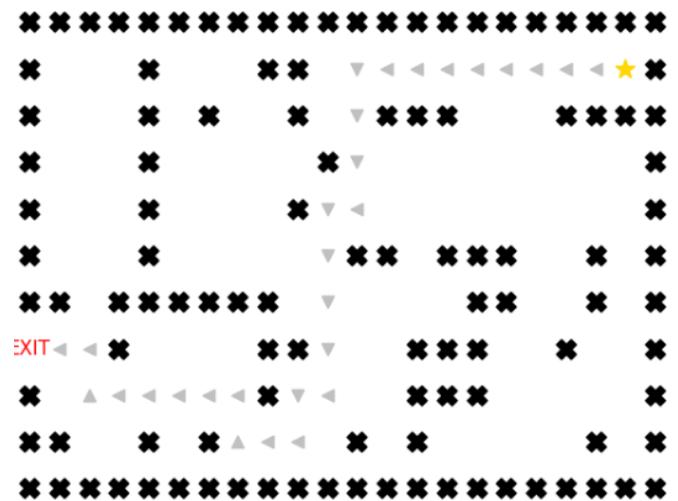


**UCS**

Độ dài final path: 40

Độ dài search path: 150

Run time: 1.44



# MAP 2

GBFS

Với map này, việc sử dụng bản đồ với nhiều điểm gãp khúc và bố trí các điểm đầu cuối phức tạp hơn khi thứ tự ưu tiên đường đi là up, left, right, down.

Manhattan:

Độ dài final path: 44

Độ dài search path: 73

Runtime: 1



Octile:

Độ dài final path: 44

Độ dài search path: 76

Runtime: 0.99

Euclid:

Độ dài final path: 44

Độ dài search path: 95

Runtime: 0.999



# MAP 2

A\*

Với map này, việc sử dụng bản đồ với nhiều điểm gãp khúc và bố trí các điểm đầu cuối phức tạp hơn khi thử tự ưu tiên đường đi là up, left, right, down.

Manhattan:

Độ dài final path: 40

Độ dài search path: 99

Runtime: 1



Euclid:

Độ dài final path: 40

Độ dài search path: 100

Runtime: 1.99

Octile:

Độ dài final path: 40

Độ dài search path:

113

Runtime: 2.98



# NHẬN XÉT

DFS

Thứ tự ưu tiên trong thuật toán là Lên → Trái → Xuống → Phải ngược với hướng cần phải đi trong mê cung nên đường đi DFS tìm được là rất xa so với đường đi ngắn nhất. Ngay cả không gian tìm kiếm cũng không ít hơn BFS là mấy. Bên cạnh đó, thời gian tìm kiếm là rất lớn.

BFS

Ta có thể thấy so với map 1 thì ở map 2 điểm kết thúc gần với điểm bắt đầu hơn cùng với việc không ảnh hưởng bởi hướng của điểm kết thúc so với điểm bắt đầu nên không gian tìm kiếm và đường đi cuối cùng BFS trả về cũng tối ưu hơn nhiều

UCS

Final path của thuật toán UCS giống với BFS nhưng do sử dụng chi phí để tìm đường nên có search path ít hơn

GBFS

GBFS cũng hưởng lợi khi điểm kết thúc gần với điểm bắt đầu khi có được đường đi ngắn nhất cùng không gian tìm kiếm là rất nhỏ. Thời gian tìm kiếm của Greedy cũng là nhỏ so với 3 thuật toán còn lại

A\*

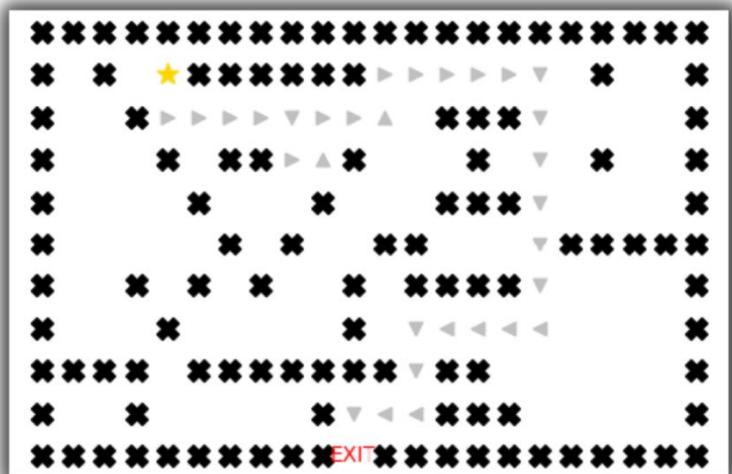
A\* không bị ảnh hưởng nhiều với với hướng đi trong mê cung nên vẫn tìm được đường đi ngắn nhất với không gian tìm kiếm là trung bình

# MAP 3

Việc sử dụng bản đồ với mục đích kiểm tra xem khi gặp các đường cùt thì với các thuật toán khác nhau sẽ cho ra kết quả như thế nào?

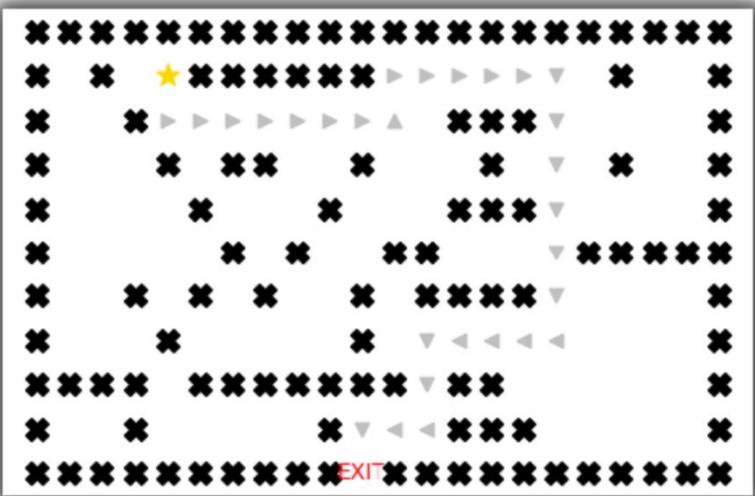
DFS

Độ dài final path: 32  
Độ dài search path: 60  
Run time: 1



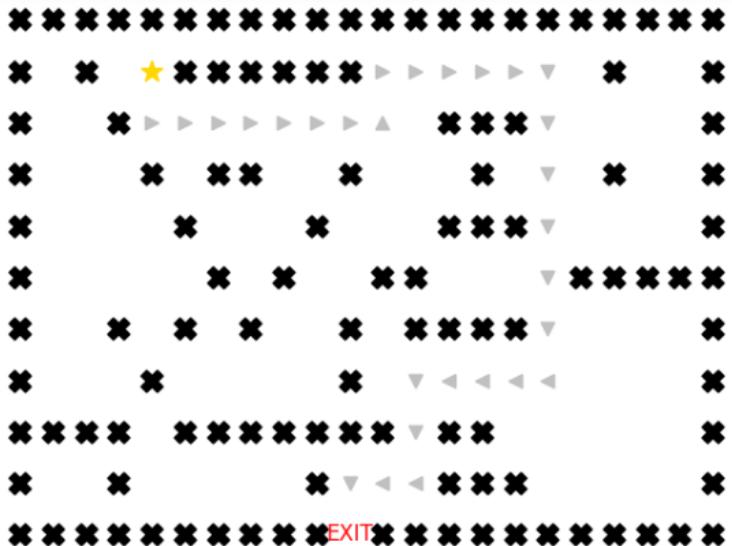
BFS

Độ dài final path: 30  
Độ dài search path: 93  
Run time: 1



UCS

Độ dài final path: 30  
Độ dài search path: 150  
Run time: 1.534



# MAP 3

**GBFS**

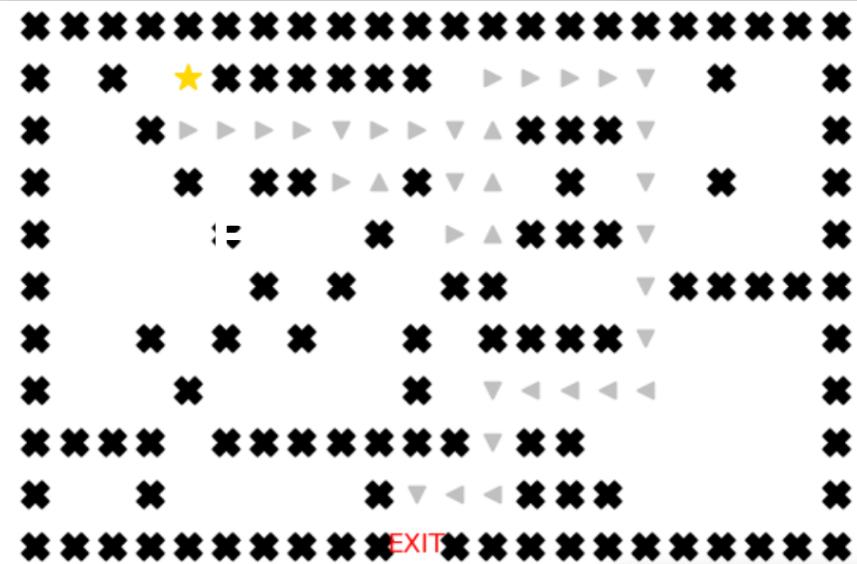
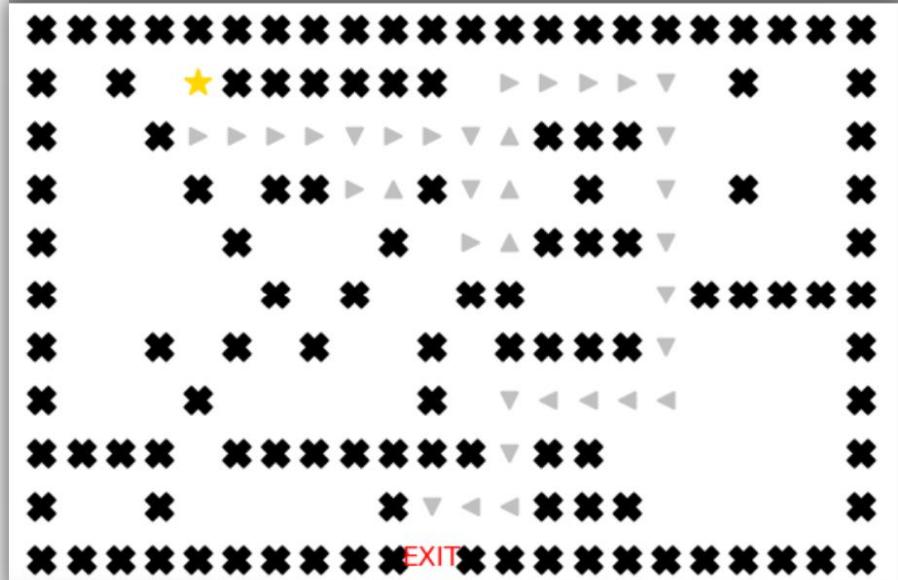
**Việc sử dụng bản đồ với mục đích kiểm tra xem khi gặp các đường cùt thì với các thuật toán khác nhau sẽ cho ra kết quả như thế nào?**

Manhattan

Độ dài final path: 36

Độ dài search path: 68

Runtime: 1.999



Euclid

Độ dài final path: 36

Độ dài search path: 67

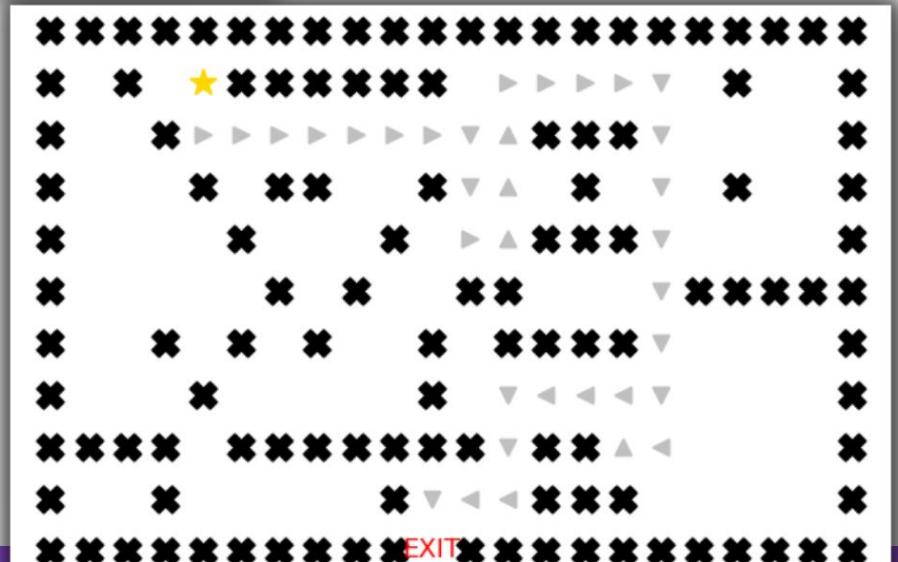
Run time: 1

Octile

Độ dài final path: 36

Độ dài search path: 68

Run time: 0.992



# MAP 3

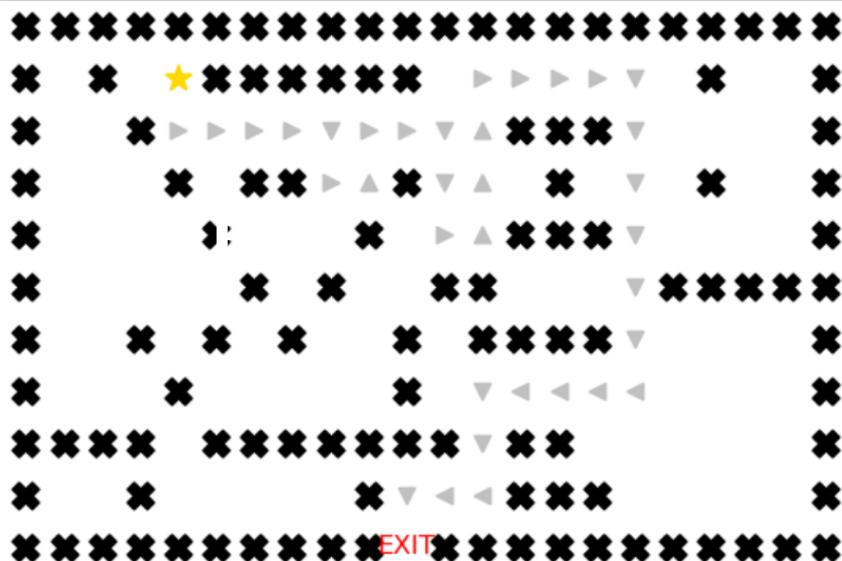
**Việc sử dụng bản đồ với mục đích kiểm tra xem khi gặp các đường cùt thì với các thuật toán khác nhau**

Manhattan

Độ dài final path: 36

Độ dài search path: 68

Runtime: 3.997



Euclid

Độ dài final path: 36

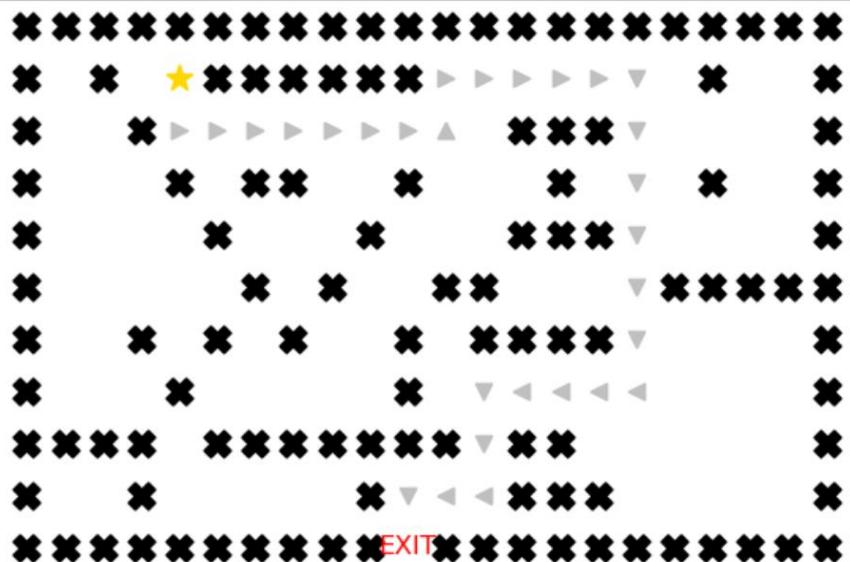
Độ dài search path: 68

Run time: 2.996

Octile

Độ dài final path: 30

Độ dài search path: 68



# NHẬN XÉT

01

## DFS

DFS vẫn tìm ra đường đi gần như ngắn nhất nhưng vì mê cung có nhiều đường cùt gây nhiễu làm cho không gian tìm kiếm tương đối lớn hơn so với bình thường.

02

## BFS

Các đường cùt trong mê cung không ảnh hưởng tới BFS vì thuật toán chỉ tìm kiếm theo chiều rộng. Do đó BFS dễ dàng tìm được đường đi ngắn nhất

03

## UCS

Các đường cùt trong mê cung khiến thuật toán phải quay lui nhiều lần khiến chi phí tăng lên

03

## GBFS + A\*

GBFS và A\* đều bị các đường cùt gây nhiễu vì chúng ưu tiên các điểm gần đích nhất để xét đến. Từ đó, đường đi cuối cùng có độ dài không còn tối ưu và không gian tìm kiếm cũng tương đối lớn. Đặc biệt với A\*, thời gian tìm kiếm là lâu hơn bình thường

# MAP 4

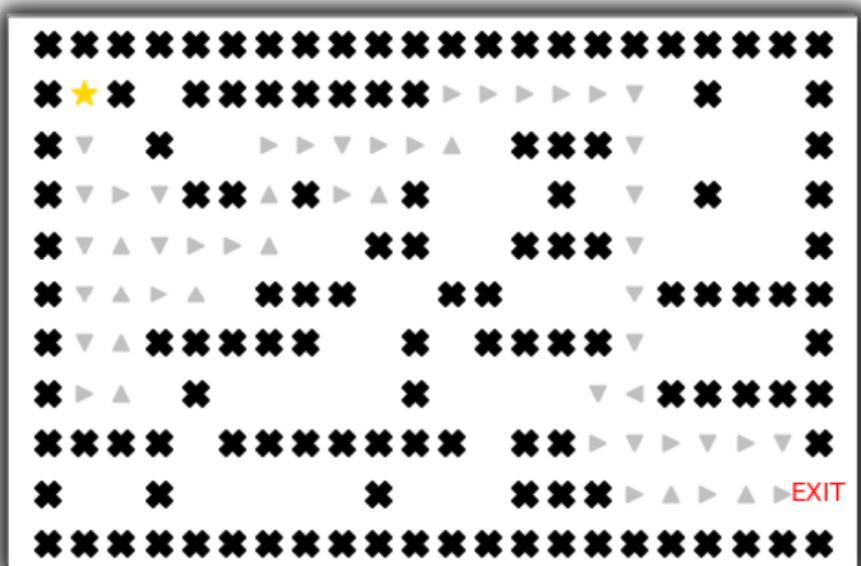
Sử dụng bản đồ bõ trí nhiều bãy vẽ độ dài khoảng cách tới điểm kết thúc để xem xét các lỗi xử lý của từng thuật toán

DFS

Độ dài final path: 53

Độ dài search path: 77

Run time: 1



BFS

Độ dài final path: 35

Độ dài search path: 96

Run time: 1.1

UCS

Độ dài final path: 45

Độ dài search path: 120

Run time: 1.67

# MAP 4

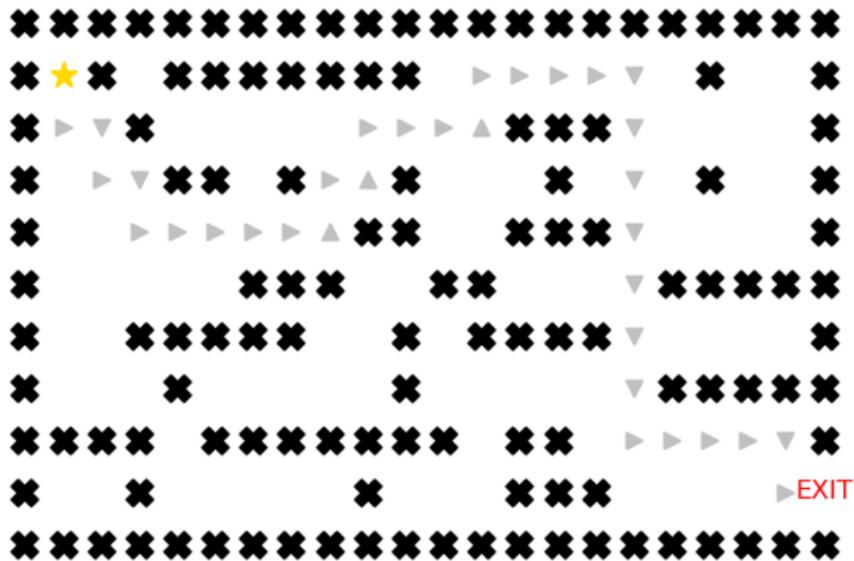
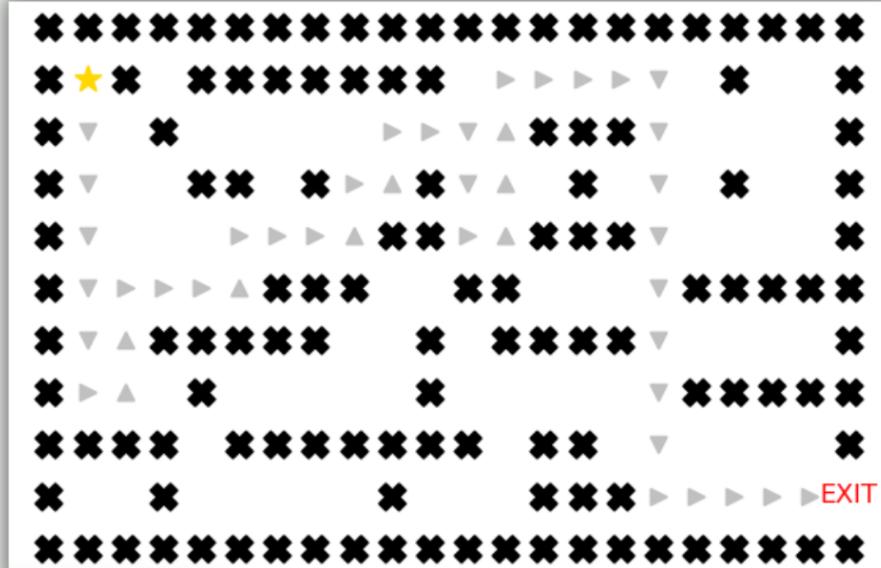
Sử dụng bản đồ bõ trí nhiều bãy vẽ độ dài khoảng cách tới điểm kết thúc để xem xét các lối xử lý của từng thuật toán

## Manhattan

Độ dài final path: 45

Độ dài search path: 67

Run time: 1



## Euclid

Độ dài final path: 35

Độ dài search path: 63

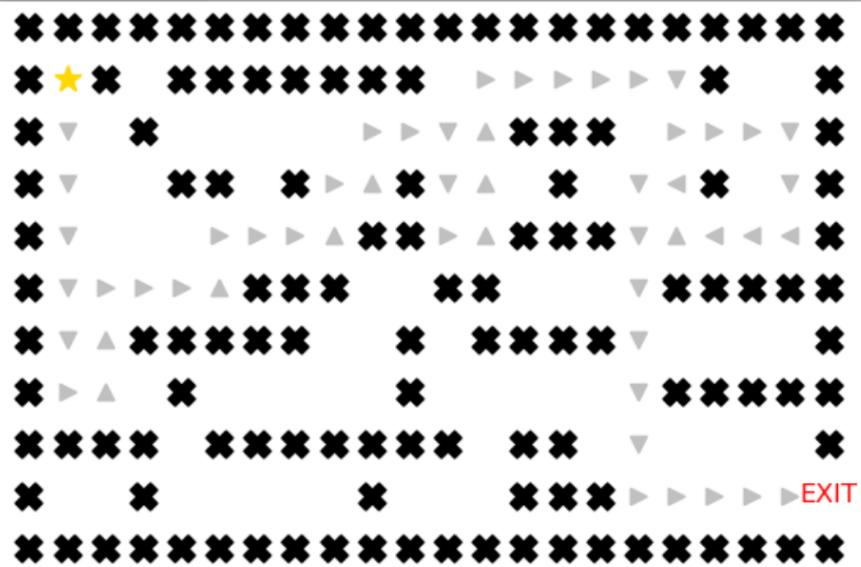
Run time: 0.999

## Octile

Độ dài final path: 55

Độ dài search path: 77

Run time: 1.997



# MAP 4

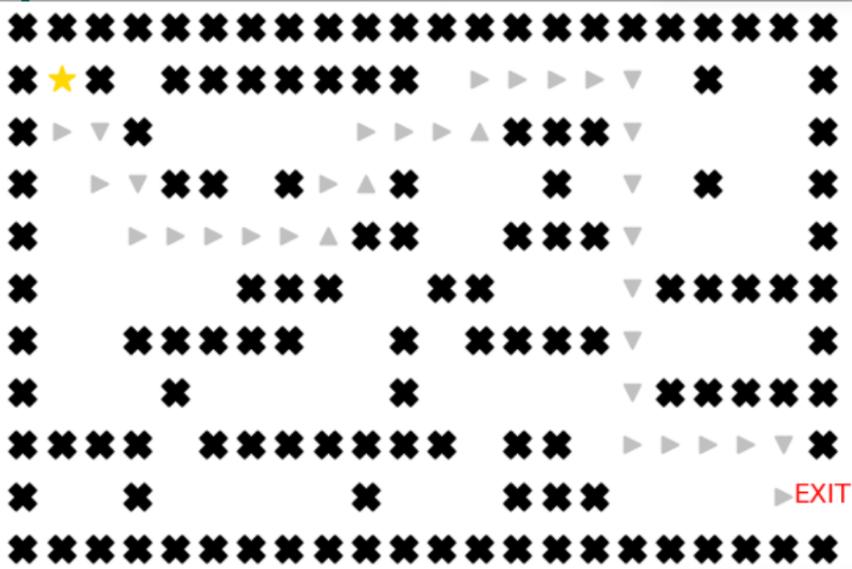
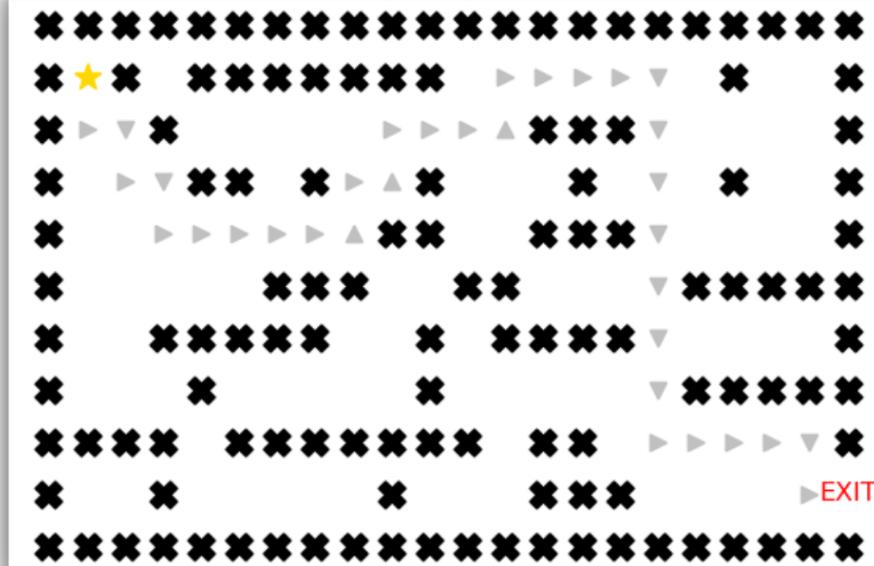
Sử dụng bản đồ bõ trí nhiều bãy vẽ độ dài khoảng cách tới điểm kết thúc để xem xét các lối xử lý của từng thuật toán

## Manhattan

Độ dài final path: 45

Độ dài search path: 67

Run time: 1.991

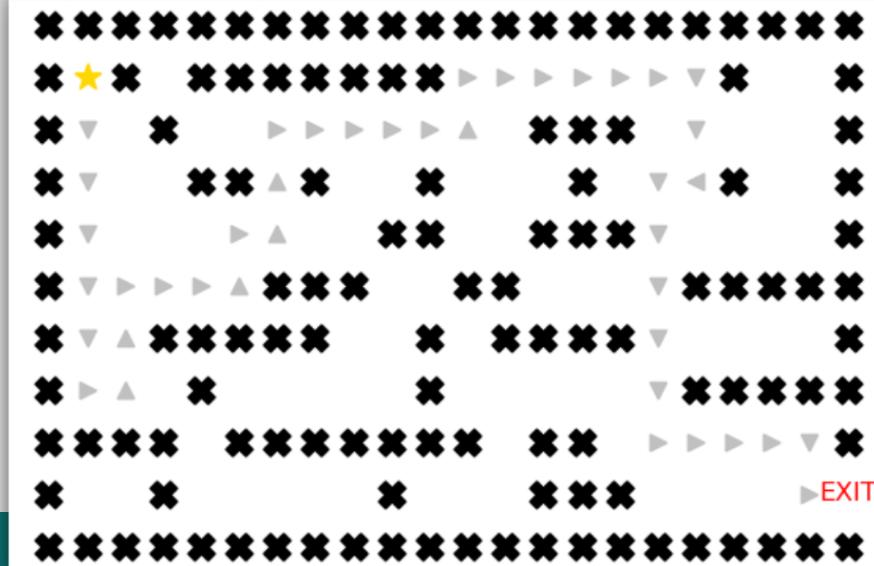


## Octile

Độ dài final path: 43

Độ dài search path: 83

Run time: 4.995



# NHẬN XÉT



**01**

## **DFS**

Các bẫy về độ dài khoảng cách làm cho đường đi của DFS bị nhiễu, xa hơn rất nhiều so với đường đi tối ưu. Điều đó cũng gây ảnh hưởng đến không gian tìm kiếm.



**02**

## **BFS**

Tương tự với đường cùt, các bẫy trong mê cung không gây ảnh hưởng nhiều tới BFS. Qua việc lưu h้าu như tất cả các nút trong hàng đợi, thuật toán tìm được đường đi ngắn nhất dù phải đánh đổi bằng không gian tìm kiếm



**03**

## **GBFS**

Có vẻ như các bẫy về độ dài khoảng cách không làm cho hiệu năng của GBFS tệ đi. Thuật toán tìm được đường đi ngắn nhất cùng không gian tìm kiếm rất nhỏ, thậm chí nhỏ hơn cả DFS. Tuy nhiên ta sẽ còn phải kiểm chứng ở bản đồ sau



**04**

## **A STAR**

Việc kiểm soát cả khoảng cách từ điểm bắt đầu đến điểm đang xét cũng như điểm đang xét đến điểm kết thúc giúp A\* không bị các bẫy làm giảm hiệu năng. Thuật toán vẫn cho được đường đi ngắn nhất cùng độ lớn không gian tìm kiếm không chênh nhiều với DFS

## MAP 5

Sử dụng bản đồ 16x35 với đầy đủ các hình thức như bầy trên đường đi, nhiều đường tới đích , ...

DFS



Độ dài final path: 99

Độ dài search path: 161

Run time: 5

BFS

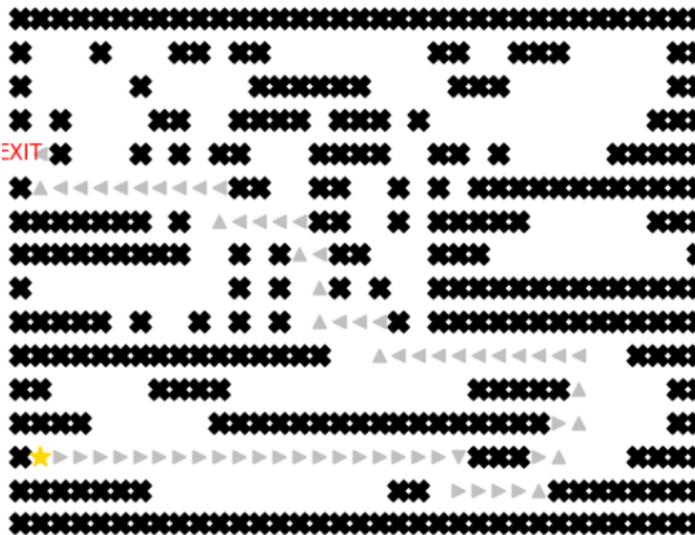


Độ dài final path: 67

Độ dài search path: 231

Run time: 3

UCS



Độ dài final path: 67

Độ dài search path: 100

Run time: 2

# MAP 5

Sử dụng bản đồ 16x35 với đây đủ các hình thức như bẫy trên đường đi, nhiều đường tới đích , ...

Manhattan

Độ dài final path: 97

Độ dài search path: 169

Run time: 4.997



Octile

Độ dài final path: 97

Độ dài search path: 194

Run time: 5.994



# MAP 5

A\*

Sử dụng bản đồ 16x35 với đầy đủ các hình thức như bẫy trên đường đi, nhiều đường tới đích , ...

Manhattan

Độ dài final path: 71

Độ dài search path: 119

Run time: 2.998



Euclid

Độ dài final path: 71

Độ dài search path: 132

Run time: 3

Octile

Độ dài final path: 69

Độ dài search path: 120

Run time: 2



# NHẬN XÉT

## DFS

Với việc kích thước bản đồ lớn, có nhiều bẫy khiến cho độ dài đường đi DFS tìm được rất xa so với đường đi tối ưu. Không gian tìm kiếm cũng do đó mà tăng lên làm hiệu năng của thuật toán giảm mạnh trong đó thời gian chạy thuật toán cũng rất lớn.

## BFS

Vì bản đồ lớn cũng như có nhiều đường khác nhau để đến được đích nên thời gian và không gian tìm kiếm của BFS cực kì lớn. Đổi lại, BFS vẫn tìm được đường đi ngắn nhất đến điểm kết thúc

## UCS

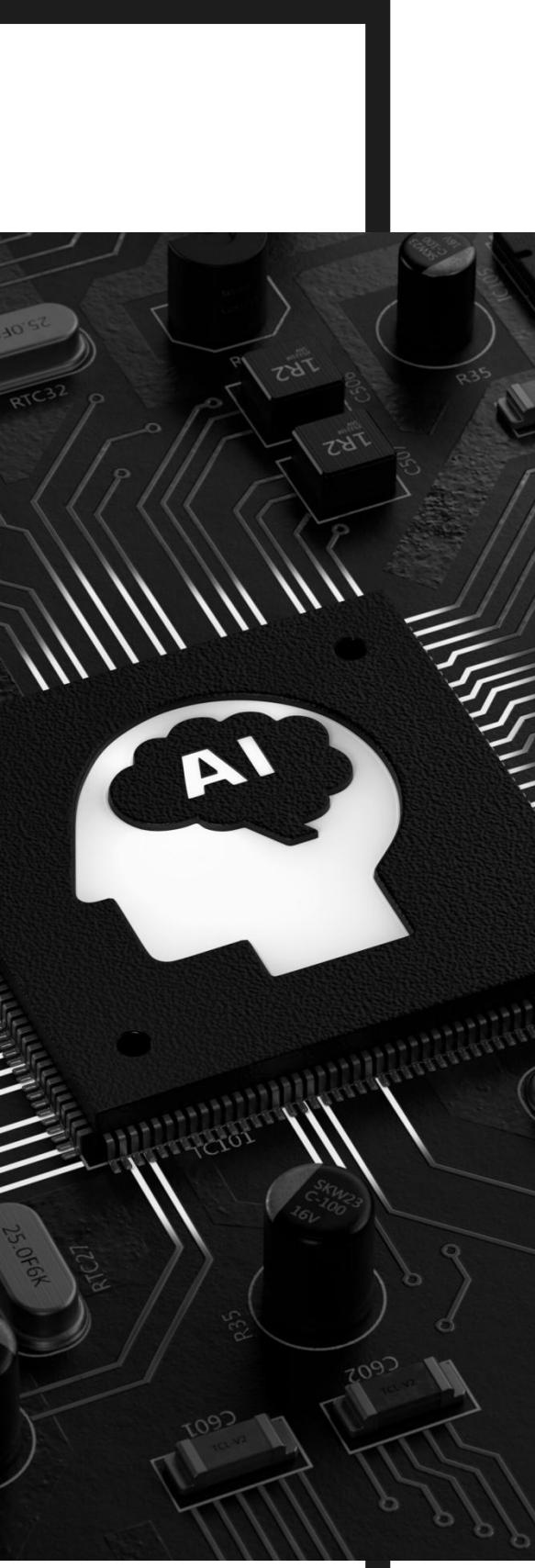
Bản đồ rộng nhưng không quá phức tạp nên thời gian tìm kiếm của UCS nhanh hơn khá nhiều so với các thuật toán khác

## GBFS

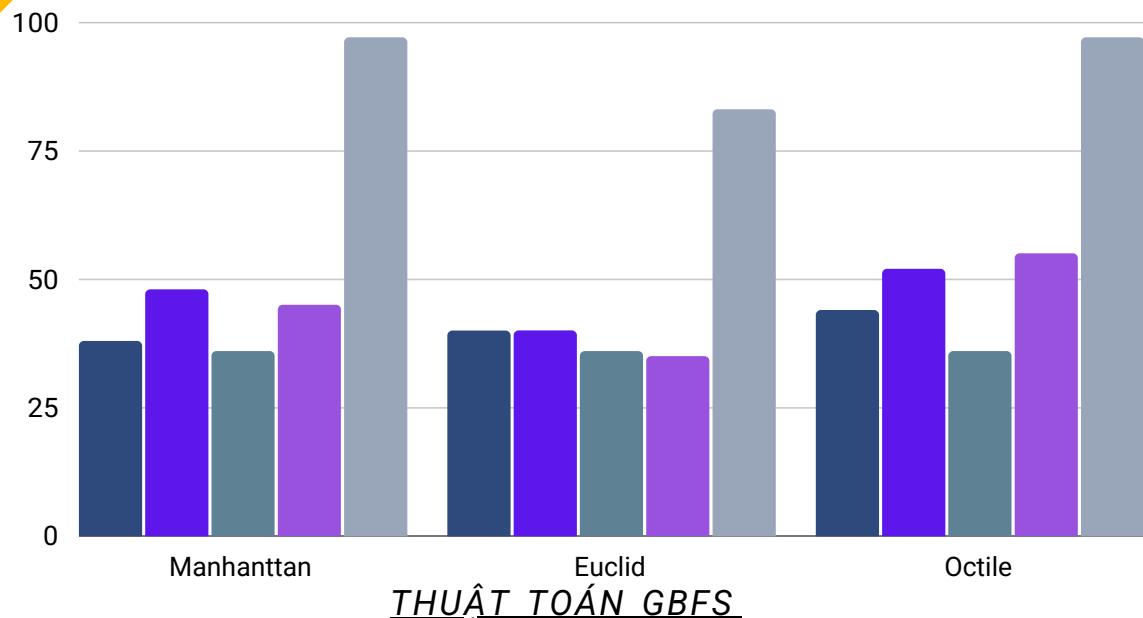
Lúc này, do bản đồ lớn cũng như các bẫy về khoảng cách đã làm cho GBFS bị nhiều ở nhiều nơi. Do đó, đường đi không còn là đường đi tối ưu. Dù vậy, không gian tìm kiếm vẫn khá nhỏ so với các thuật toán còn lại. Thời gian tìm kiếm cũng có thể gọi là tối ưu nhất trong 4 thuật toán,

## A STAR

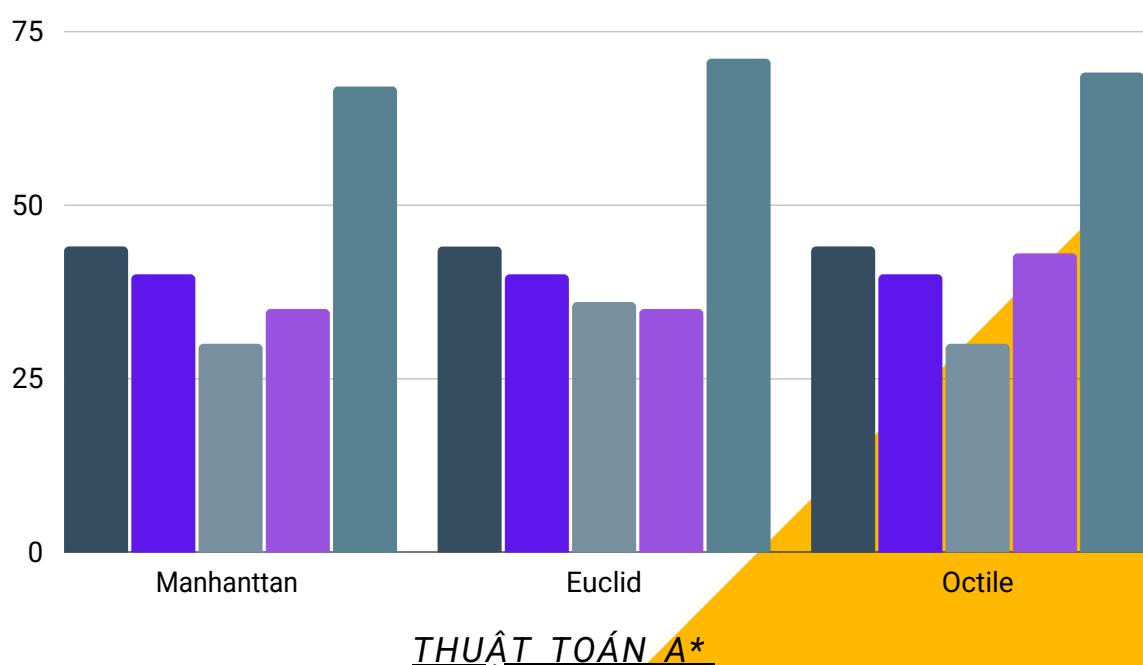
Có thể thấy thuật toán A\* khá ổn định trong mọi loại mê cung. Hầu như thuật toán này đều cho được đường đi tối ưu hoặc gần tối ưu với nhiều kích thước bản đồ, đường cùt và các bẫy về độ dài khoảng cách. Không gian tìm kiếm của A\* cũng ở mức trung bình so với BFS



# COMPARE HEURISTIC



- **Đối với GBFS**, hàm Euclid hầu như luôn là hàm giúp thuật toán tìm ra được đường đi ngắn nhất, đặc biệt là khi bản đồ có kích thước lớn (bản đồ 2 và 5), hàm Manhattan cũng tìm ra đường đi gần như tối ưu nhưng không gian tìm kiếm lớn hơn nhiều so với khi dùng Euclid. Cả 2 hàm Euclid và Manhattan đều có thời gian tìm kiếm không quá lớn Cuối cùng, hàm Octile hiếm khi tìm ra đường đi tối ưu (hoặc gần tối ưu) và không gian tìm kiếm cũng rất lớn.
- **Đối với A\***, hàm Manhattan lại là hàm thường xuyên tìm ra được đường đi tối ưu nhất với mọi loại bản đồ với không gian tìm kiếm tương đối nhỏ. Còn hàm Euclid lại không quá vượt trội so với hàm Octile về độ dài đường đi cuối cùng khi cả 2 hàm đều có những bản đồ bị nhiễu khi tìm đường đi. Tương tự với thời gian và không gian tìm kiếm của cả Euclid và Octile.



# COMPARE ALL MAP

## SO SÁNH SEARCH PATH CÁC THUẬT TOÁN QUA CÁC MAP (GBFS VÀ A\* SỬ DÙNG HÀM HEURISTIC LÀ EUCLID)

- Có thể thấy với những bản đồ có kích thước lớn hoặc bản đồ có hướng của điểm kết thúc so với điểm bắt đầu khác với thứ tự ưu tiên thì DFS luôn tìm đường đi cuối cùng khá xa so với đường đi tối ưu.
- Thuật toán GBFS thường xuyên có được đường đi ngắn nhất và hiệu năng (không gian và thời gian tìm kiếm) tối ưu nhất, tuy nhiên lại bị ảnh hưởng bởi kích thước bản đồ cùng các bẫy về độ dài khoảng cách làm giảm hiệu năng
- Thuật toán BFS luôn có được đường đi tối ưu nhất trong mọi bản đồ. Dù vậy lại phải đánh đổi bằng không gian tìm kiếm rất lớn (gần như là mọi điểm trên bản đồ nếu điểm kết thúc nằm ở xa với điểm bắt đầu)
- Thuật toán UCS có thời gian tìm kiếm và không gian tìm kiếm tối ưu tương đối lớn so với các thuật toán khác khi tìm kiếm trong các bản đồ phức tạp và có không gian lớn
- Thuật toán A\* khá ổn định khi luôn tìm được đường đi gần như là tối ưu nhất với mọi loại bản đồ. Bên cạnh đó, không gian và thời gian tìm kiếm luôn ở mức trung bình trong 4 thuật toán

# USING MATRIX WITH BONUS\_POINT

ARTIFICIAL INTELLIGENCE

# THUẬT TOÁN 1

---

Ý tưởng thuật toán như sau:

- Gọi điểm đang xét là curr. Ban đầu curr sẽ là điểm bắt đầu. Gọi path là danh sách các điểm của đường đi với phần tử đầu tiên là curr. Từ curr ta sẽ tìm các điểm có thể đến được theo 4 hướng: lên, xuống, trái, phải.
- Để chọn điểm tiếp theo (n), ta sẽ xét đến giá trị của  $f(n) = g(n) + h(n)$  với:
  - $g(n)$  là độ dài đường đi từ điểm curr đến điểm thưởng n
  - $h(n)$  là giá trị của điểm thưởng đó
- Từ các  $f(n)$  tính được từ các điểm thưởng, ta sẽ chọn  $f(n)$  nhỏ nhất. Lúc này, ta cần so sánh  $f(n)$  với 2 lần khoảng cách Euclid từ điểm đang xét đến điểm kết thúc. Nếu  $f(n) >$  khoảng cách thì lấy  $f(n)$  là khoảng cách. Ta sẽ có tối đa 4  $f(n)$ .
- Tiếp theo từ các  $f(n)$  đã có ta sẽ tiếp tục chọn n tương ứng với  $f(n)$  nhỏ nhất làm điểm curr tiếp theo. Lưu ý:
  - Nếu cả 4 hướng đều không khả thi, ta sẽ quay lui về giá trị trước đó trong path để tiếp tục xét. Đồng thời cũng xóa giá trị curr trong path
  - Nếu có ít nhất 2  $f(n)$  có giá trị nhỏ nhất bằng nhau thì ta sẽ xét độ dài đường đi BFS của điểm n với điểm kết thúc và điểm có khoảng cách nhỏ nhất sẽ làm curr tiếp theo
- Ta sẽ đánh dấu với mỗi curr được chọn và thêm curr vào path.
- Mỗi khi đi qua điểm thưởng :
  - Reset các giá trị được đánh dấu lại từ đầu. Đồng thời cũng xóa điểm thưởng khỏi danh sách điểm thưởng được xét để tính  $f(n)$
- Nếu curr là điểm kết thúc, kết thúc bài toán. Nếu không, tiếp tục đến các bước sau



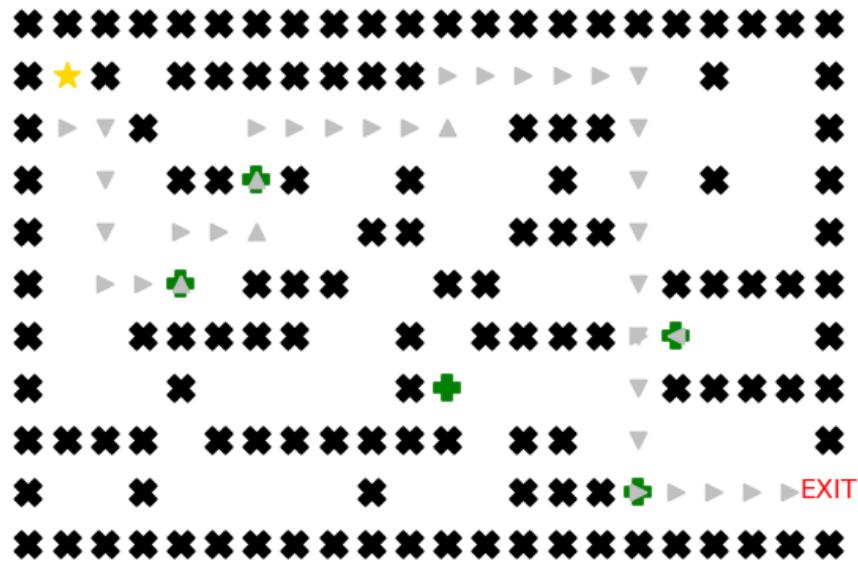
# MAP 1

2 BONUS POINTS:

(3,6) : -3

(9,16) : -1

Chi phí: 34



5 BONUS POINTS:

(5,4) : -4 ; (3,6): -3

(7,11) : -2; (6,17):-4

(9,16): -1

Chi phí: 22

10 BONUS POINTS:

(4, 5): -10 ; (3, 19) : -5

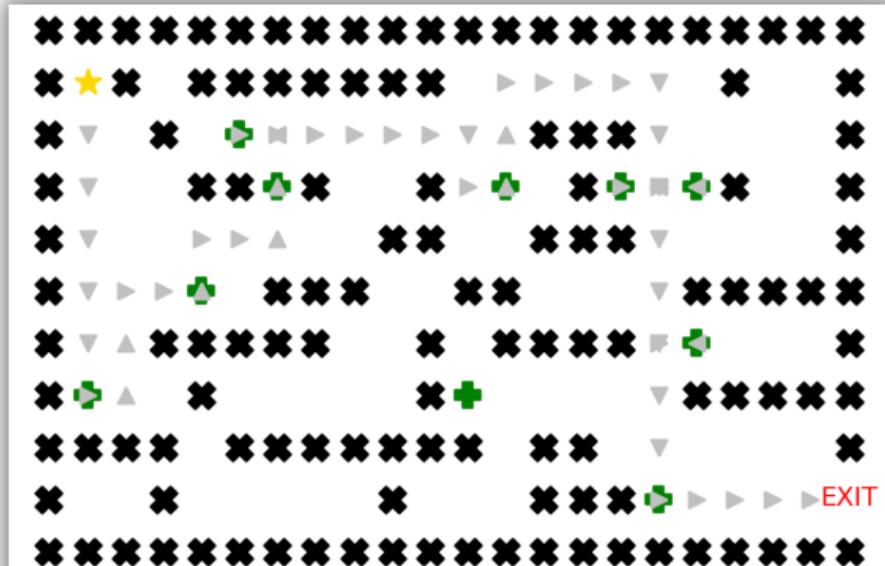
(2, 15): -2 ; (2, 9): -7

(9, 14): -4 ; (5, 18): -1

(1,2): -2 ; (1, 3): -6

(5, 4): -9 ; (7, 4): -8

Chi phí: 0



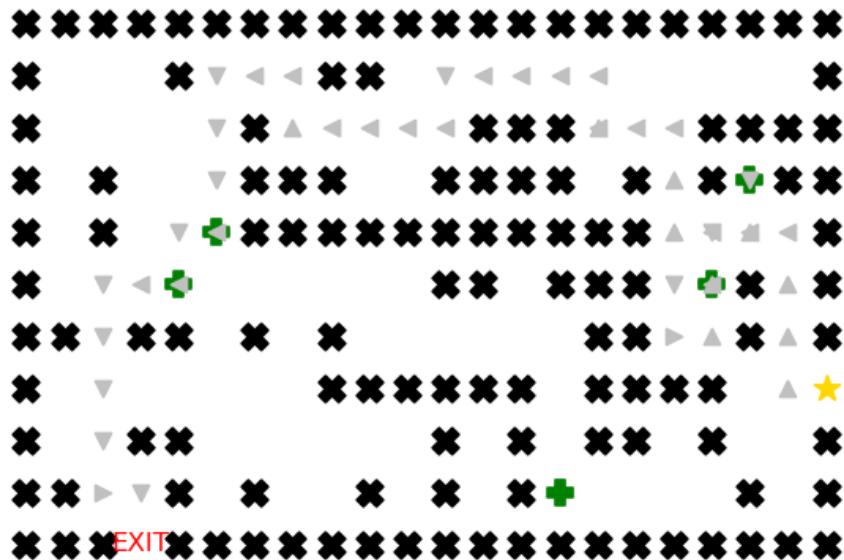
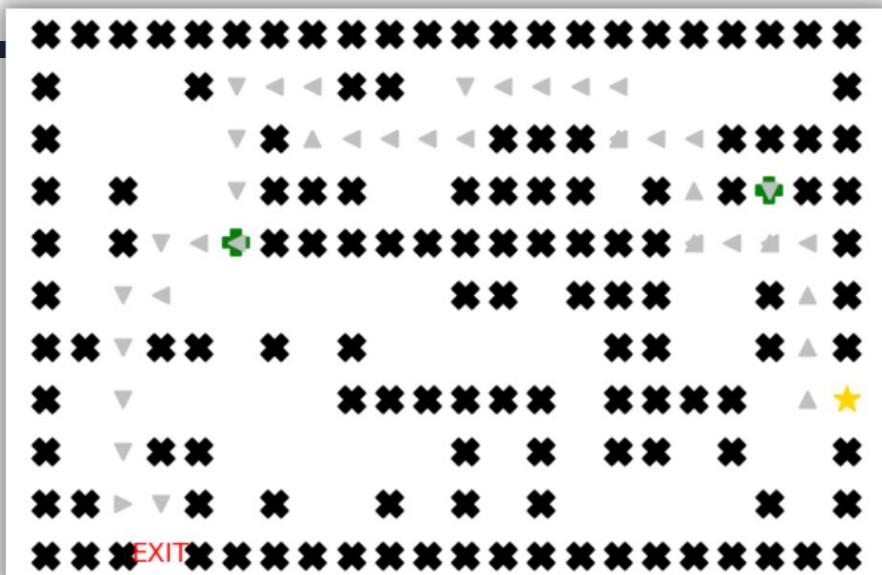
# MAP 2

**2 BONUS POINTS:**

(4,5) : -10

(3,19) : -5

**Chi phí: 24**



**5 BONUS POINTS:**

(4,5) : -10 ; (3,19) : -5

(9, 14): -4 ; (5, 18): -1

(5, 4) : -9

**Chi phí: 15**

**10 BONUS POINTS:**

(4, 5): -10 ; (3, 19) : -5

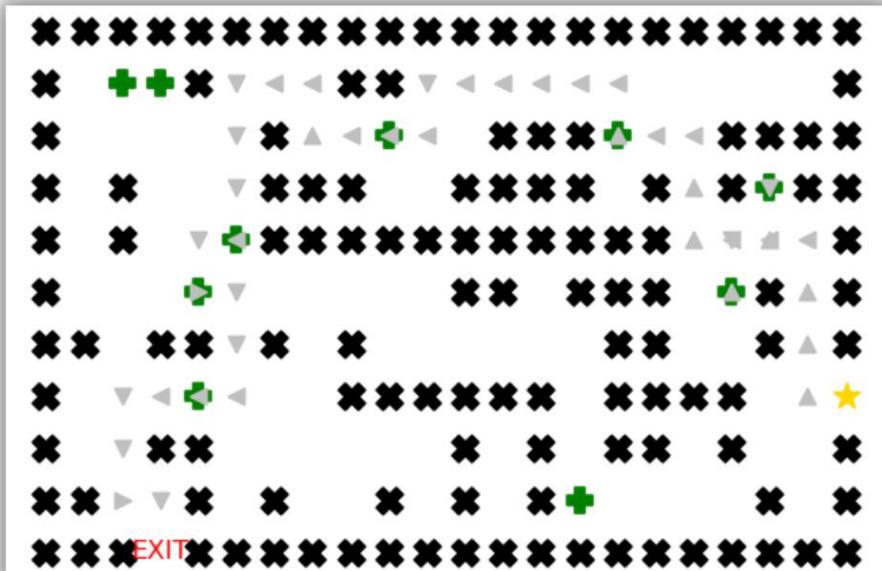
(2, 15): -2 ; (2, 9): -7

(9, 14): -4 ; (5, 18): -1

(1,2): -2 ; (1, 3): -3

(5, 4): -9 ; (7, 4): -8

**Chi phí: -6**



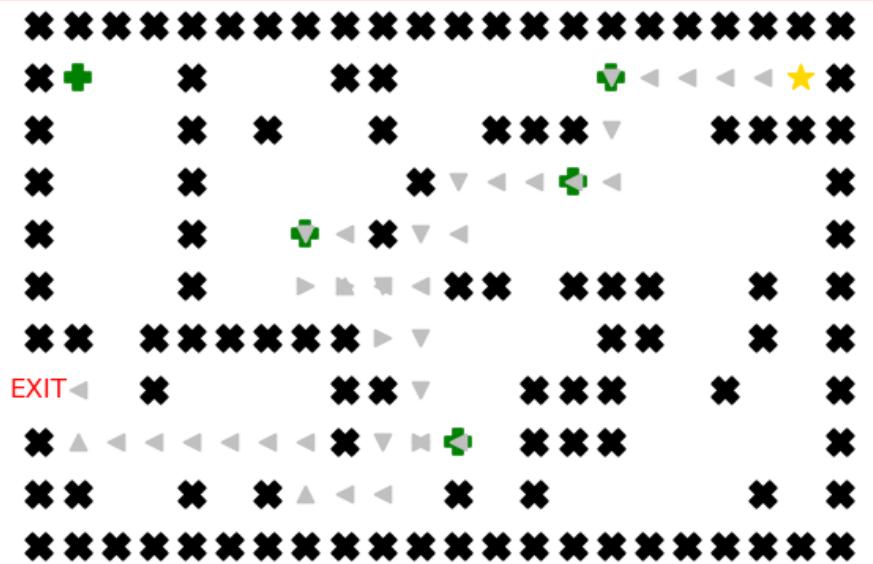
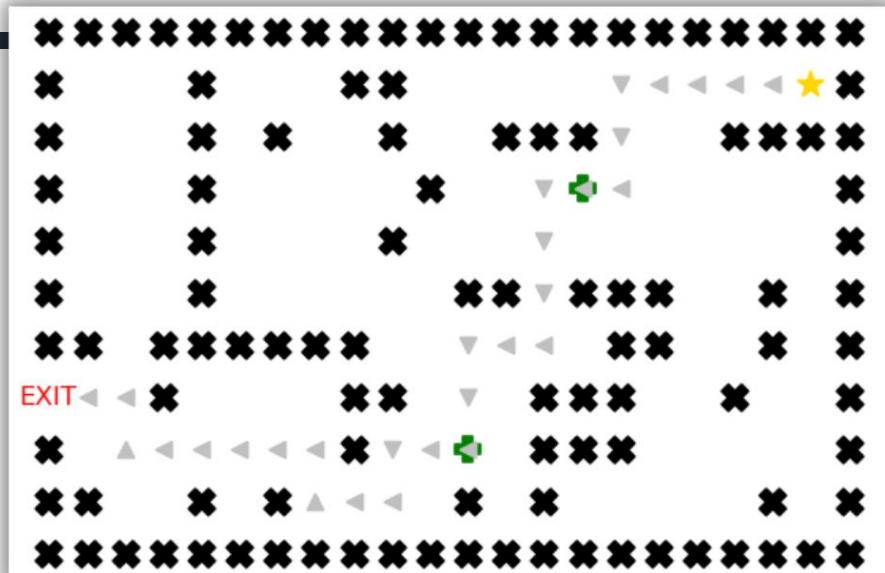
# MAP 3

**2 BONUS POINTS:**

(3,14) : -4

(8,11) : -2

**Chi phí: 22**



**5 BONUS POINTS:**

(3, 14): -4; (1, 15): -6

(1, 1): -1 ; (4, 7): -9

(8, 11): -2

**Chi phí: 15**

**10 BONUS POINTS:**

(3, 6) -3 ; (5, 17): -5

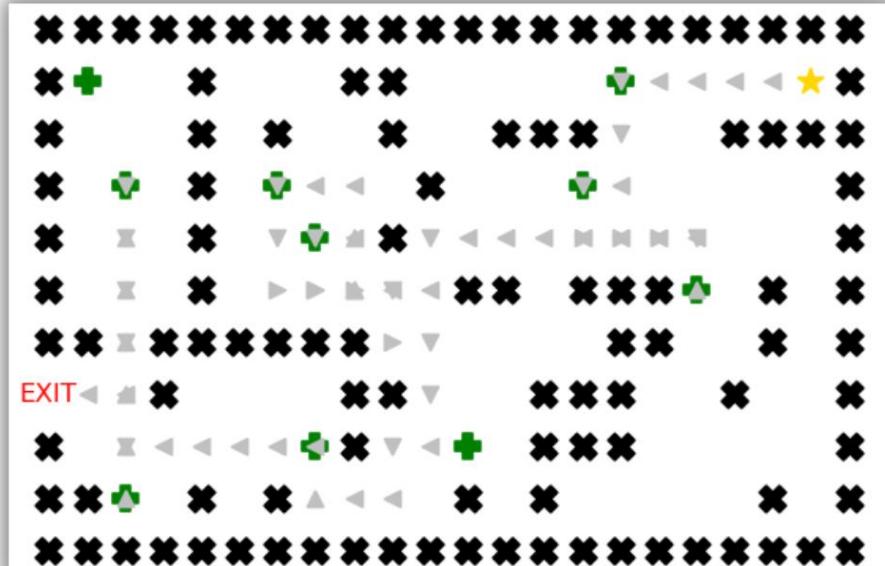
(3, 14): -4 ; (1, 15): -6

(1, 1): -1 ; (4, 7): -9

(8, 11): -2 ; (3, 2): -7

(8, 7): -8 ; (9, 2): -10

**Chi phí: -3**



# THUẬT TOÁN 2

---

Ta quy ước gọi các điểm đầu điểm cuối là start, end. Các điểm thưởng (bonus) là bonus\_point.

- Mục tiêu chủ đạo của thuật toán này là kế thừa ý tưởng tuyệt vời mà thuật toán A\* để lại. Vì thế ta xét 3 điểm trong không gian lần lượt là A, B, C với điều kiện có thể tạo thành một tam giác hoặc một đường thẳng để ta có thể làm việc trên đó.
- Với việc quy ước như vậy ta sẽ có một quy tắc luôn luôn tìm được đường ngắn nhất đi từ A tới C như sau:
  - Gọi  $G(n)$  là chi phí đi từ A --> B
  - Gọi  $H(n)$  là chi phí đi từ B --> C
- Như vậy chi phí đi từ A --> C sẽ có 2 cách tính là đi thẳng từ A--> C hoặc là đi từ A --> B rồi sau đó đi từ B --> C. Áp dụng định lý này trong trường hợp bản đồ có điểm thưởng ta sẽ có một đường đi với chi phí hoàn hảo.
- Với một ý tưởng như vậy việc triển khai bắt đầu từ việc tính chi phí cho việc đi tới mỗi cạnh bằng cách áp dụng thuật toán A\* ở phần trước. Sau đó với điểm đầu tiên là start ta sẽ quyết định xem đi các bonus\_point hay là đi thẳng tới end. Rồi tiếp đó sau khi đã đi tới một điểm bonus\_point chỉ định ta lại tiếp tục chọn đường đi như thế đến khi ta tới được end.



# MAP 1

## 01. 2 bonus points:

(3,6) : -3

(9,16) : -1

Chi phí: 28



## 02. 5 bonus points:

(5,4) : -4 ; (3,6) : -3

(7,11) : -2; (6,17) : -4

(9,16) : -1

Chi phí: 22

(7,1) : -7 ; (5,4) : -4

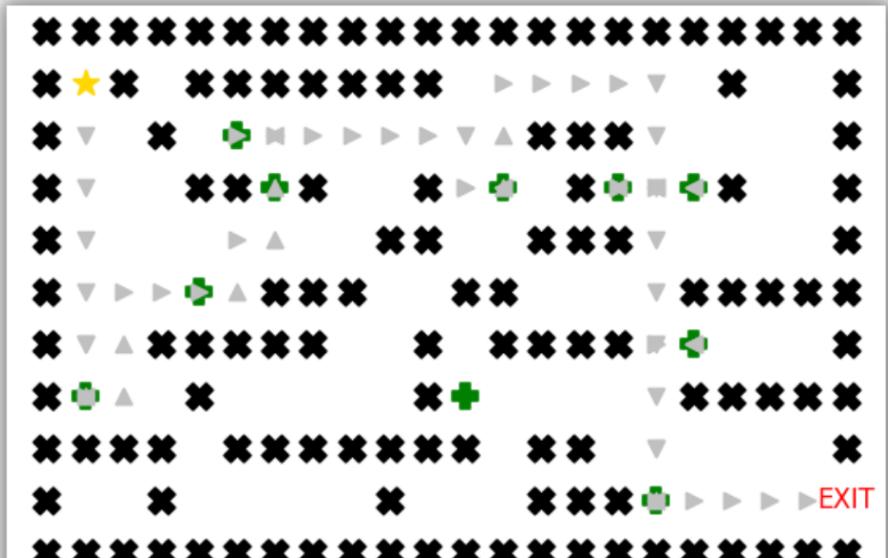
(3,6) : -2 ; (2,5) : -6

(3,12) : -8 ; (7,11) : -5

(3,17) : -10 ; (3,15) : -9

(6,17) : -3 ; (8,15) : -1

Chi phí: -3



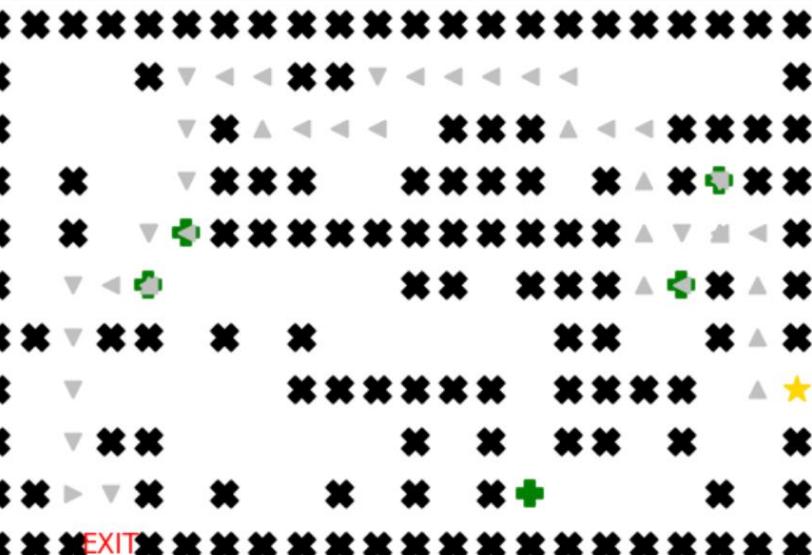
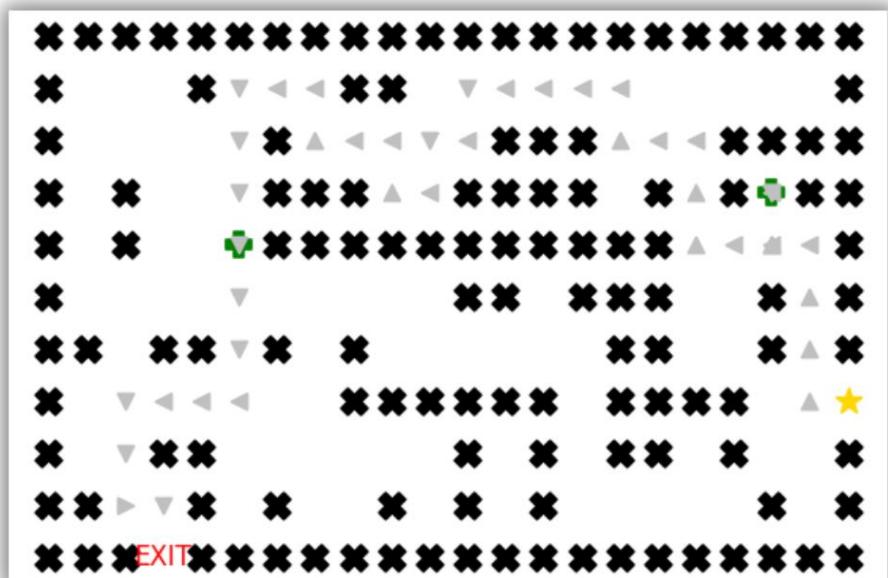
# MAP 2

## 01. 2 bonus points:

(4,5) : -10

(3,19) : -5

Chi phí: 27



## 03. 10 bonus points:

(4, 5) : -10 ; (3, 19) : -5

(2, 15) : -2 ; (2, 9) : -7

(9, 14) : -4 ; (5, 18) : -1

(1, 2) : -2 ; (1, 3) : -3

(5, 4) : -9 ; (7, 4) : -8

Chi phí: -6



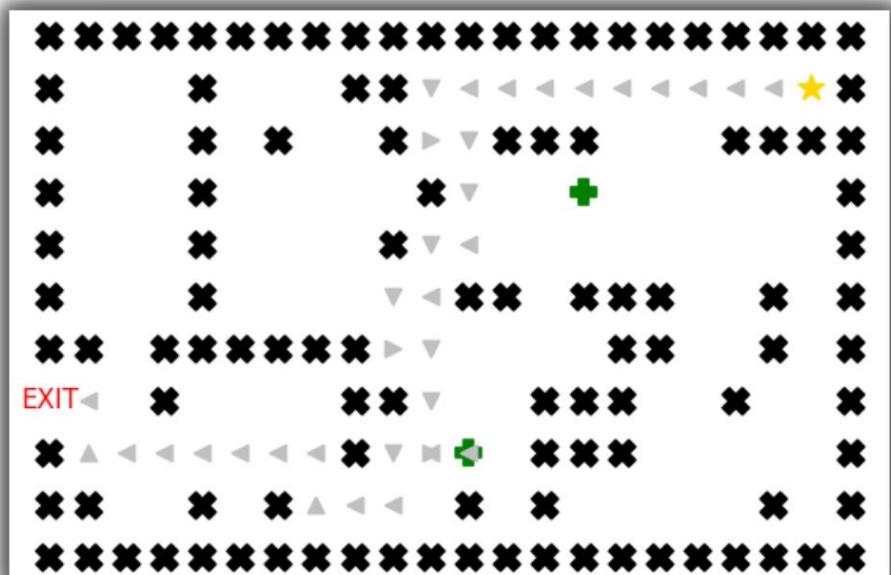
# MAP 3

## 01. 2 bonus points:

(3,14) : -4

(8,11) : -2

Chi phí: 32



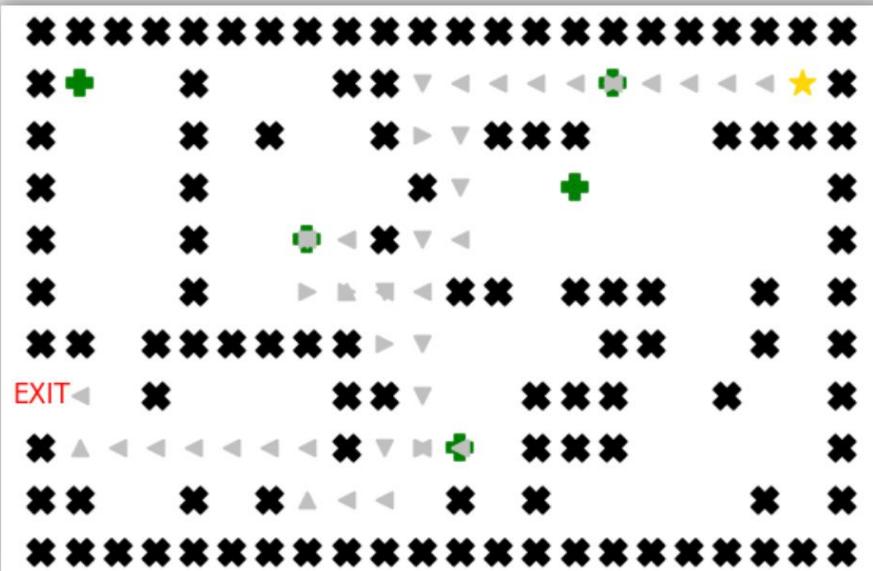
## 02. 5 bonus points:

(3, 14): -4; (1, 15): -6

(1, 1): -1 ; (4, 7): -9

(8, 11): -2

Chi phí: 19



## 03. 10 bonus points:

(3, 6) -3 ; (5, 17): -5

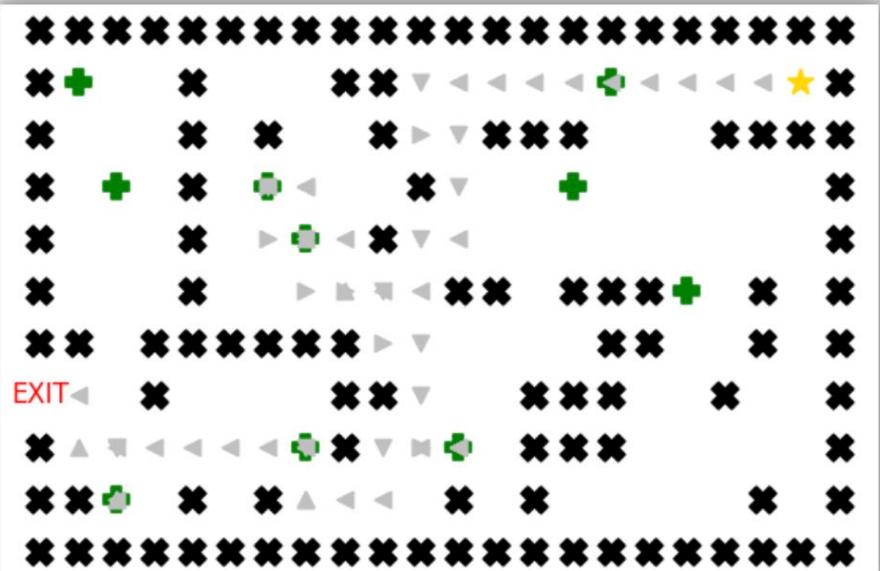
(3, 14): -4 ; (1, 15): -6

(1, 1): -1 ; (4, 7): -9

(8, 11): -2 ; (3, 2): -7

(8, 7): -8 ; (9, 2): -10

Chi phí: -1



# NGUỒN THAM KHẢO

## YOUTUBE

<https://youtu.be/-igjC-VoHas>

## SLIDE THẦY BẮC

## WIKIPEDIA

[https://fr.wikipedia.org/wiki/Algorithm\\_A%27](https://fr.wikipedia.org/wiki/Algorithm_A%27)

## GEEKFORGEEKS

<https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/>

## LABS.FLINTERS

<https://labs.flinters.vn/algorithms/cac-thuat-toan-tim-kiem-trong-ai/>

## STANFORD

<http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html>



THAT'S ALL

---

THANK YOU  
FOR READING

---

ARTIFICIAL INTELLIGENCE