ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO ĐỒ ÁN THỰC HÀNH

MÔN CƠ SỞ TRÍ TUỆ NHÂN TẠO

ĐỒ ÁN 1: CÁC THUẬT TOÁN TÌM KIẾM

Lớp 20_23

Nguyễn Thị Hồng Nhung MSSV: 20120093

Trần Thái San MSSV: 20120177

Trần Minh Nhựt MSSV: 20120343

GVLT: GS.TS Lê Hoài Bắc

Trợ giảng: Nguyễn Duy Khánh

Nguyễn Ngọc Băng Tâm

Thành phố Hồ Chí Minh – 10/2022

MỤC LỤC

I. ĐÁNH GIÁ ĐỒ ÁN:

Bảng phân công công việc:

STT	MSSV	Họ tên	Công việc	% Hoàn thành
1	20120343	Trần Minh Nhựt	Cài đặt BFS và bản đồ có điểm thưởng	100%
2	20120093	Nguyễn Thị Hồng Nhung	Cài đặt GBFS và DFS	100%
3	20120177	Trần Thái San	Cài đặt UCS, AStar	100%

Đánh giá mức độ hoàn thành đồ án:

- Đối với bài toán bản đồ không có điểm thưởng: Hoàn thành 5 giải thuật bao gồm BFS, DFS, UCS, GBFS, A Star.
- Đối với bản đồ có điểm thưởng: Đề xuất chiến lược tìm đường đi tối ưu.
- Kịch bản nâng cấp: Chưa hoàn thành.

II. BẢN ĐỒ KHÔNG CÓ ĐIỂM THƯỞNG:

***Lưu ý:** Khi chạy thực thi thuật toán sẽ có một cửa sổ Python Turtle Graphics mở ra, phóng to cửa sổ lên để nhìn rõ được đường đi của từng thuật toán theo thứ tự là UCS, A* xài heuristic 1 và heuristic 2. Giữ cửa sổ Python Turtle Graphics trong suốt quá trình thực thi để thuận tiện cho việc Output. Sau khi chạy xong, tắt cửa sổ Turtle Graphics để thực thi tiếp

1. BFS (Breadth First Search):

Ý tưởng thuật toán:

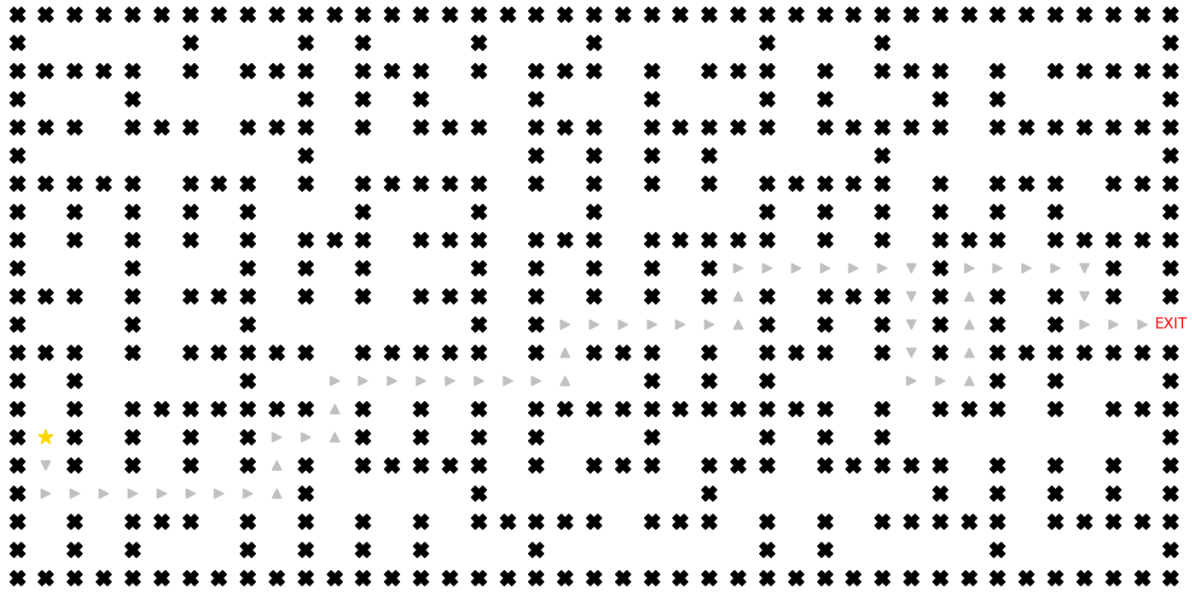
Tìm kiếm theo chiều rộng (BFS) là một thuật toán để tìm kiếm cấu trúc dữ liệu dạng cây cho một nút thỏa mãn một thuộc tính nhất định. Nó bắt đầu từ gốc cây và đi đến tất cả các nút ở độ sâu hiện tại trước khi chuyển sang các nút ở mức độ sâu tiếp theo. Bộ nhớ bổ sung, thường là một hàng đợi, là cần thiết để theo dõi các nút con đã gặp nhưng chưa được khám phá.

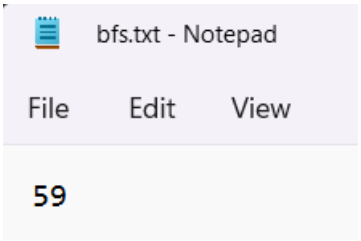
Mã giả:

```
BFS(v: Vertex)
    q = a new empty queue
    q.enqueue(v)
    Mark vas visited
    while (q is not empty){
        w= q.dequeue()
        for (each unvisited vertex u adjacent to w){
            Mark was visited
            q.enqueue(u)
        }
    }
```

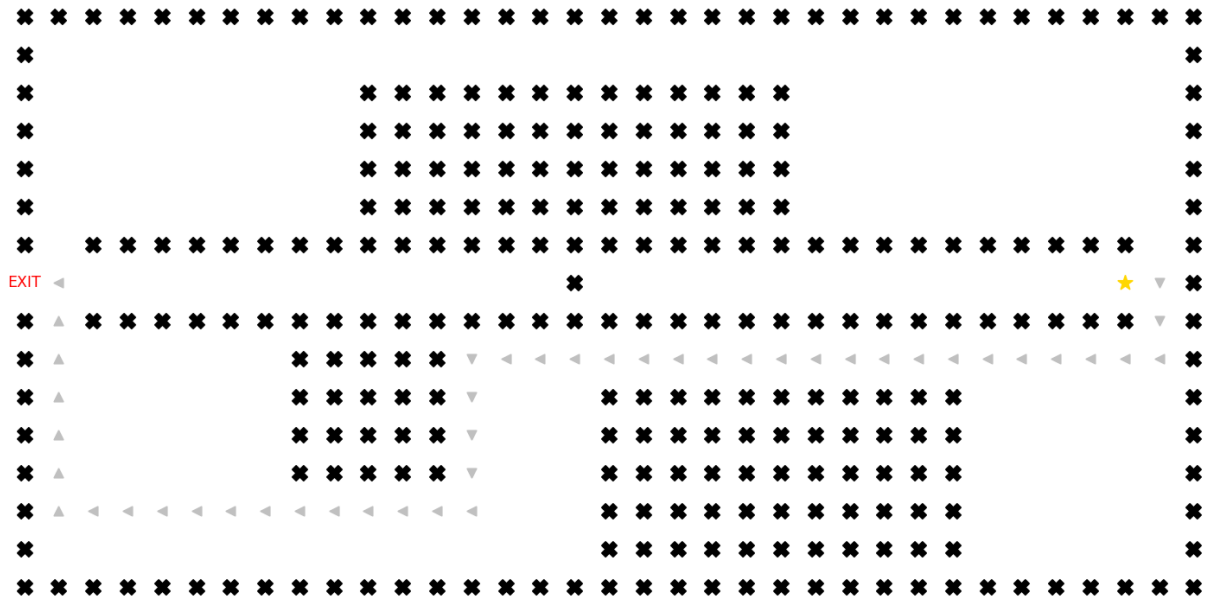
Kết quả chạy thuật toán:

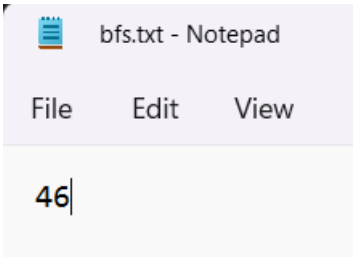
Maze 1:



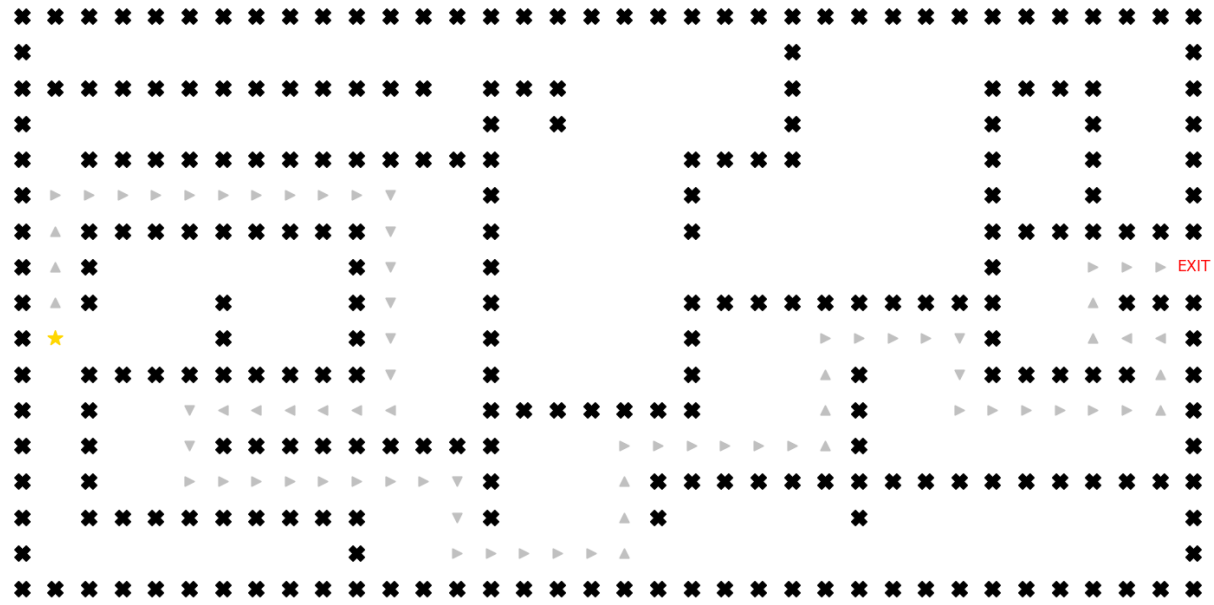


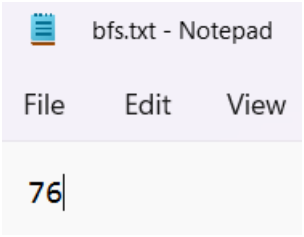
Maze 2:



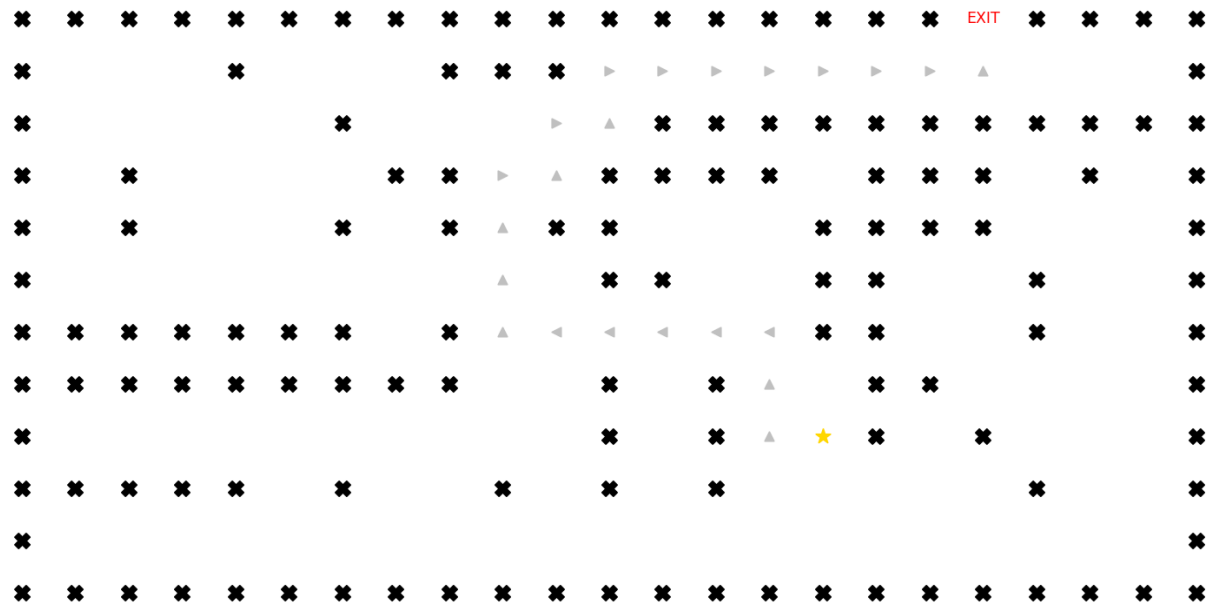


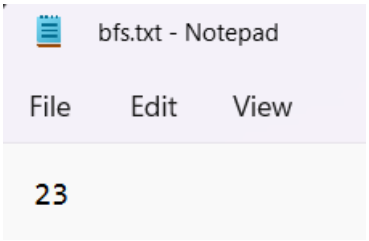
Maze 3:



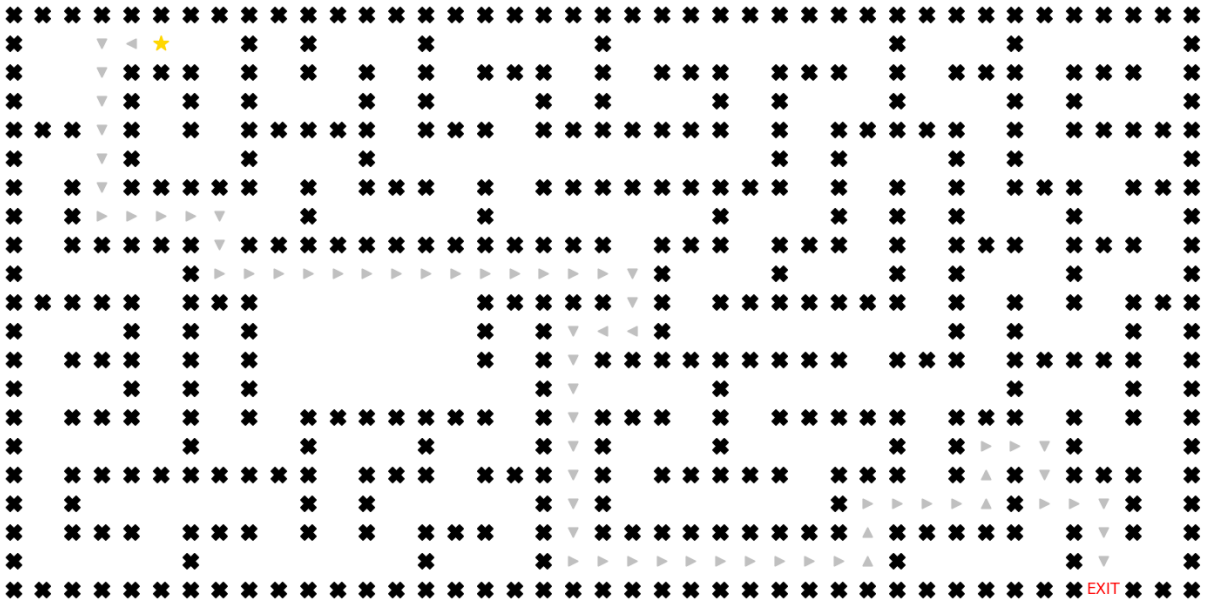

bfs.txt - Notepad
File
Edit
View
76

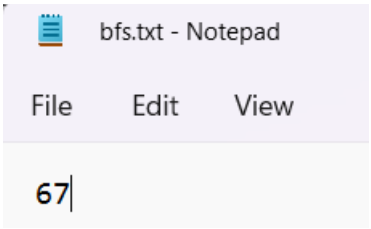
Maze 4:


EXIT


bfs.txt - Notepad
File
Edit
View
23

Maze 5:





Đánh giá thuật toán:

Thuật toán BFS là tìm kiếm mù theo chiều ngang dẫn đến việc có thể phải duyệt nhiều nút, nhiều nhánh mà không có khả năng nằm trong đường đi tối ưu.

Trường hợp xấu, thuật toán sẽ duyệt toàn bộ bản đồ.

2. **DFS(Depth First Search):**

Ý tưởng thuật toán:

Dùng DFS để tìm đường đi từ node bắt đầu đến node kết thúc trên maze.

Từ node bắt đầu, ta đi qua sâu vào từng nhánh. Khi đã duyệt hết nhánh của node, quay lại duyệt node tiếp theo.

Sử dụng cấu trúc STACK để lưu lại các node lân cận chưa được khám phá.

Thuật toán dừng khi tìm được node kết thúc và đưa ra đường đi từ node bắt đầu đến node kết thúc.

Cách cài đặt thuật toán:

Input: matrix: maze

Start: vị trí điểm bắt đầu

End: vị trí điểm kết thúc

Output:

File .jpg : trả về solution của maze là road

File .txt: trả về cost của road

Cài đặt:

Frontier là mảng các node lân cận dự định sẽ mở. Dùng cấu trúc STACK để lưu trữ các frontier node

Explored là mảng các node đã đi qua.

Direction là mảng để tạo hướng dịch chuyển của node đang xét.

Curren là node đang được xét.

Parent là dictionary để định nghĩa các parent node của child node

Chạy vòng lặp để add các frontier node của node đang xét vào Frontier và kiểm tra:

- Nếu không có bất kì frontier node nào thì trả về kết quả không tìm được đường đi và in ra chi phí là NO.
- Kiểm tra node có phải end node hay không. Nếu true thì set_parent cho end node và return , nếu false thì add các frontier node lân cận vào Stack.
- Đưa current node vào Explored và current node lúc này sẽ là phần tử được lấy từ stack frontier ra.

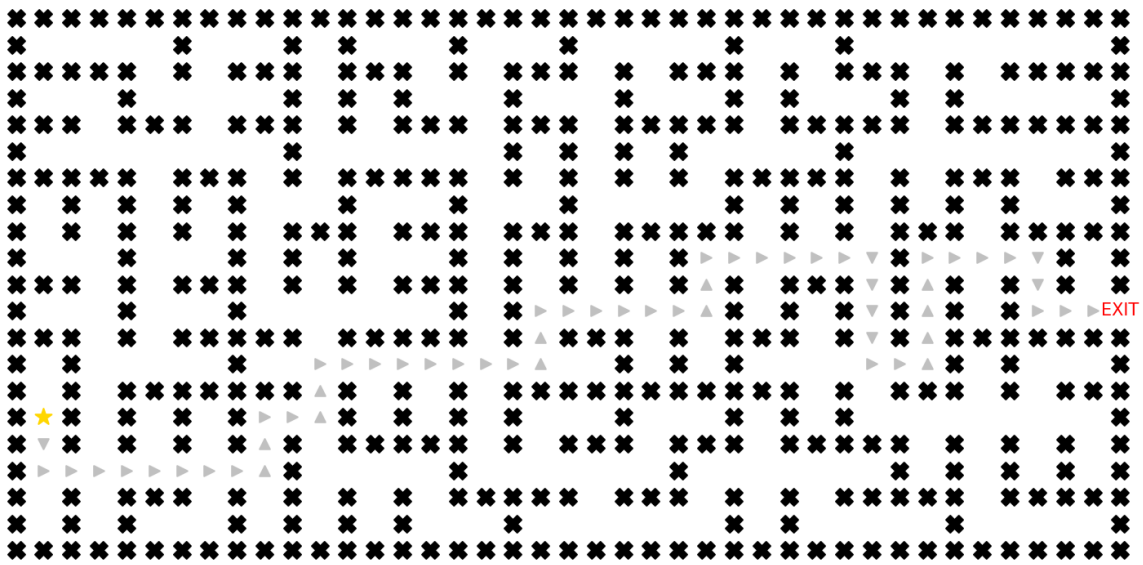
Đưa current node vào Explored

Set_parent cho end node là current node.

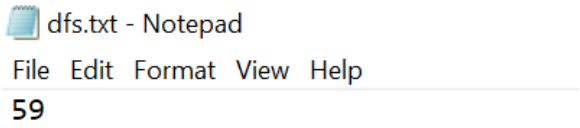
Trả ra solution của bài toán là road và cost. Mỗi 1 bước dịch chuyển thì sẽ tốn 1 đơn vị chi phí.

Ouput của thuật toán DFS:

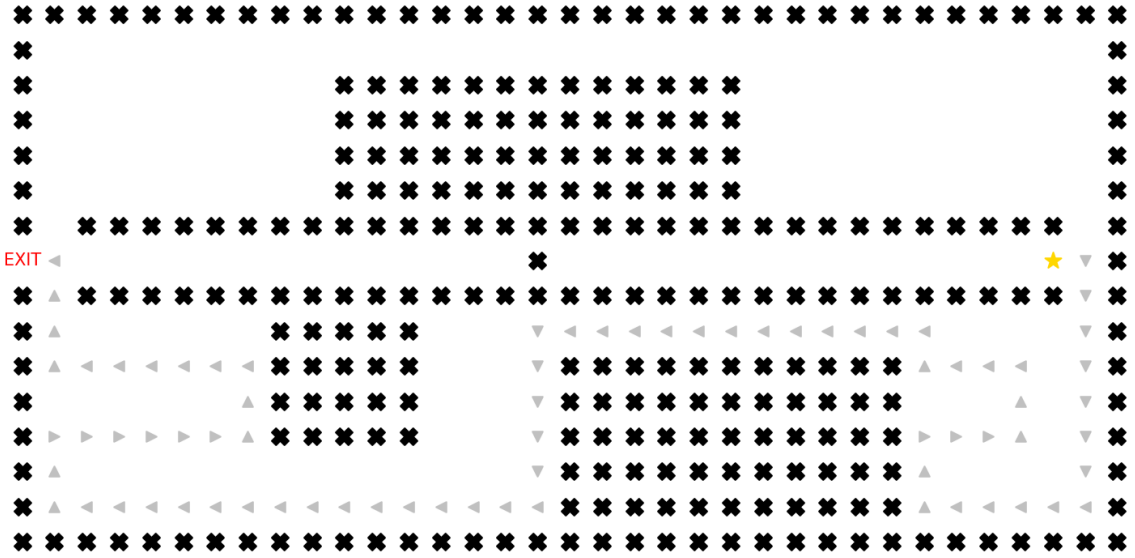
Output maze 1:



Cost maze:1



Output maze 2:



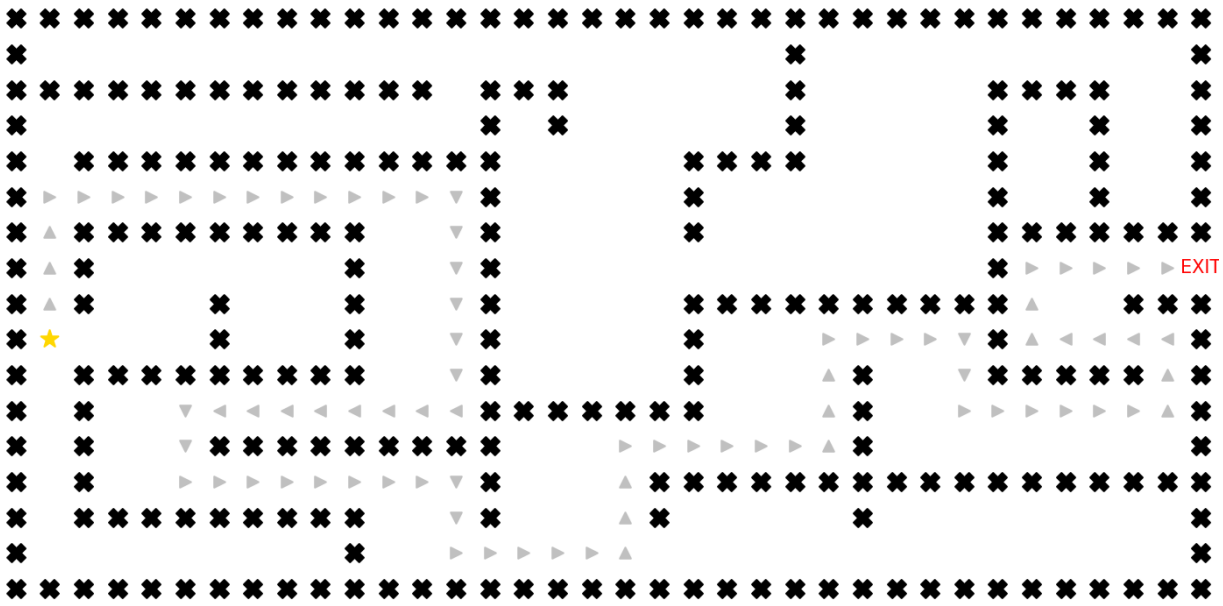
Cost maze 2:

dfs.txt - Notepad

File Edit Format View Help

76

Output maze 3:



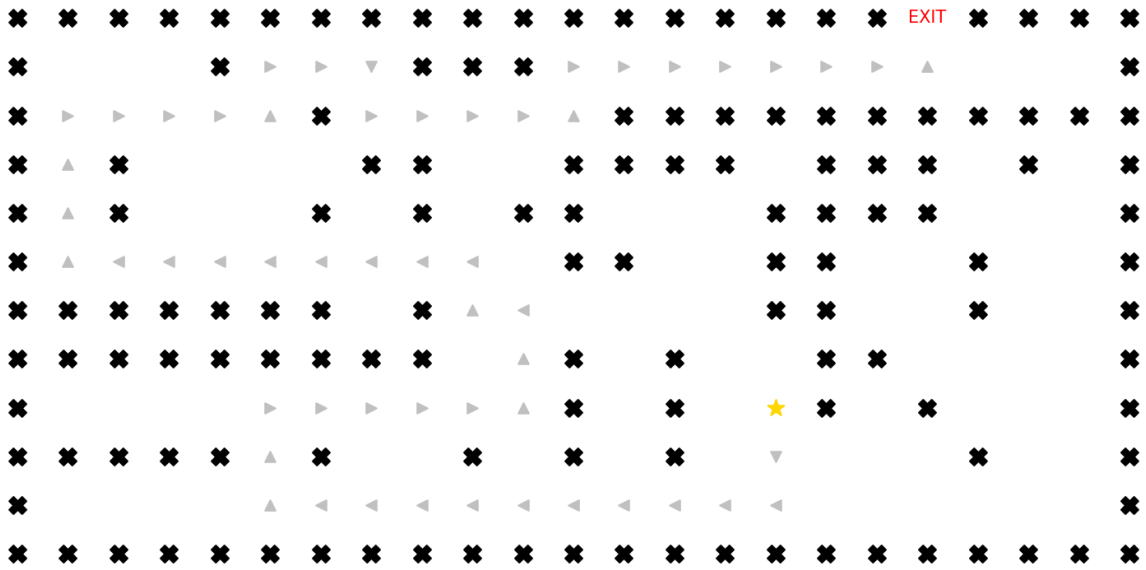
Cost maze 3:

dfs.txt - Notepad

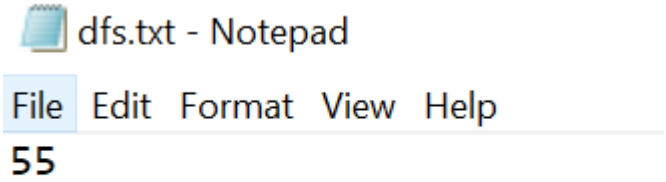
File Edit Format View Help

84

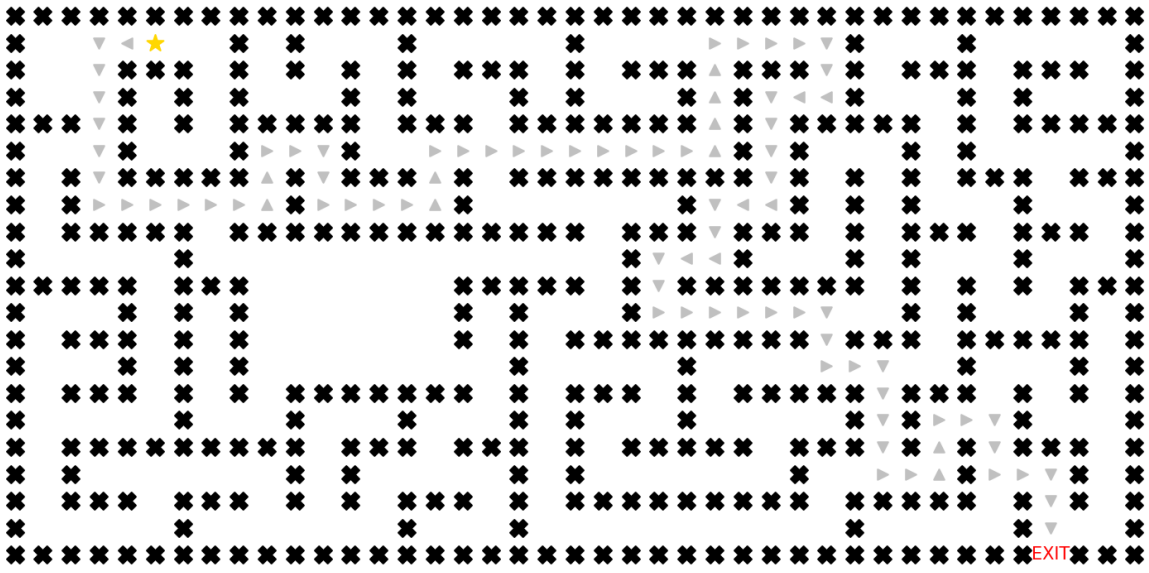
Output maze 4:



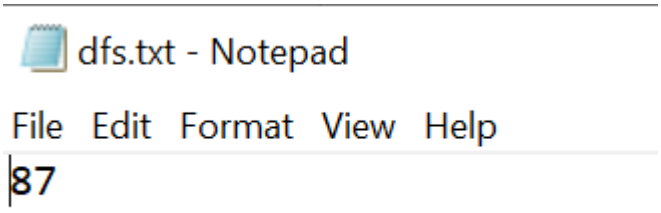
Cost maze 4:



Output maze 5:



Cost maze 5:



Đánh giá thuật toán DFS:

Tính đầy đủ: Có (nếu không gian hữu hạn)

Tính tối ưu: lời giải "phải" nhất

Độ mở các nút: ưu tiên mở các node sâu

Không gian: O(bm) kích thước không gian tuyến tính!

Thời gian : $O(b^m)$

Tệ nếu m lớn hơn nhiều so với b.

3. **UCS (Uniform- Cost Search)**

Ở thuật toán này do em xài Turtle Graphics để cài đặt, nên để chạy thuật toán cần cài thêm một số thư viện hỗ trợ như pyautogui để hỗ trợ

Cách cài : sử dụng lệnh pip install pyautogui

Ý tưởng thuật toán

Ý tưởng của thuật toán UCS gần giống như BFS nhưng trong thuật toán UCS có sử dụng hàng đợi ưu tiên(Priority Queue) thay cho hàng đợi (Queue). Do đó các node sẽ có thêm chi phí đường đi dẫn tới nó.

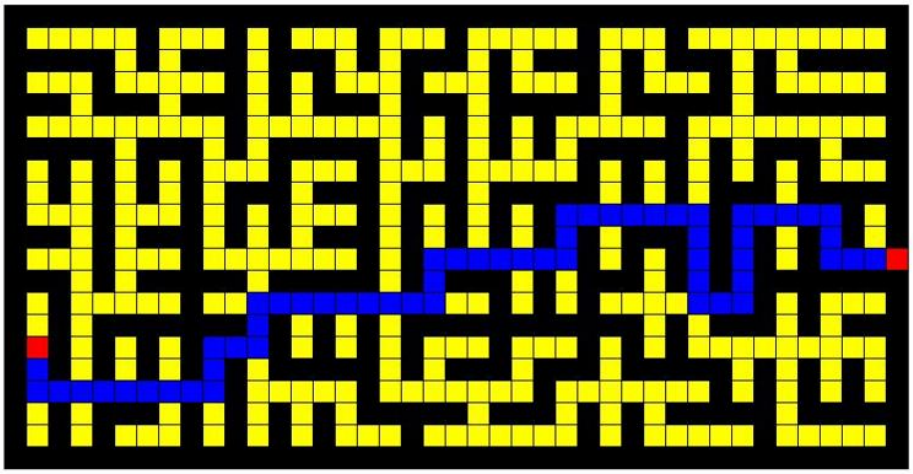
Thuật toán này có điều kiện dừng khi chiều dài của hàng đợi bằng 0.

Cài đặt thuật toán này tương tự với cài đặt BFS nhưng thay hàng đợi Queue bằng hàng đợi ưu tiên Priority Queue, và cài thêm hàm CanMove() để xác định các bước có thể di chuyển của thuật toán.

Kết quả chạy thuật toán:

Ở đây thuật toán in ra bản đồ thực hiện đường đi và chi phí thực hiện tìm kiếm đường đi.

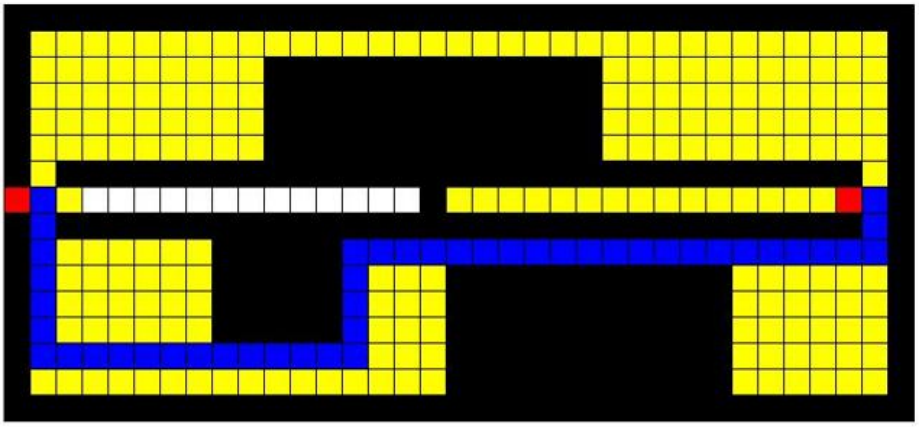
Maze 1:



File Edit Format View Help

THE PATH COST IS: 60

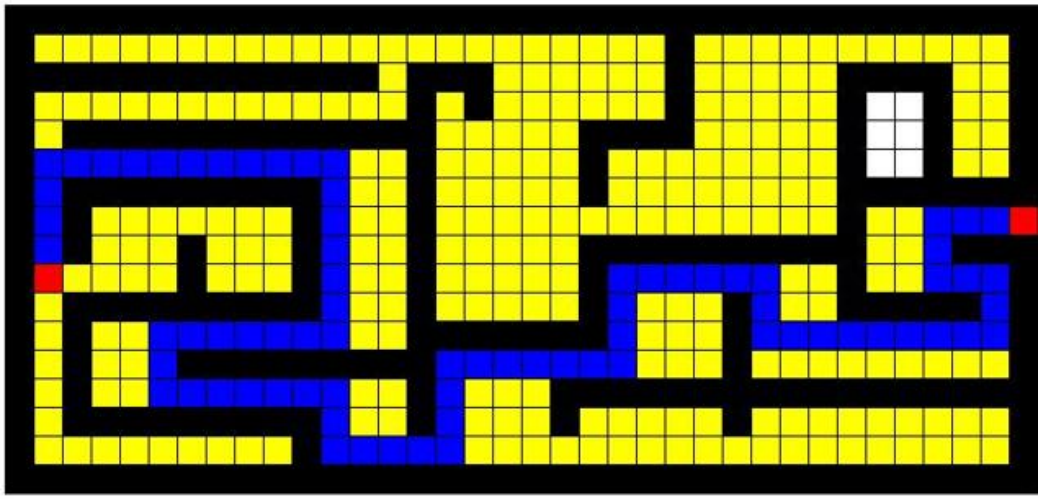
Maze 2:



File Edit Format View Help

THE PATH COST IS: 47

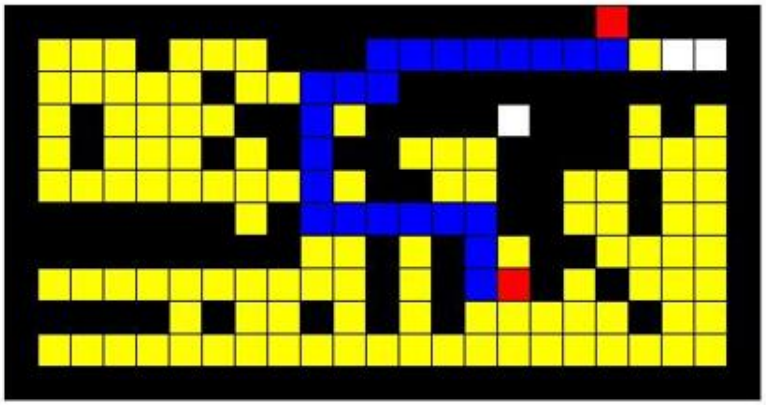
Maze 3:



File Edit Format View Help

THE PATH COST IS: 77

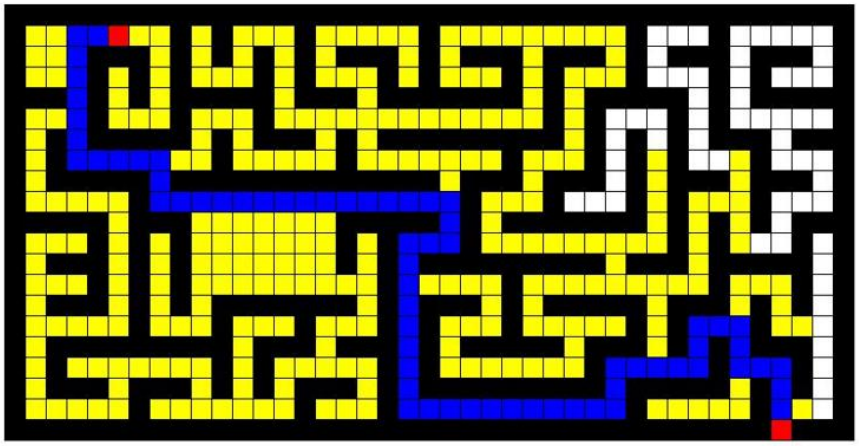
Maze 4:



File Edit Format View Help

THE PATH COST IS: 24

Maze 5:



File Edit Format View Help

THE PATH COST IS: 68

Đánh giá thuật toán UCS:

Với chi phí di chuyển thấp nhất là €, C ∗ là chi phí lời giải tối ưu

Tính đầy đủ: Có

Tính tối ưu: có

Không gian: $O\left(b^{1+\lfloor C^*/\epsilon \rfloor}\right)$

Thời gian: $O\left(b^{1+\lfloor C^*/\epsilon \rfloor}\right)$

4. **GBFS (Greedy Breadth First Search):**

Ý tưởng thuật toán:

Tương tự như thuật toán BFS, GBFS cũng được cài đặt dựa trên Queue. Tuy nhiên, các phần tử trong frontier trước khi lấy ra sẽ được sort lại theo các hàm heuristic được định nghĩa để tìm được đường đi tối ưu nhất.

Cài đặt thuật toán:

Ở phần GBFS, được cài đặt như DFS, nhưng để chuyển về thuật toán BFS, ta sẽ dùng Queue để cài cho frontier.

Cài đặt thêm hàm sort dựa trên heuristic để tìm được node tiếp theo để đến các end node là ngắn nhất.

Cài đặt heuristic 1:

Dựa trên Manhattan Distance với việc current node có thể đi về 4 phía (up, down, left, right). Ở hàm heuristic này, ta trả về giá trị heuristic của 1 node là tổng chiều dài của hình chiếu của đường thẳng nối hai điểm này trong hệ trục tọa độ Descartes.

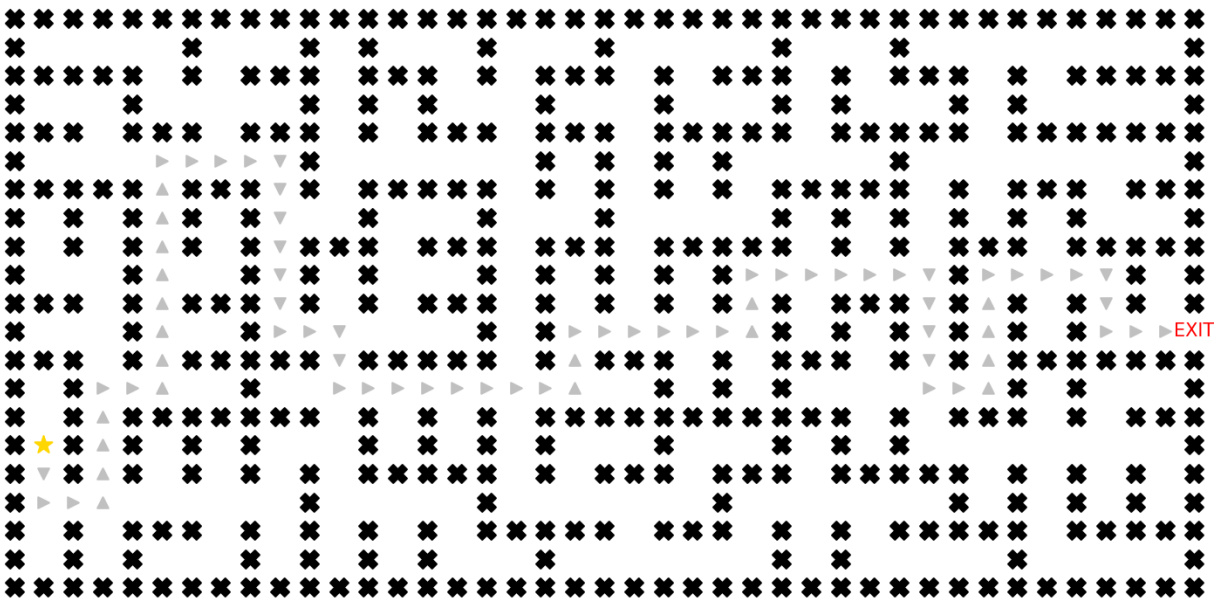
Cài đặt heuristic 2:

Hàm được cài đặt dựa trên Euclidean Distance. Ở hàm heuristic này, ta trả về giá trị heuristic của 1 node là khoảng cách của node đó đến end node theo đường chim bay.

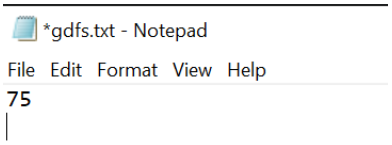
Và các frontier sẽ được lấy ra từ Queue sau khi Queue đã được sort theo các hàm heuristic.

Kết quả chạy thuật toán:

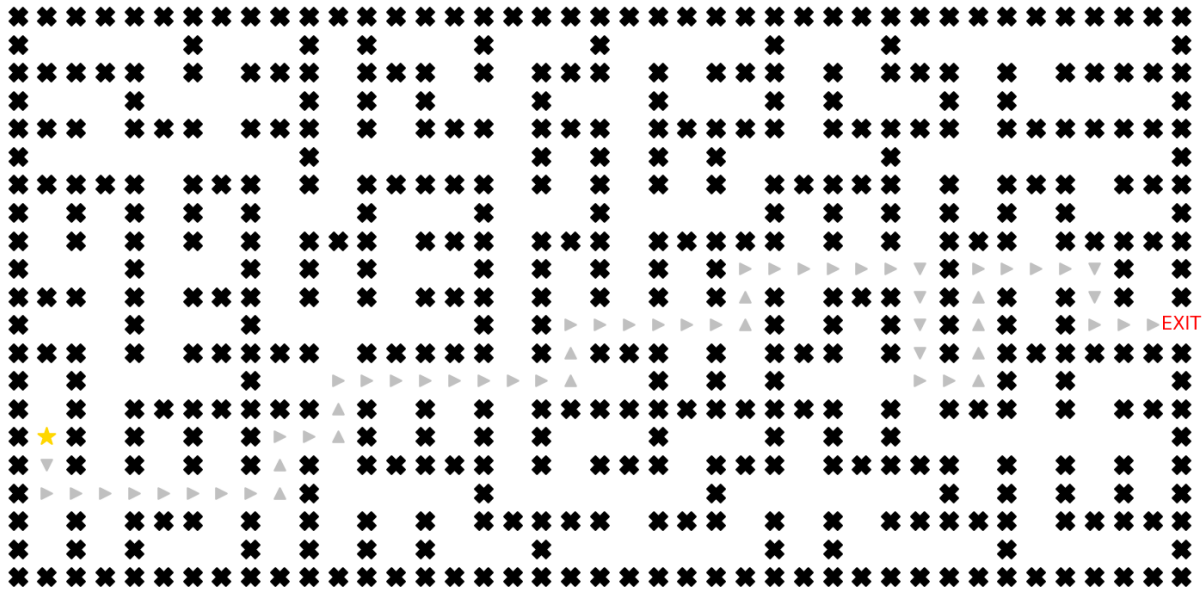
Output maze 1 – heuristic 1:



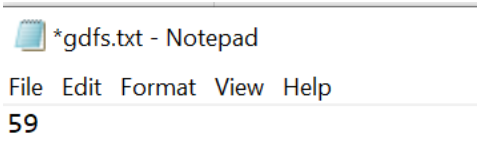
Cost maze 1- heuristic 1:



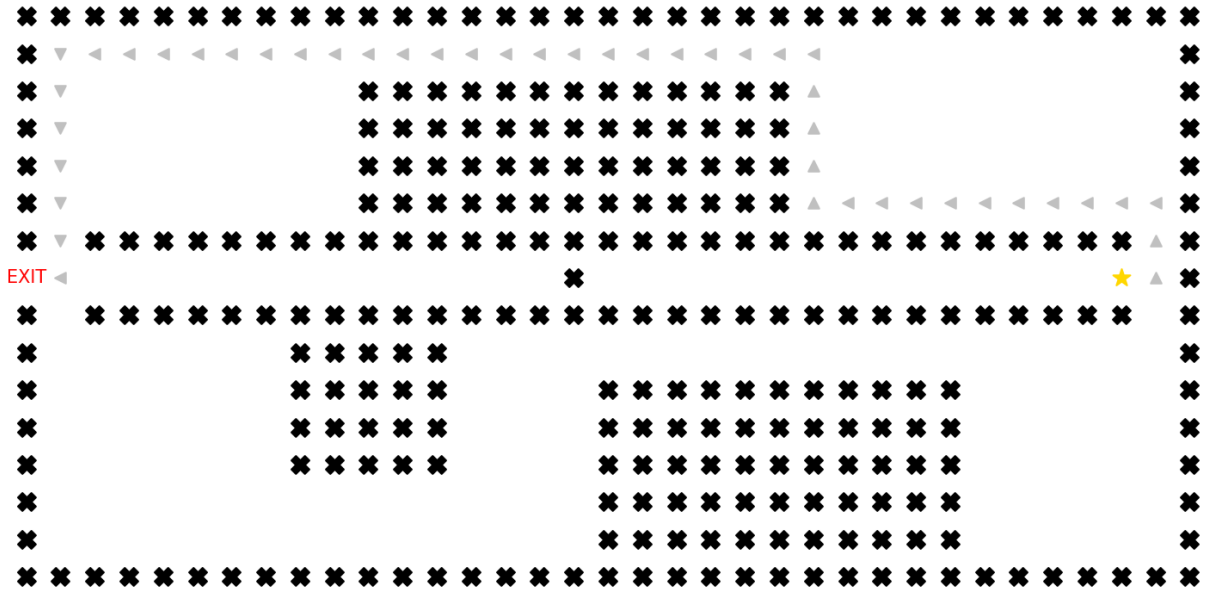
Output maze 1 - heuristic 2:



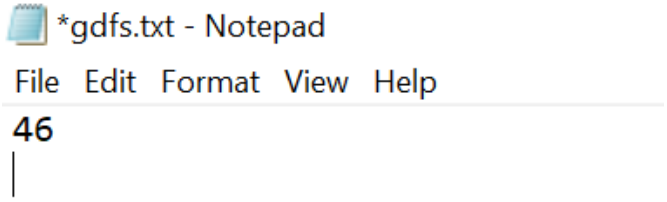
Cost maze 1- heuristic 2:



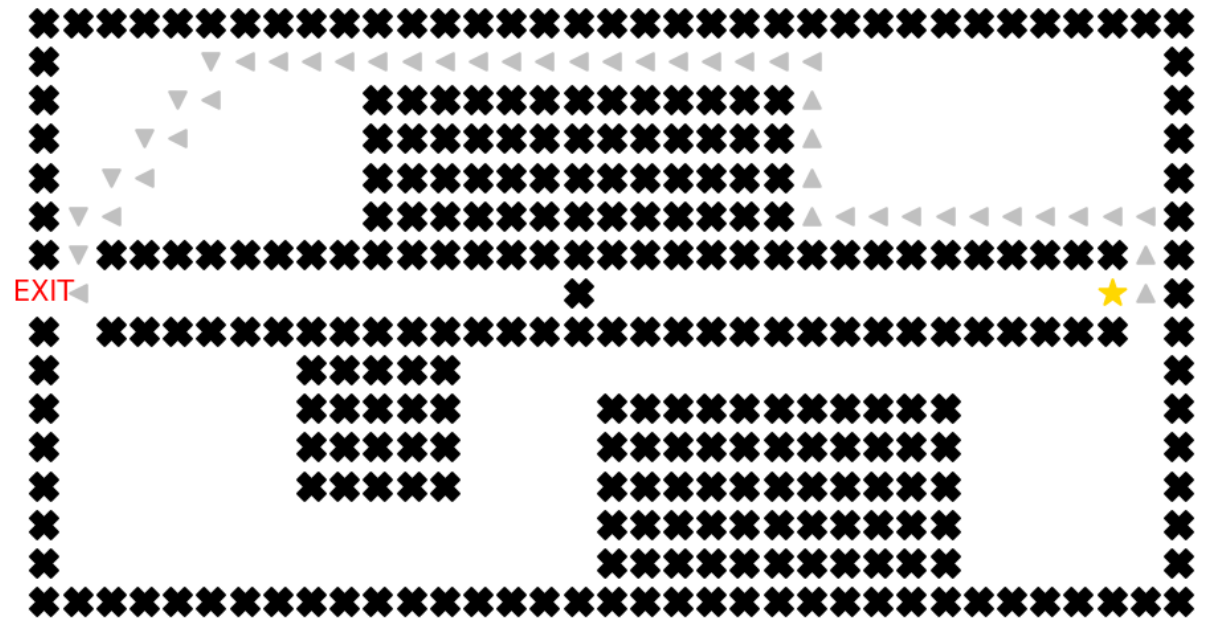
Output maze 2 – heuristic 1:



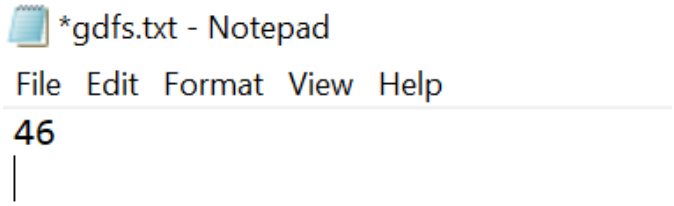
Cost maze 2-heuristic 1:



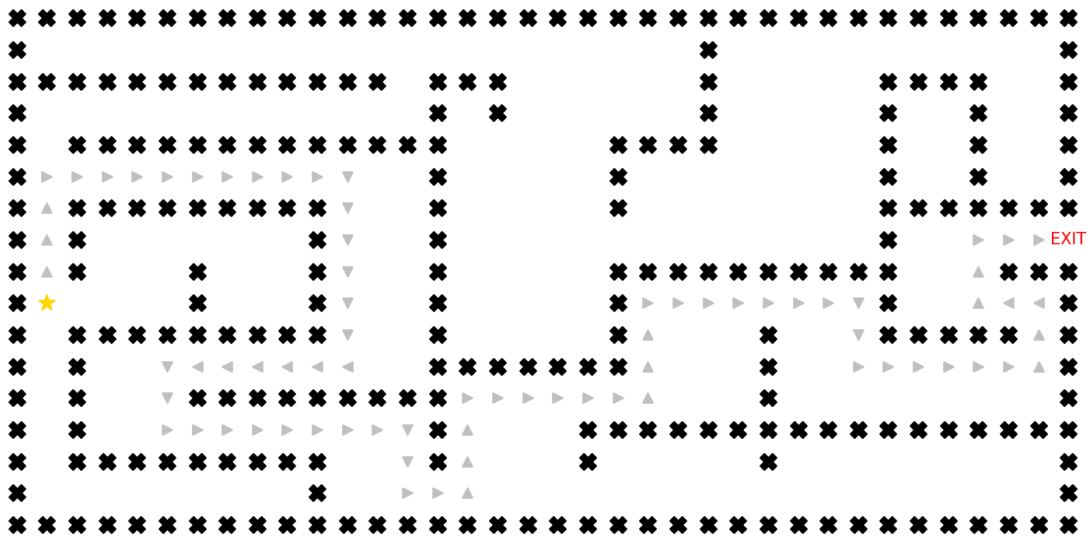
Output maze 2- heuristic 2:



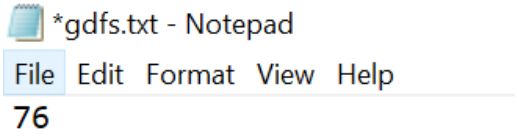
Cost maze 2 – heuristic 2:



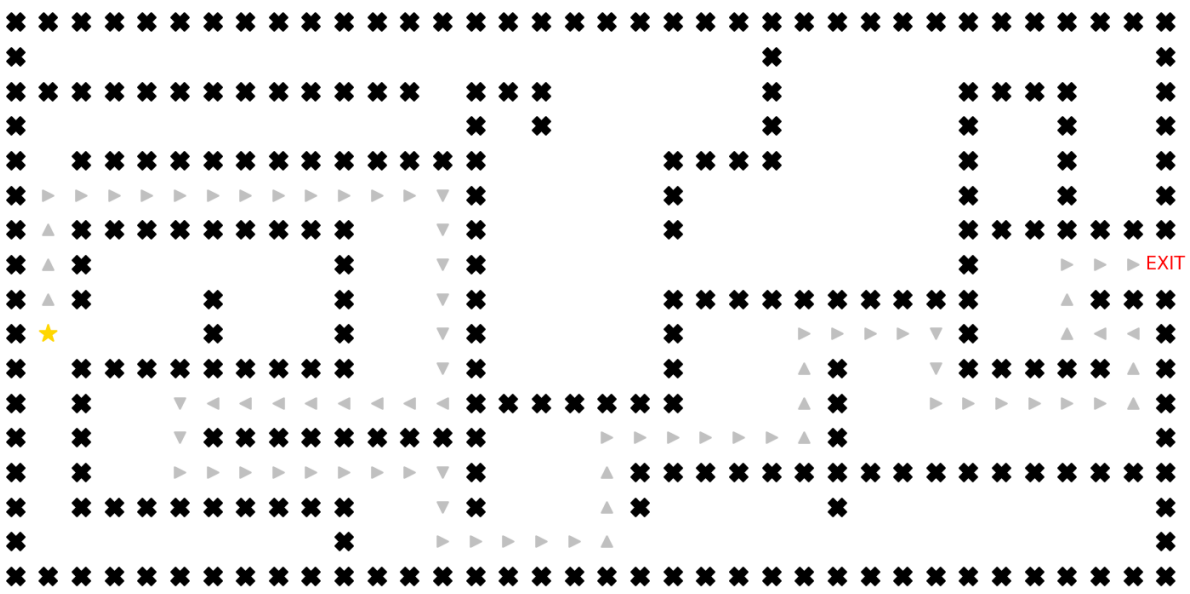
Output maze 3– heuristic 1:



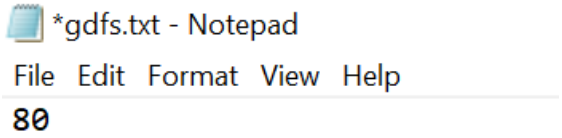
Cost maze 3-heuristic 1:



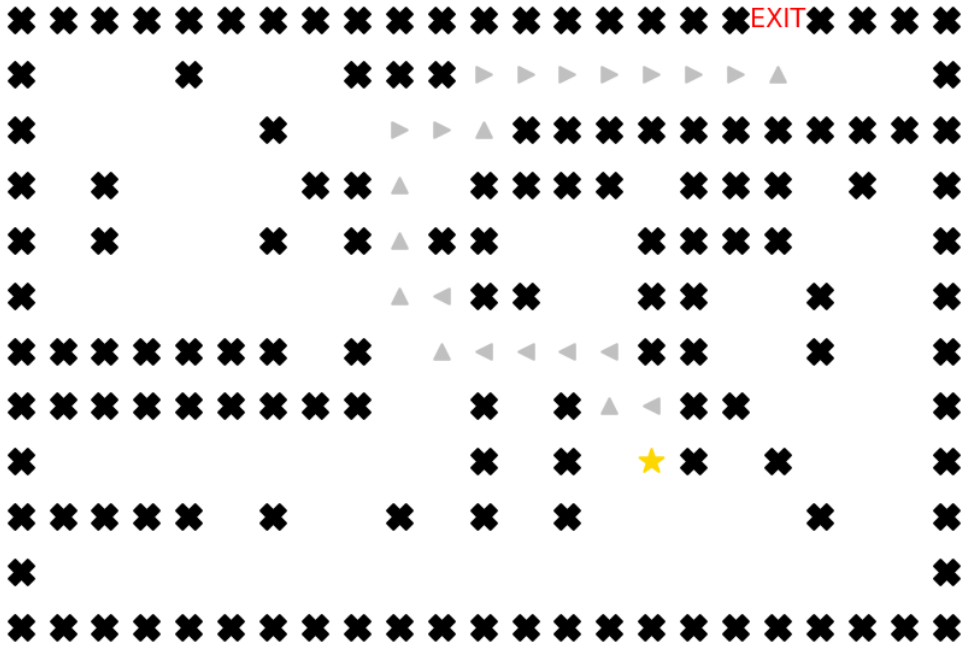
Output maze 3 - heuristic 2:



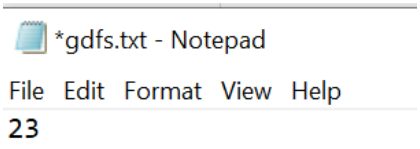
Cost maze 3 – heuristic 2:



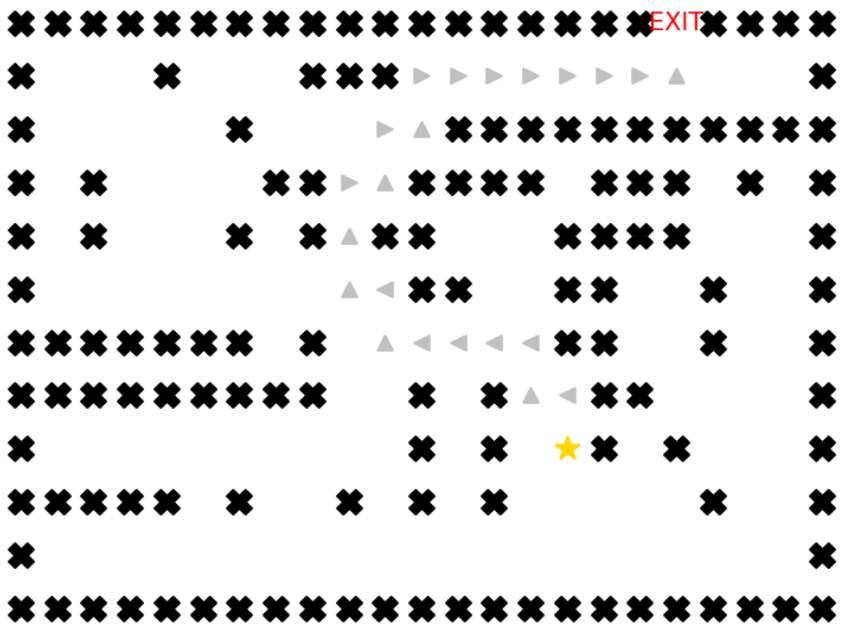
Output maze 4 – heuristic 1:



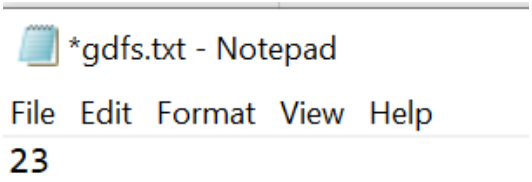
Cost maze 4-heuristic 1:



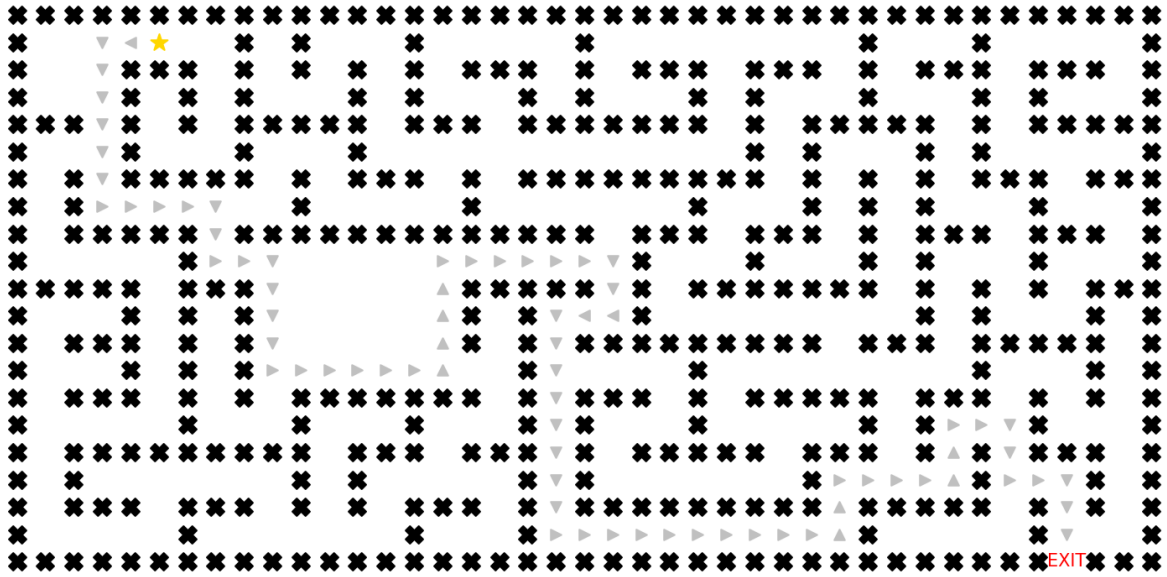
Output maze 4 - heuristic 2:



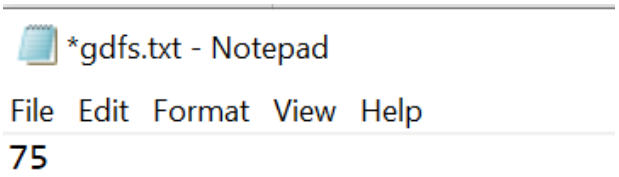
Cost maze 4 – heuristic 2:



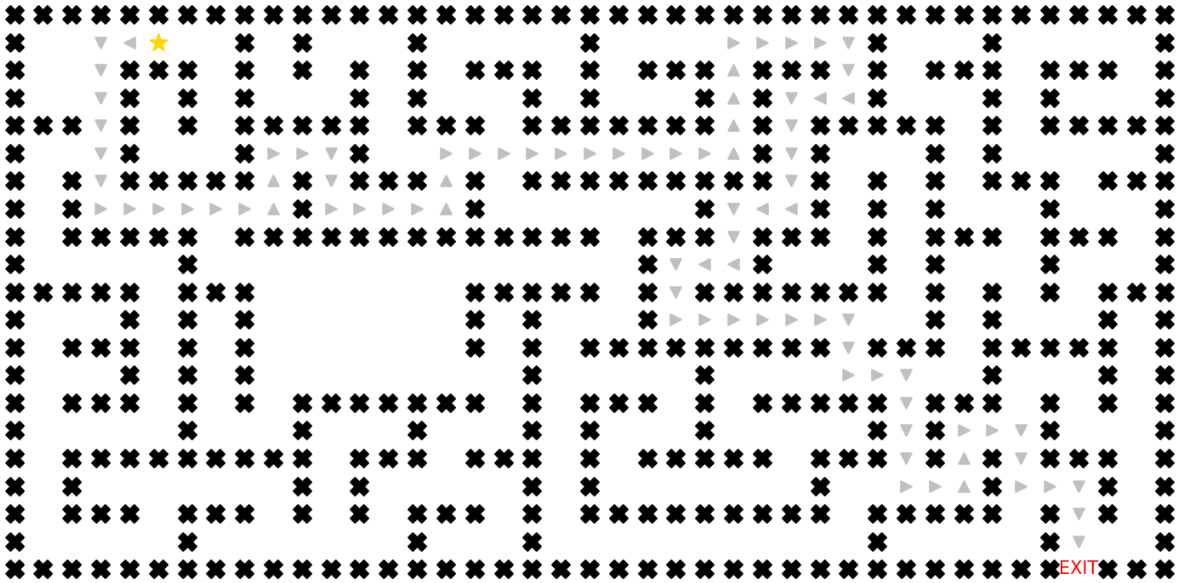
Output maze 5– heuristic 1:



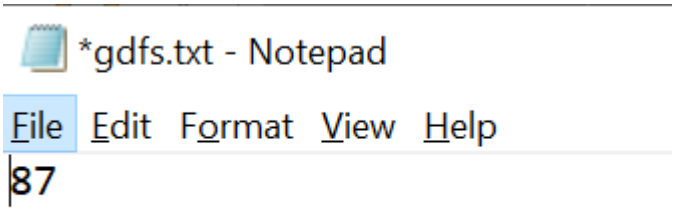
Cost maze 5-heuristic 1:



Output maze 5 - heuristic 2:



Cost maze 5 – heuristic 2:



Đánh giá thuật toán GBFS:

Tính đầy đủ: Có

Tính tối ưu: tìm được đường đi ngắn nhất với heuristic đủ tốt

Thời gian chạy: O (b^m) thực thi nhanh với heuristic đủ tốt

Không gian: O (b^m) giữ hết các node trong bộ nhớ

Về thời gian: Hàm heuristic 1 đa số trường hợp tối ưu hơn hàm heuristic 2.

Vì hàm heuristic 1 ưu tiên dùng cho maze có thể di chuyển về 4 phía, còn hàm heuristic 2 ưu tiên dùng cho việc di chuyển về nhiều phía hơn, nên khi tính heuristic 2 ta sẽ đường khoảng cách đường chim bay từ current node đến end node, nên khi current node chuyển tới node khác sẽ dễ bị chặn bởi tường nên sẽ chọn đường khác dài hơn.

5. **A STAR:**

Ý tưởng thuật toán:

A* lưu giữ một tập các lời giải chưa hoàn chỉnh, nghĩa là các đường đi qua đồ thị, bắt đầu từ nút xuất phát. Tập lời giải này được lưu trong một hàng đợi ưu tiên (priority queue). Tương tự GFBS nhưng hàm heuristic có thêm phần chi phí đường đi từ toạ độ ban đầu tới vị trí đó

Cài đặt thuật toán:

Cài đặt 2 hàm heuristic:

- Hàm heuristic 1:

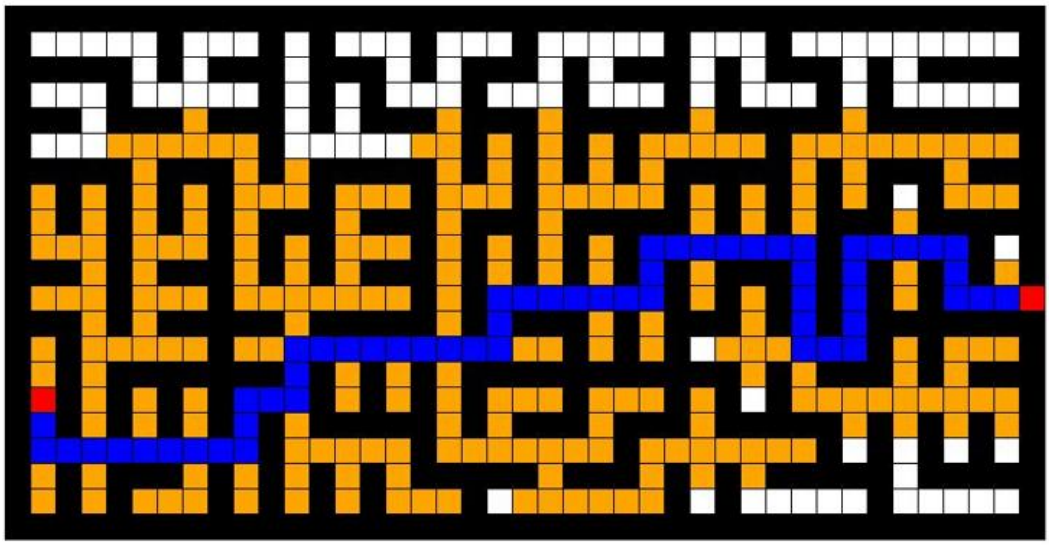
Dựa trên Manhattan Distance với việc vị trí có thể đi về 4 phía (up, down, left, right). Ở hàm heuristic này, ta trả về giá trị heuristic của 1 node là tổng chiều dài của hình chiếu của đường thẳng nối hai điểm này trong hệ trục tọa độ Descartes.

- Hàm heuristic 2: Cài đặt dựa trên ý tưởng đường chim bay là đường ngắn nhất, hàm trả về độ dài đoạn thẳng nối hai điểm bắt đầu và kết thúc

Kết quả chạy thuật toán:

Output maze 1:

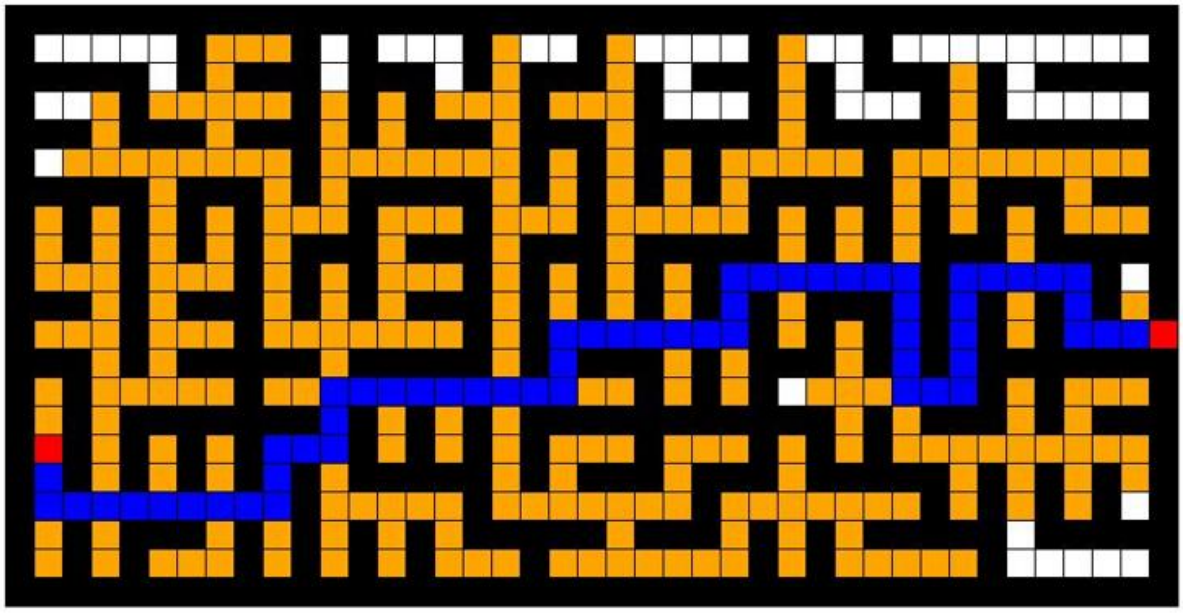
- Heuristic 1:



File Edit Format View Help

THE PATH COST IS: 60

Heuristic 2:

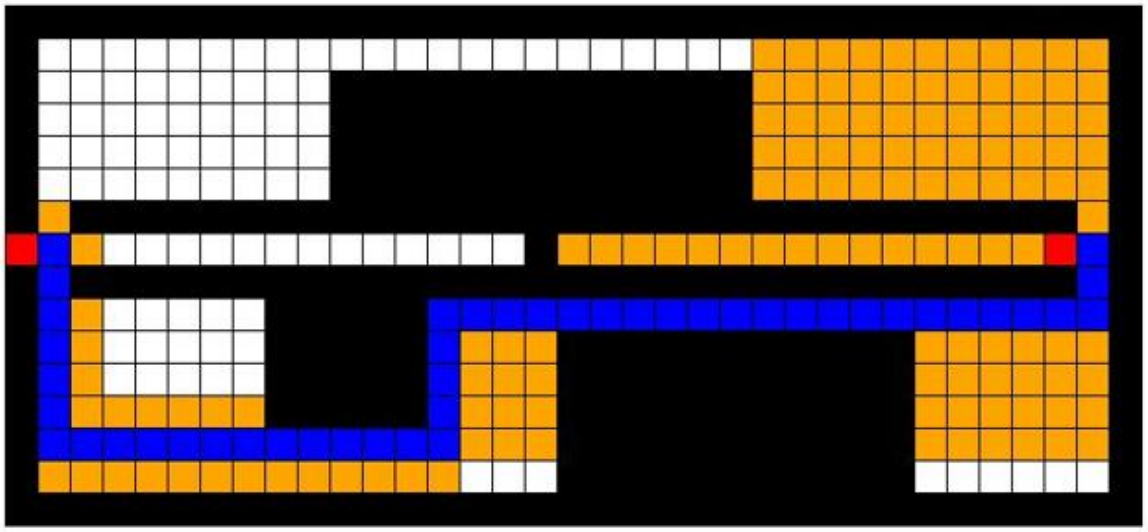


File Edit Format View Help

THE PATH COST IS: 60

Output maze 2:

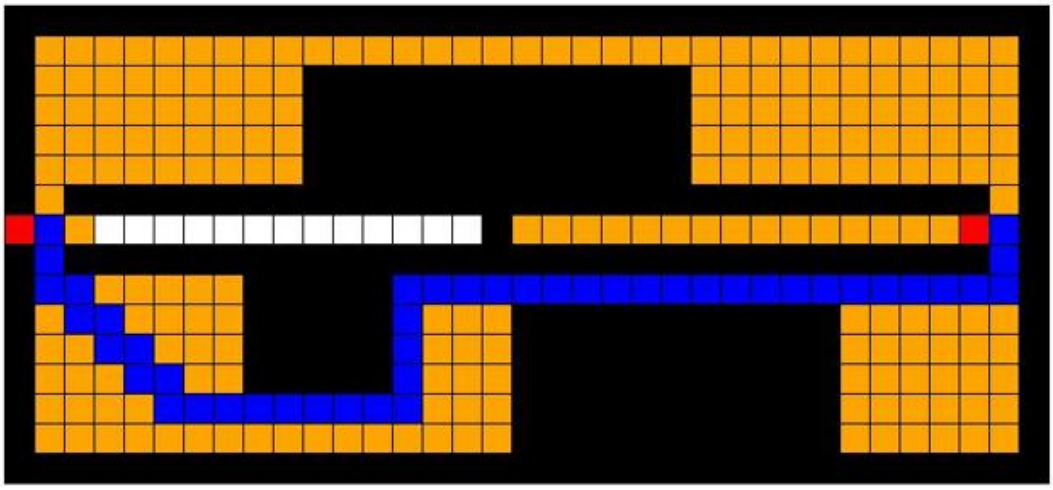
- Heuristic 1:



File Edit Format View Help

THE PATH COST IS: 47

- Heuristic 2:

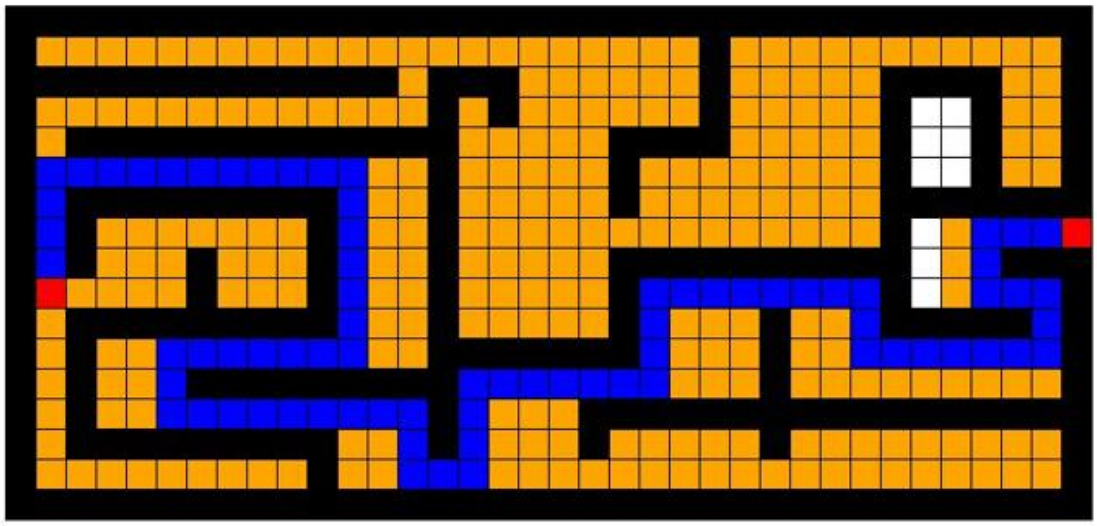


File Edit Format View Help

THE PATH COST IS: 47

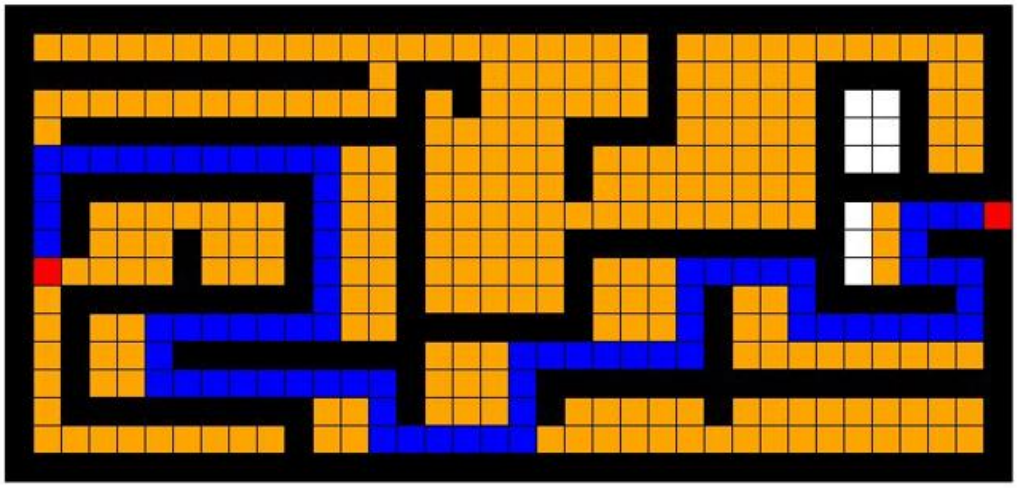
Output maze 3:

- Heuristic 1:



File Edit Format View Help
THE PATH COST IS: 77

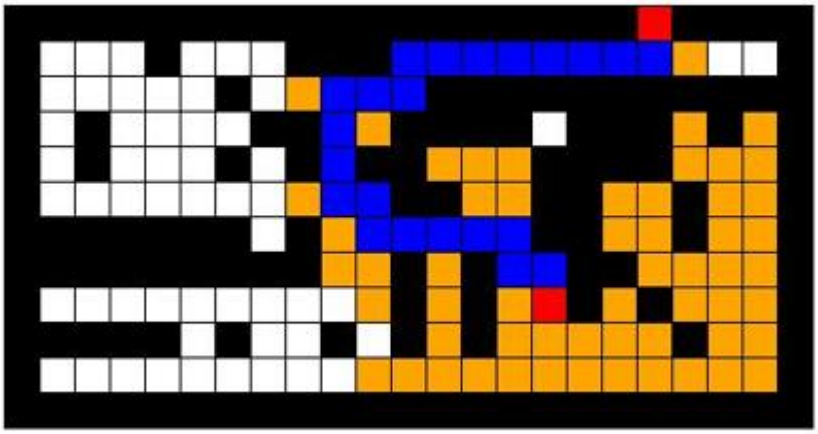
- Heuristic 2:



File Edit Format View Help
THE PATH COST IS: 77

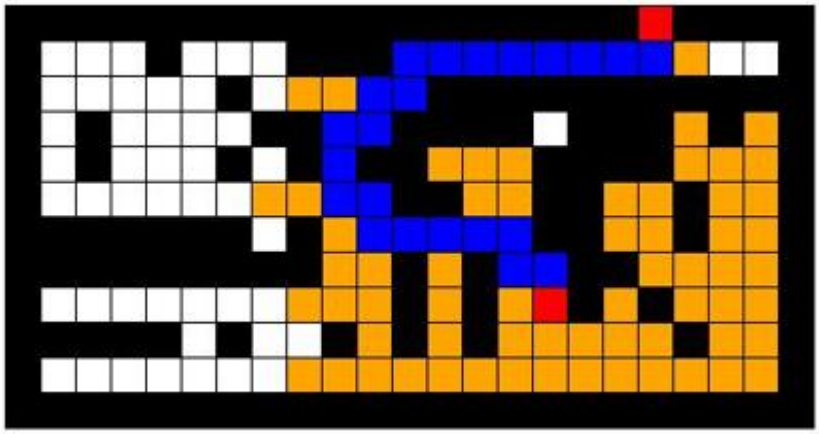
Output maze 4:

- Heuristic 1:



File Edit Format View Help
THE PATH COST IS: 24

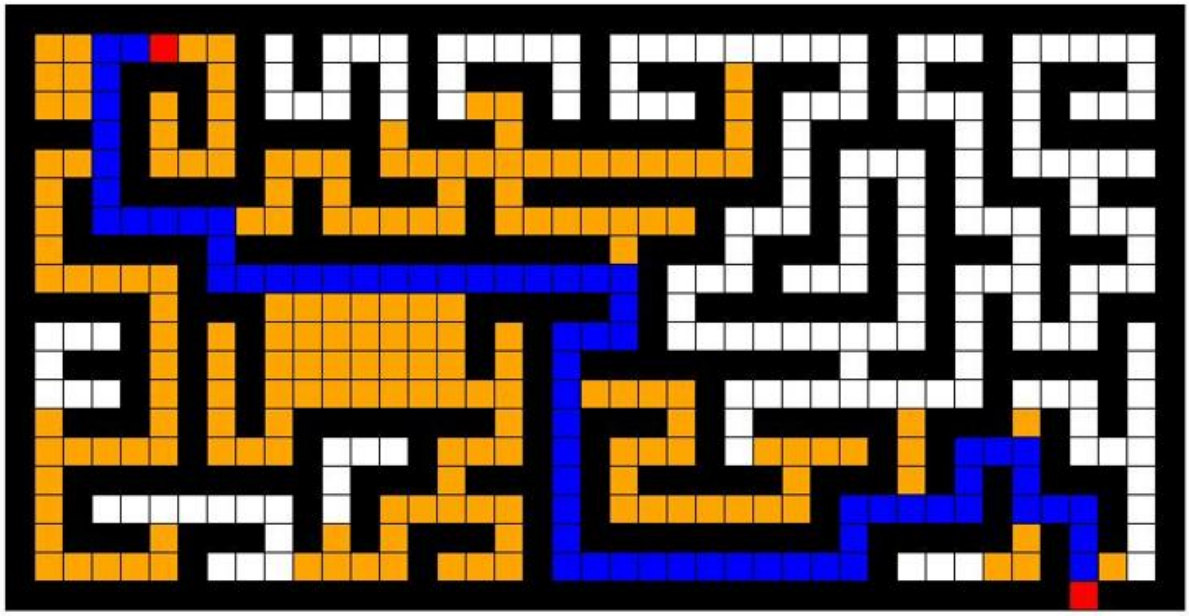
- Heuristic 2:



File Edit Format View Help
THE PATH COST IS: 24

Output maze 4:

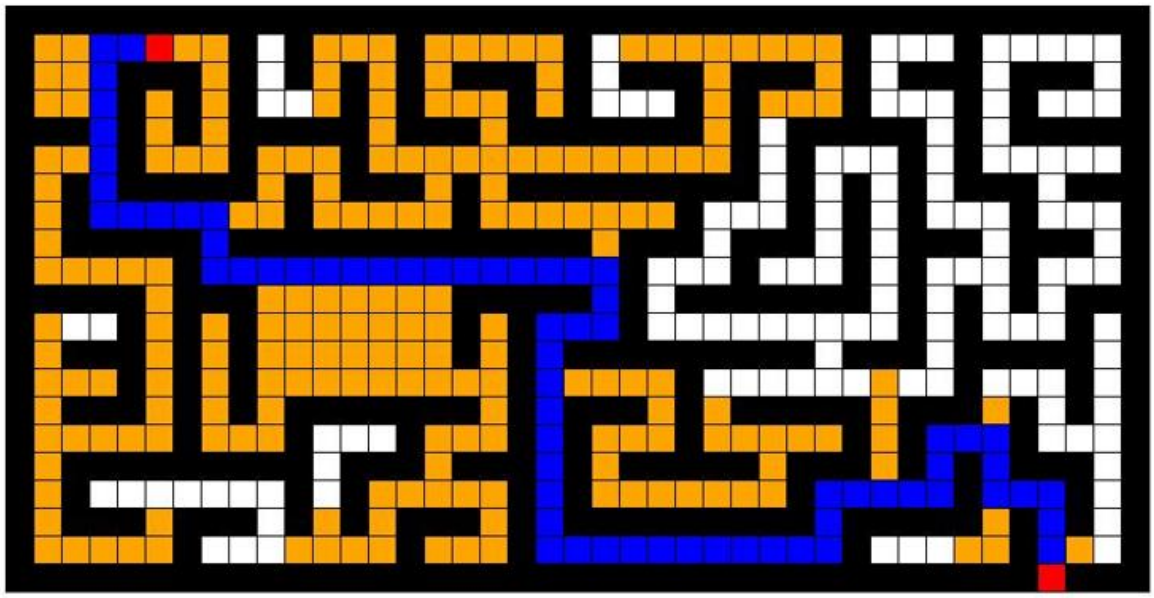
- Heuristic 1:

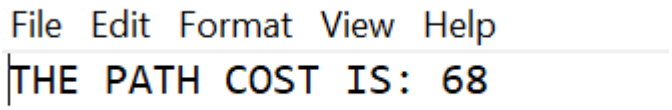


File Edit Format View Help

THE PATH COST IS: 68

- Heuristic 2:





Đánh giá thuật toán A start:

Với chi phí di chuyển thấp nhất là ϵ, C^* là chi phí lời giải tối ưu

Tính đầy đủ:

Tính tối ưu: có nếu heuristic hợp lý và nhất quán

Thời gian: cấp số mũ

Không gian: cấp số mũ

Đánh giá: với 2 heuristic đều cho ra 1 chi phí đường đi bằng nhau nên 2 heuristic đều có tối ưu như nhau

6. So sánh 5 thuật toán:

Xét về độ phức tạp thời gian, BFS sẽ chạy chậm hơn so với A* khi bản đồ lớn, vì BFS tìm kiếm mù theo chiều ngang dẫn đến việc có thể phải duyệt nhiều nút, nhiều nhánh mà không có khả năng nằm trong đường đi tối ưu. Trường hợp xấu, thuật toán sẽ duyệt toàn bộ bản đồ.

Thuật toán DFS đảm bảo luôn tìm đường đi ra khỏi mê cung, nhưng lại không phải là lời giải tối ưu. Trong hầu hết các trường hợp, DFS đều cho ra đường đi dài hơn đường đi ngắn nhất. Trong trường hợp tốt, thuật toán này có thể đưa ra đường đi ngắn hơn đường đi của GBFS. Có rất ít trường hợp tốt để DFS có thể cho ra kết quả tối ưu.

Thuật toán UCS đảm bảo về tính tối ưu nhưng chưa đảm bảo về thời gian tìm kiếm vì thực hiện tìm kiếm mù, trong trường hợp xấu nhất thì có thể duyệt toàn bộ bản đồ mới tìm ra được đường đi.

Thuật toán GBFS không có tính tối ưu, nhưng thời gian tìm kiếm kết quả tốt hơn thuật toán tìm kiếm không có thông tin. Có một số trường hợp tốt, thuật toán này sẽ tìm được đường đi ngắn nhất. Với trường hợp chỉ có 1 đường đi thì GBFS chạy nhanh hơn BFS và DFS.

Nhìn vào cả 5 bản đồ, kết quả nhận được khi sử dụng thuật toán A* đều là đường đi ngắn nhất. Có thể thấy A* là thuật toán tốt nhất trong 5 thuật toán khi vừa đảm bảo về thời gian tìm kiếm nhanh (giống GBFS) vừa đảm bảo về tính tối ưu (giống UCS).

III. BẢN ĐỒ CÓ ĐIỂM THƯỞNG

1. Phân tích bài toán:

Với bản đồ không có điểm thưởng, sử dụng thuật toán A* sẽ cho ta đường đi ngắn nhất (cũng là đường đi tốn ít chi phí nhất) từ Start (S) đến End (E). Việc xuất hiện các điểm thưởng (ký hiệu B_i) trên bản đồ làm cho đường đi ngắn nhất S → E không phải là đường đi tốn ít chi phí nhất. Lúc này, đường đi cần tìm có thể phải là S → B_i → … → B_j → E.

2. Đề xuất thuật toán:

Ban đầu ta có được đường đi từ S đến các đỉnh còn lại. Với mỗi đỉnh Bi, ta tìm được đường đi Bi → Bj, sau đó kiểm tra các chi phí: S → Bj và S → Bi + Bi →Bj. Nếu đường đi mới đến Bj thông qua Bi tốn ít chi phí hơn đường đi cũ S → Bj, ta cập nhật đường đi mới S → Bj = S → Bi + Bi → Bj (đỉnh E cũng được xem như 1 đỉnh Bj để xét).

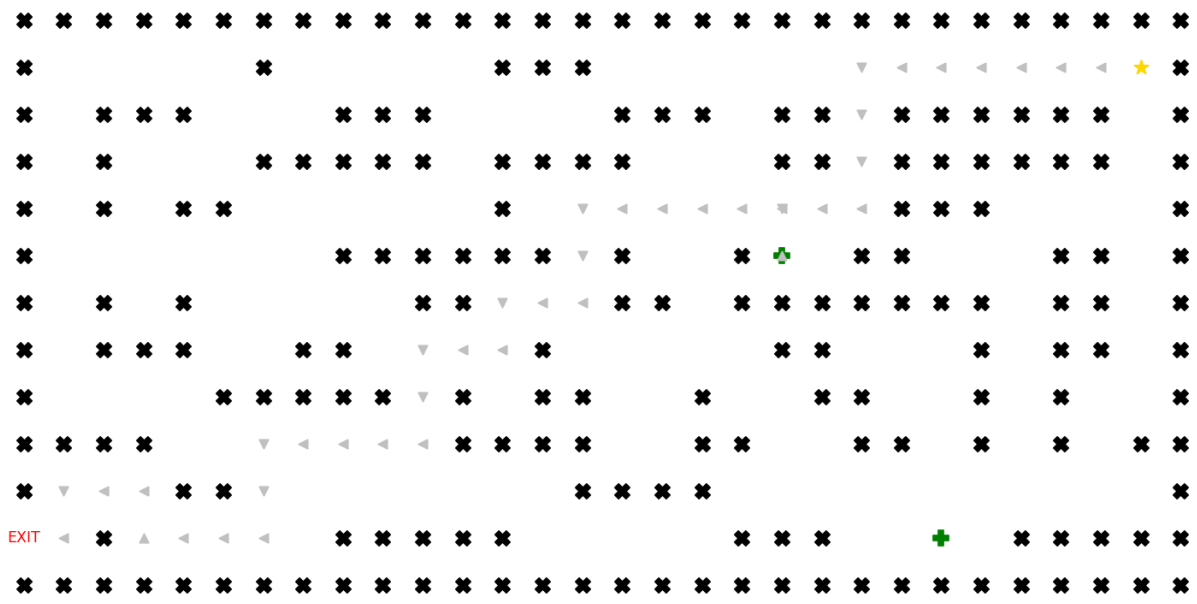
Để giảm bớt thời gian chạy chương trình khi phải duyệt 1 đỉnh Bi nhiều lần, nếu S → Bi có sự thay đổi, ta mới xét lại Bi để kiểm tra S → Bi → Bj theo giá trị S → Bi mới cập nhật. Còn nếu không thì không cần xét lại Bi.

Kết quả cuối cùng S → E sẽ là đường đi tốn ít chi phí nhất.

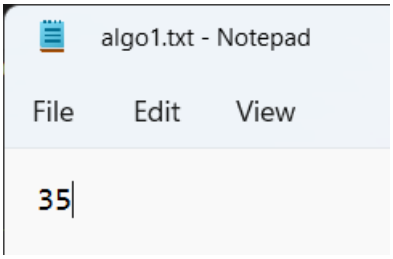
3. Kết quả thuật toán:

Maze 1 (Bản đồ 2 điểm thưởng):

Kết quả đường đi:

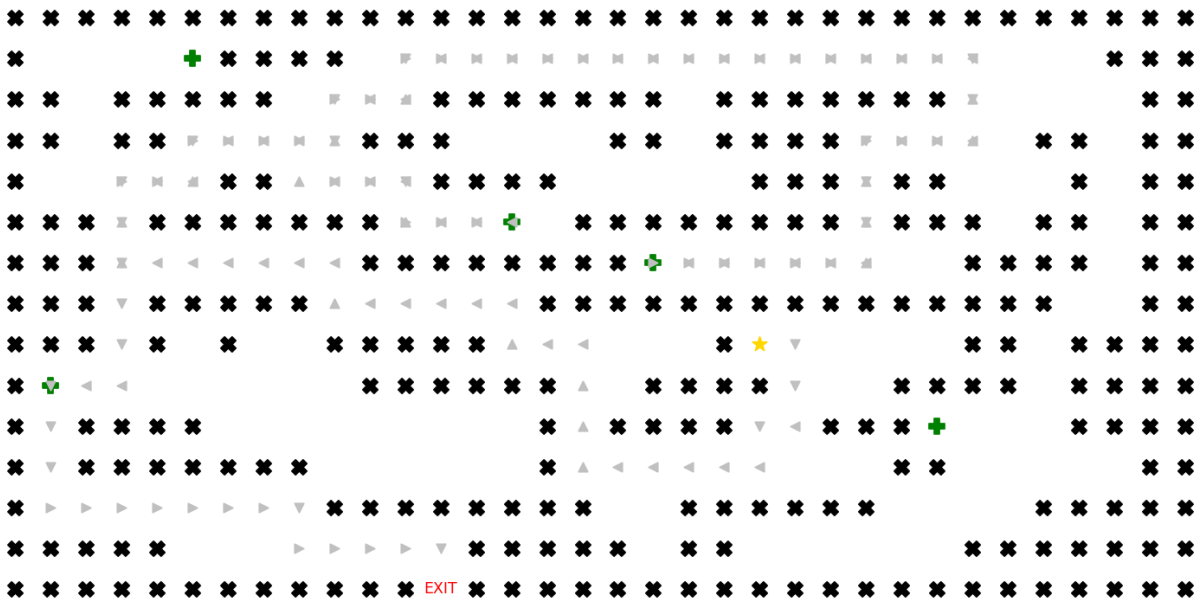


Chi phí đường đi:

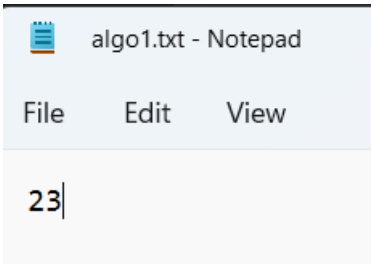


Maze 2 (Bản đồ 5 điểm thưởng):

Kết quả đường đi:

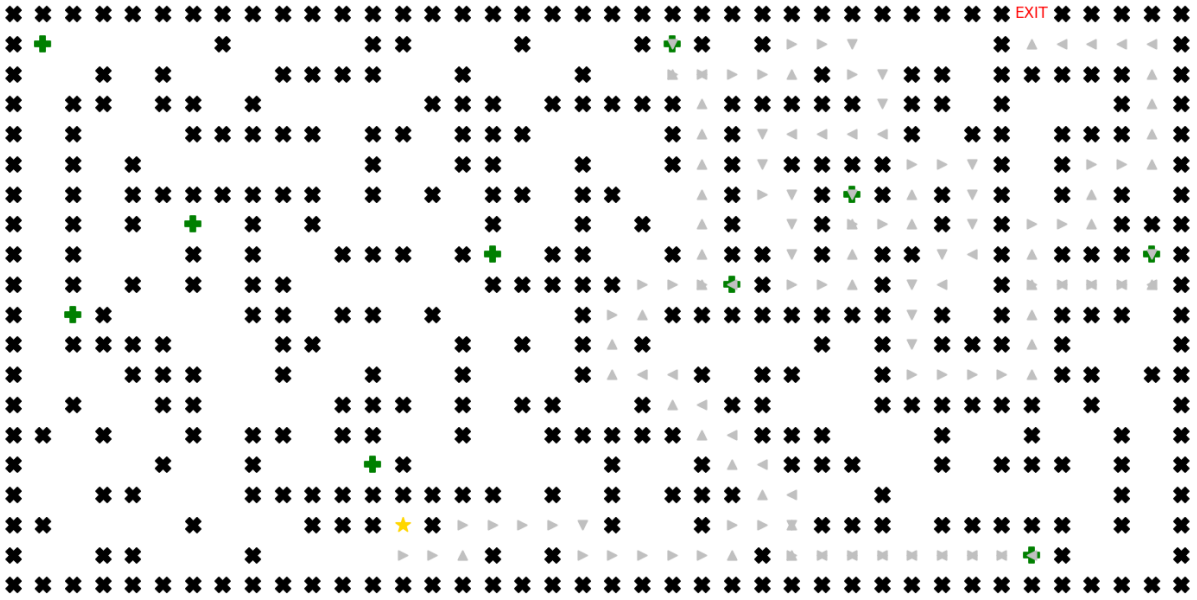


Chi phí đường đi:

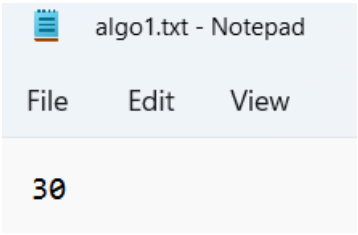


Maze 3 (Bản đồ 10 điểm thưởng):

Kết quả đường đi:



Chi phí đường đi:



IV. TÀI LIỆU THAM KHẢO

1. Implementing a Graph in Python - AskPython
2. Bellman–Ford Algorithm | DP-23 - GeeksforGeeks
3. Tài liệu lí thuyết của các thầy
4. https://www.youtube.com/watch?v=PMMc4VsIacU
5. Greedy Best First Search | Quick Explanation with Visualization - YouTube