

BÁO CÁO ĐỒ ÁN 1

CƠ SỞ TRÍ TUỆ NHÂN TẠO



ROBOT TÌM ĐƯỜNG

Mục lục

I.	Thông tin thành viên	3
II.	Mức độ hoàn thành	3
III.	Tổ chức đồ án.....	4
IV.	Tài liệu tham khảo.....	19

I. Thông tin thành viên

MSSV	HỌ VÀ TÊN
1712078	Ngô Phan Nhật Lâm
1712360	Phạm Hoàng Đức
1712475	Cao Nhon Hưng

II. Mức độ hoàn thành

STT	YÊU CẦU	HOÀN THÀNH
1	Cài đặt thành công 1 thuật toán để tìm đường đi từ S tới G. Báo cáo lại thuật toán và quá trình chạy thử. Chạy thử trường hợp không có đường đi.	100 %
2	Cài đặt ít nhất 3 thuật toán khác nhau (ví dụ tìm kiếm mù, tham lam, heuristic, ...). Báo cáo nhận xét sự khác nhau khi chạy thử 3 thuật toán.	100 %
3	Trên bản đồ sẽ xuất hiện thêm một số điểm khác được gọi là điểm đón. Xuất phát từ S, sau đó đi đón tất cả các điểm này rồi đến trạng thái G. Thứ tự các điểm đón không quan trọng. Mục tiêu là tìm ra cách để tổng đường đi là nhỏ nhất. Báo cáo thuật toán đã áp dụng và quá trình chạy thử.	100 %
4	Các hình đa giác có thể di động được với tốc độ h tọa độ/s. Cách thức di động ở mức đơn giản nhất là tới lui một khoảng nhỏ để đảm bảo không đè lên đa giác khác. Chạy ít nhất 1 thuật toán trên đó. Quay video và đính kèm link vào báo cáo.	100 %
5	Thể hiện mô hình trên không gian 3 chiều (3D)	0 %

III. Tổ chức đề án

Có 4 file code python tương ứng với 4 mức đề án, mỗi mức có các file input, output riêng tuân theo chuẩn input, output của yêu cầu đề án

Một số quy ước chung:

- Bản đồ tọa độ là số nguyên.
- Các cạnh của đa giác được vẽ bằng các điểm đã làm tròn số.
- Các mã nguồn đặt trong thư mục Mức 1, Mức 2, Mức 3, Mức 4.
- Ở mức 1 và mức 2 không cho phép robot đi chéo

1. Mức độ 1:

Chạy được một thuật toán tìm đường:

➤ Mô tả thuật toán:

Sử dụng chiến thuật tìm kiếm BFS, loan từ điểm xuất phát ra xung quanh nó đến khi đến được đích. Robot có thể đi ra 4 hướng, nên khi xây dựng cây BFS, hệ số phân nhánh của cây là 4. Số lần duyệt tối đa bằng số ô trong bảng $m \times n$, với $m, n \leq 50$ thuật toán vẫn có thể chạy tốt trong thời gian cho phép. Chi phí bộ nhớ là $O(m \times n)$, vì dùng một mảng để đánh dấu các ô trên ma trận, ô có giá trị bằng -1 là ô không được phép đi, ô có giá trị $INT_MAX=1000$ là ô chưa được đi, các giá trị còn lại là độ dài đường đi từ S đến ô đó.

Thuật toán:

BFS:

Tạo một hàng đợi Q

Cho ô S vào Q.

WHILE ô G chưa được cập nhật và Q chưa rỗng:

 Lấy P ra từ Q

 For A là các ô xung quanh P

 Nếu A không phải vật cản.

 Cập nhật đường đi từ S đến A.

 Kết nạp A vào Q.

Backtrack:

P=G

WHILE P!=S

Cho A là 4 ô xung quanh P

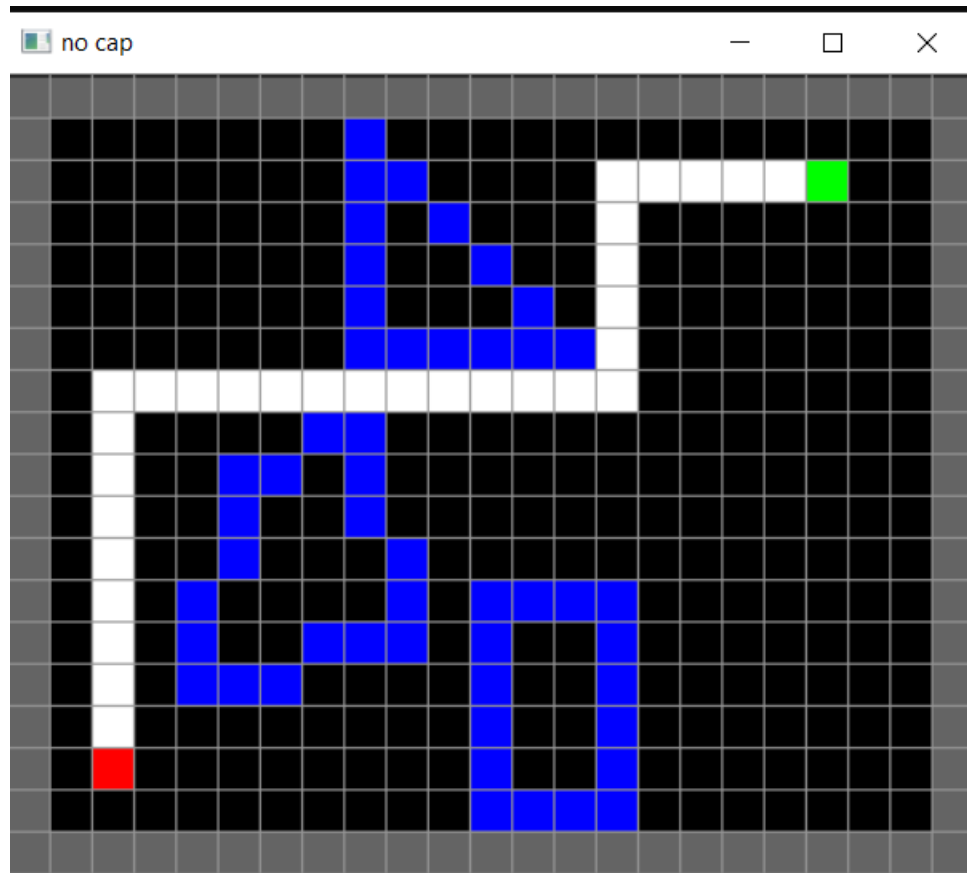
Cho S=A.min // tìm ô có giá trị nhỏ nhất để truy ngược

Thêm ô A vào đường đi.

➤ **Test:**

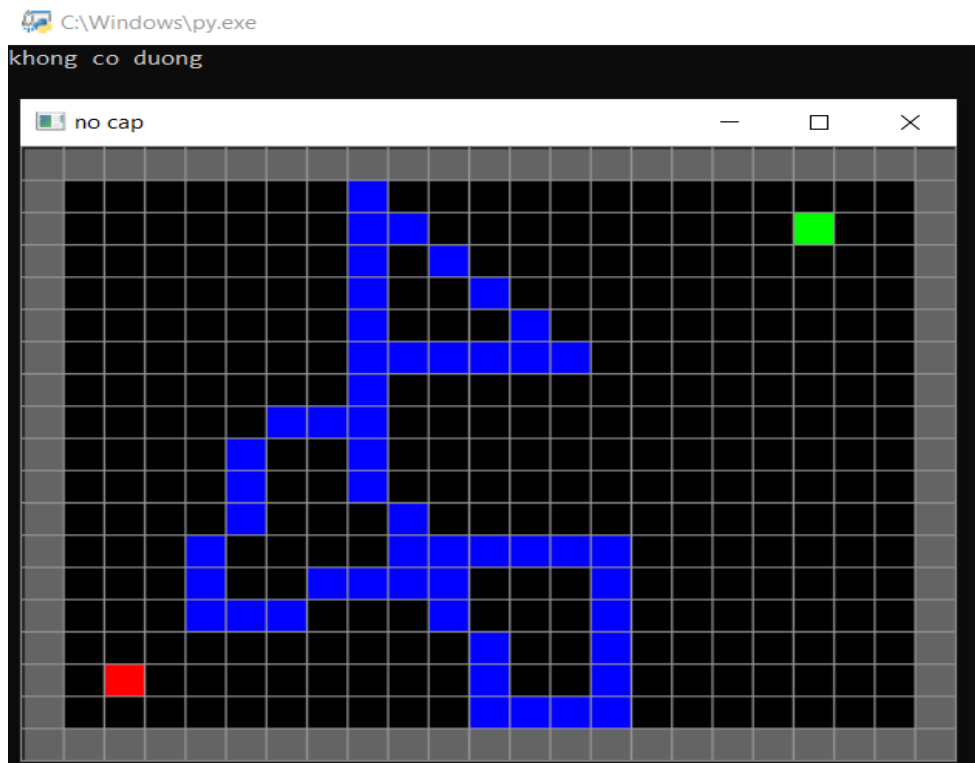
Trường hợp có đường đi:

Hướng dẫn: Chạy file BFS.py, đặt file input.txt chung thư mục với file BFS.py



Trường hợp không có đường đi:

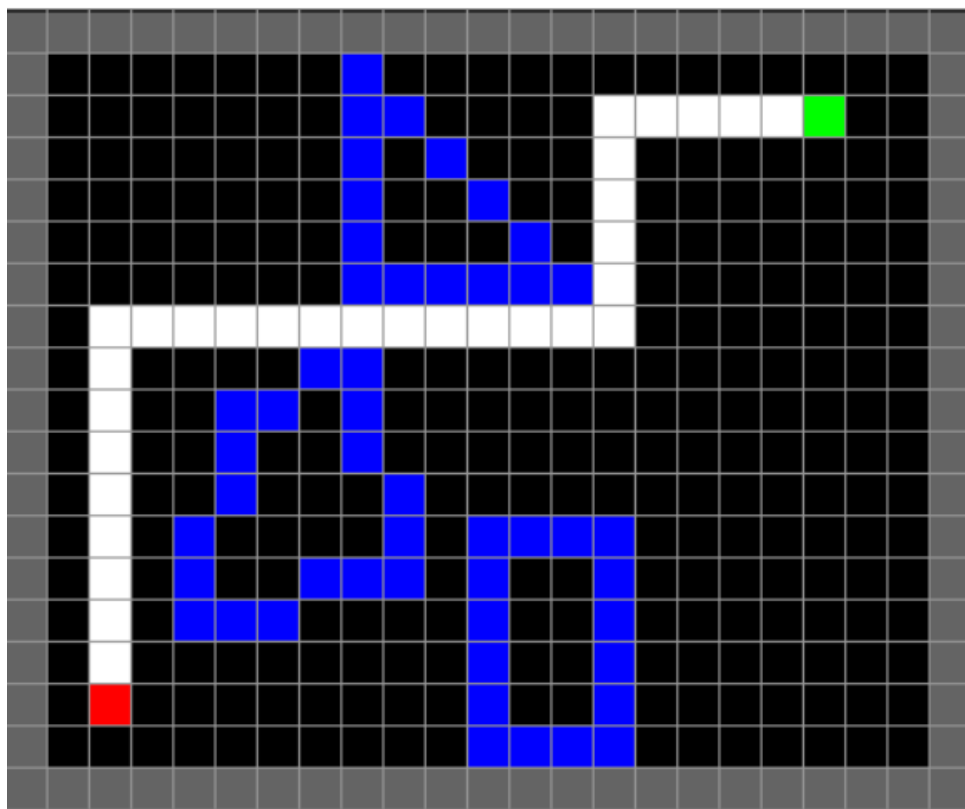
Hiện thông báo ra màn hình console “không có đường đi”.



2. Mức độ 2: BFS, GREEDY, BFSMC-A*

Tìm kiếm mù:

Sử dụng chiến thuật BFS như trong mức 1.



Tìm kiếm tham lam:

Hàm đánh giá:

Điểm đang xét P có tọa độ X_p, Y_p điểm đích có tọa độ X_g, Y_g hàm $f(P)$ được định nghĩa:

$$F(P) = (X_p - X_g)^2 + (Y_p - Y_g)^2$$

Thuật toán:

Greedy:

$P = S$

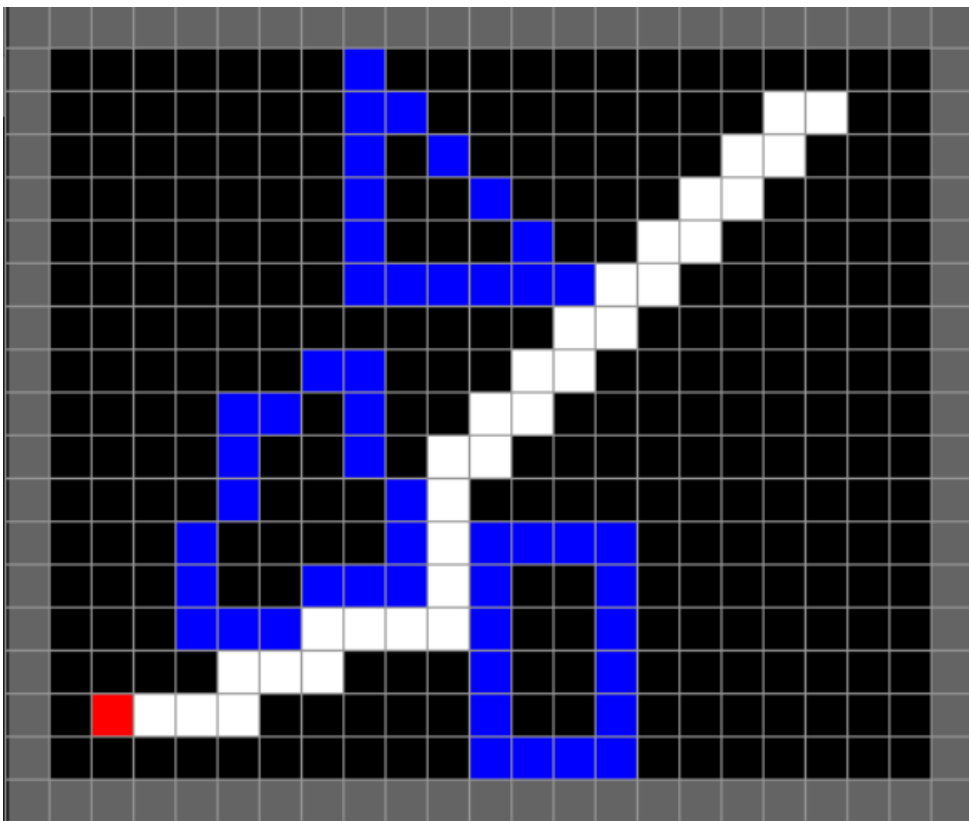
WHILE $P \neq G$

Cho A là tập 4 điểm xung quanh P

$P =$ phần tử A có $f(A_i)$ nhỏ nhất.

Thêm giá trị P vào đường đi.

Chạy file greedy.py (đặt tệp input.txt chung thư mục với GREEDY.py)



Tìm kiếm A^* :

Hàm đánh giá:

Điểm đang xét P có tọa độ X_p, Y_p điểm đích có tọa độ X_g, Y_g hàm $f(P)$ được định nghĩa:

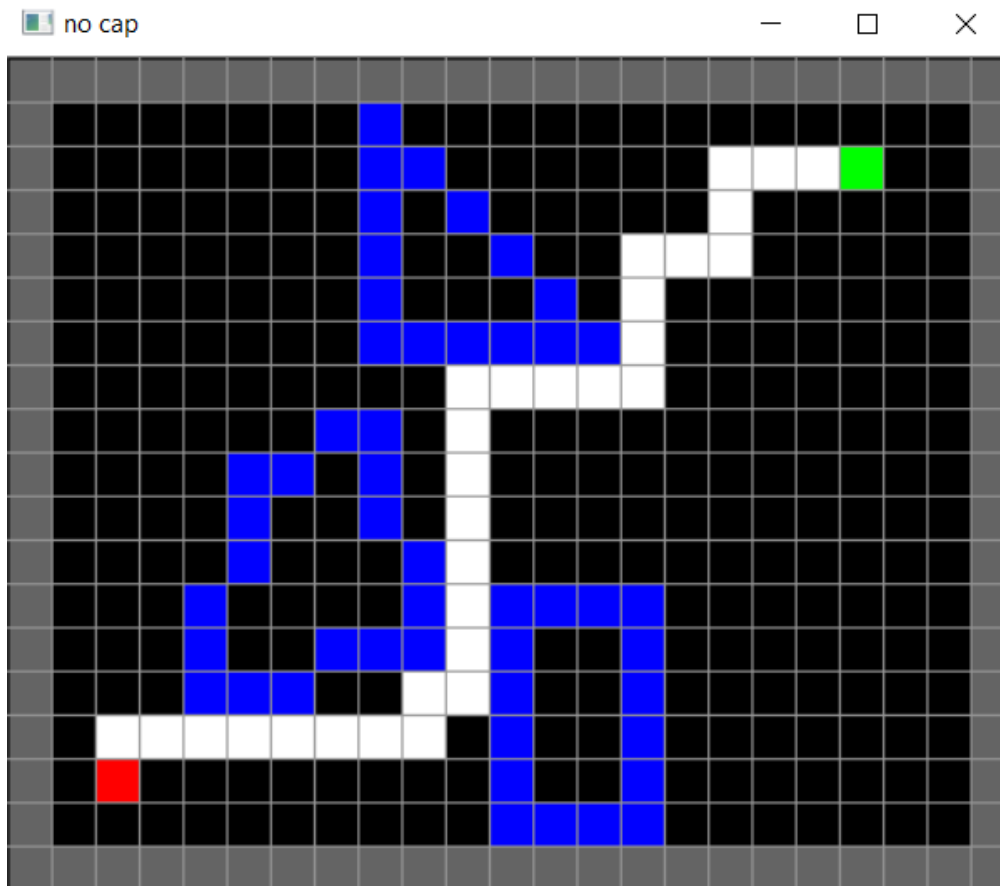
$$F(P) = \text{chi phí đi từ } S \text{ đến } P + (X_p - X_g) + (Y_p - Y_g)$$

Trong đó chi phí từ S đến P được cập nhật tăng thêm 1 sau mỗi bước. Áp dụng chiến thuật tìm kiếm lai BFS-MC: Đầu tiên ta cho duyệt BFS 15 điểm lân cận đó, sau đó dò trong tập mở để tìm ô có $f(P)$ thấp nhất để chọn làm nút gốc duyệt BFS lần tiếp theo.

Do thuật toán A* cần hàm $g(x)$ là chi phí thực sự đi từ điểm xuất phát đến điểm đang xét, nên ta bắt buộc phải dùng BFS để có thể tính được $g(x)$.

Chạy file BFSMC.py:

Đặt tệp input.txt chung thư mục với tệp BFSMC.py



Đánh giá về các thuật toán:

◆ Thuật toán tìm kiếm mù:

BFS đảm bảo luôn tìm được đường đi (nếu có) và với chi phí nhỏ nhất do quy định không đi chéo và chi phí đi bằng với số lần duyệt để đến G.

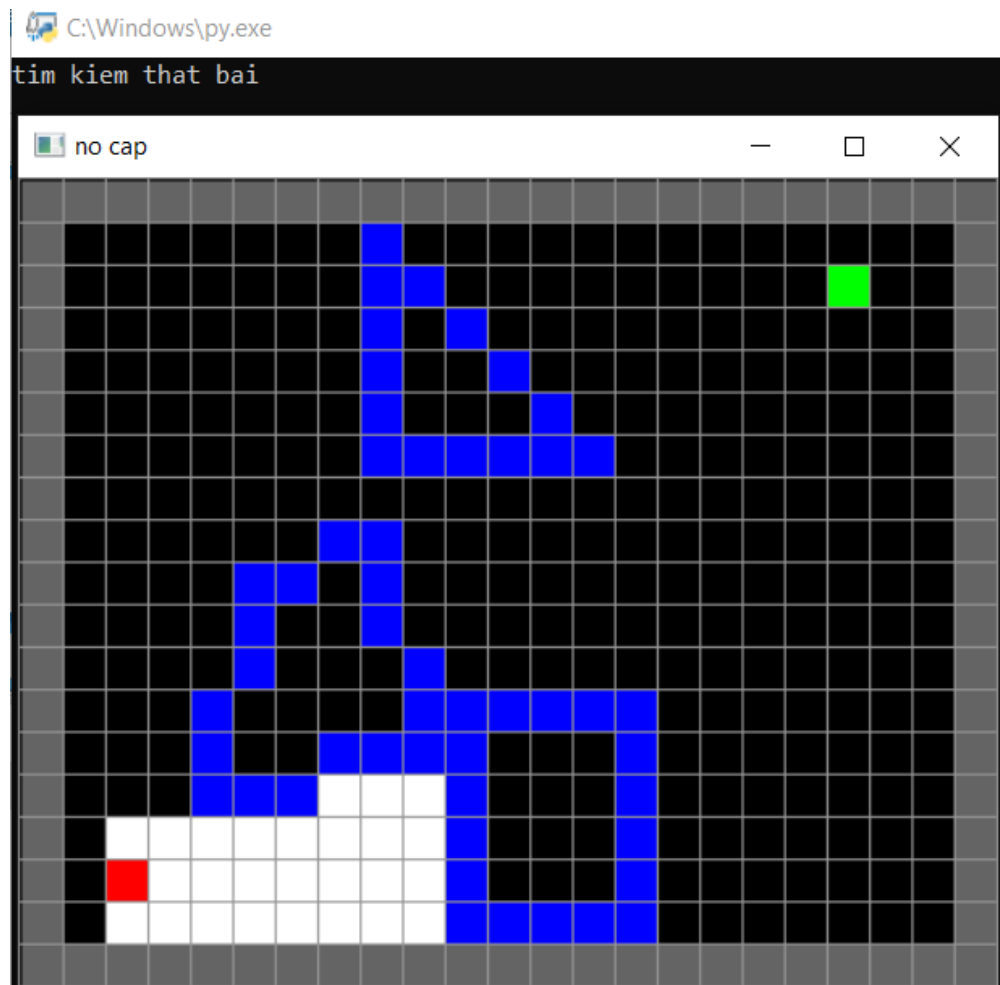
◆ Greedy:

Gần với cách nghĩ tự nhiên của con người, đi theo một hướng duy nhất tới đích.

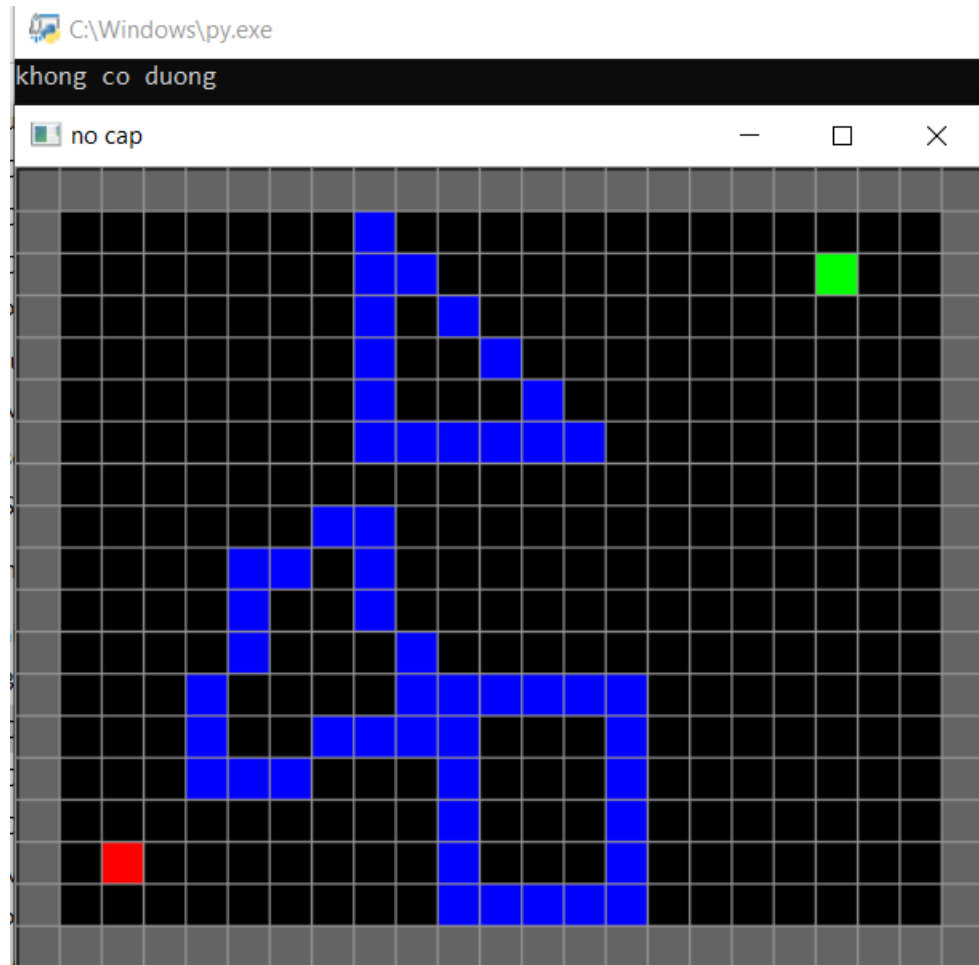
◆ Thuật toán A*:

Trong trường hợp này A* bị suy biến do không có thông tin về chi phí từ S đến các ô cần xét để tính $g(x)$.

Trường hợp đi vào ngõ cụt của Greedy:



Trường hợp đi vào ngõ cụt của BFSMC: in ra thông báo trên màn hình console.



3. Mức độ 3: muc3.py

▪ Vấn đề:

Trên bảng đồ sẽ xuất hiện thêm một số điểm khác nhau (được gọi là điểm đón). Tìm đường đi xuất phát từ S đến G sao cho lần lượt đi qua các điểm này và phải sắp xếp thứ tự ưu tiên đi qua chúng sao cho đoạn đường đi là ngắn nhất.

▪ Cài đặt:

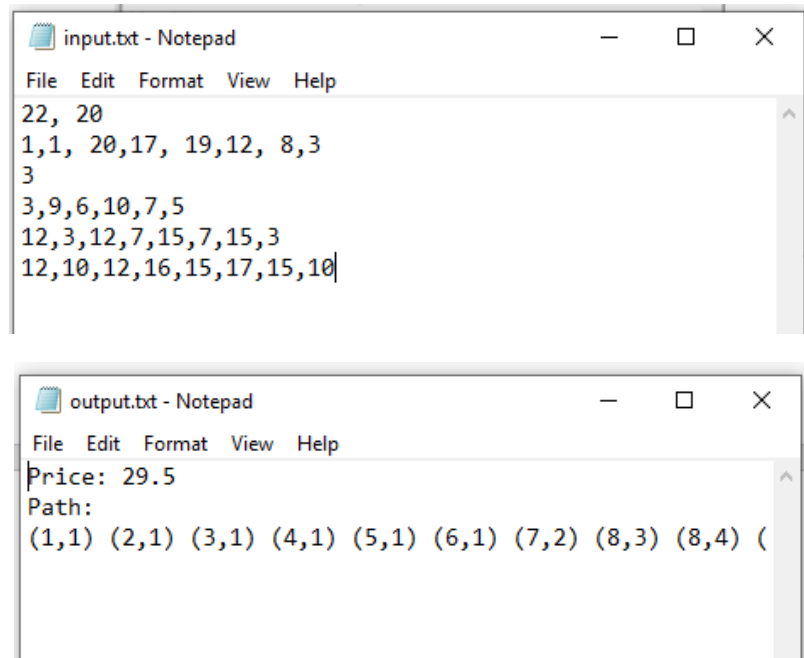
- Đầu tiên đọc các tham số cần thiết từ file input.txt bằng hàm readfile().

- Tiếp theo, ta sẽ đi tìm thứ tự ưu tiên đi qua các điểm đón nào trước bằng hàm FindPriority(). Quy ước chi phí được tính là quãng đường đi giữa 2 điểm, nếu đi dọc hay ngang thì chi phí mỗi bước là 1, nếu đi chéo thì chi phí là 1,5. Quá trình tìm này dựa vào ý tưởng thuật toán AStar với hàm ước lượng heuristic $f(x)$. Trước hết, ta sẽ tìm một điểm gần G nhất trong số các điểm đón rồi đưa điểm đó vào tập đóng stack, cứ tiếp tục tìm trong tập các điểm đón còn lại gần nhất với điểm trên top của stack, cho đến khi đi về điểm S.

- Sau khi đã có thứ tự ưu tiên đi qua mỗi điểm thì ta sẽ tìm đường đi chi tiết từ S đi qua các điểm đó và đến G sao cho tránh các vật thể đã cho bằng hàm FindPath(). Cuối cùng là xuất kết quả ra file output.txt và biểu diễn đồ họa đường đi.

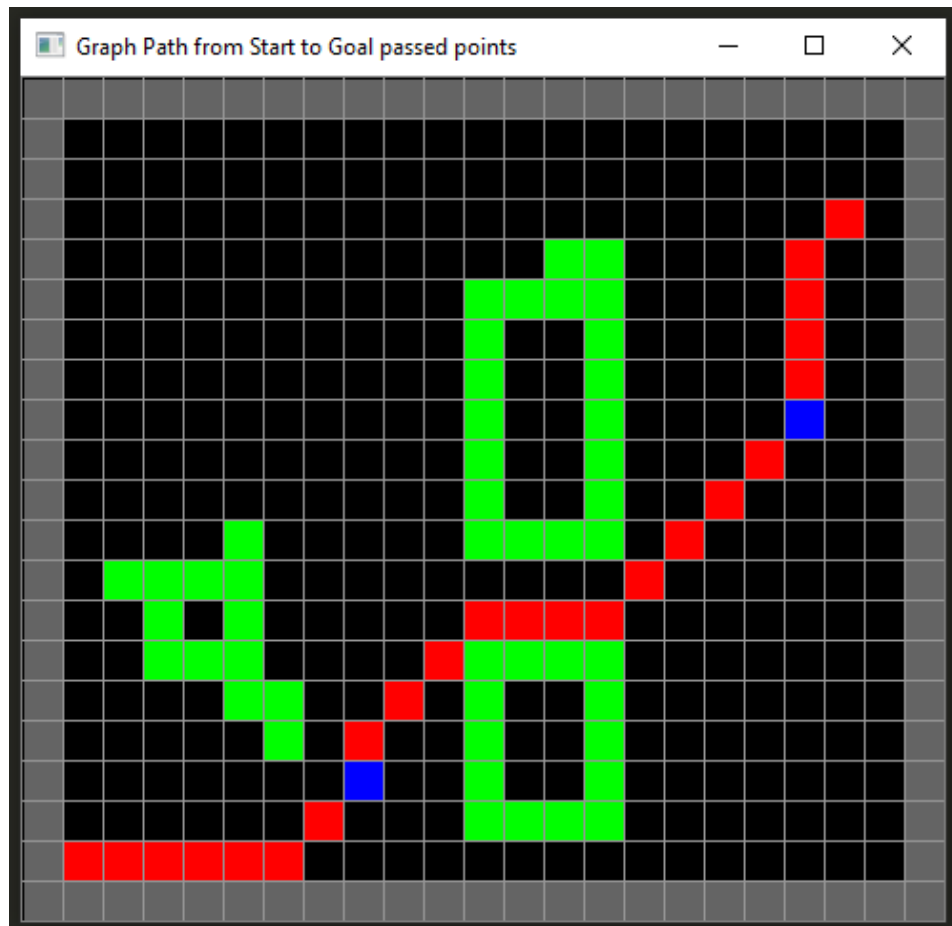
▪ Chạy thử:

Test1:



```
input.txt - Notepad
File Edit Format View Help
22, 20
1,1, 20,17, 19,12, 8,3
3
3,9,6,10,7,5
12,3,12,7,15,7,15,3
12,10,12,16,15,17,15,10

output.txt - Notepad
File Edit Format View Help
Price: 29.5
Path:
(1,1) (2,1) (3,1) (4,1) (5,1) (6,1) (7,2) (8,3) (8,4) (
```



Test2:

```

input.txt - Notepad
File Edit Format View Help
22, 20
1,1, 20,17, 16,5,8,3
3
3,9,6,10,7,5
12,3,12,7,15,7,15,3
12,10,12,16,15,17,15,10

output.txt - Notepad
File Edit Format View Help
Price: 34.0
Path:
(1,1) (2,1) (3,1) (4,1) (5,1) (6,1) (7,2) (8,3) (9,3) (

```



Test 3:

```

input.txt - Notepad
File Edit Format View Help
22, 20
1,1, 12,16,16,5,8,3
3
3,9,6,10,7,5
12,3,12,7,15,7,15,3
12,10,12,16,15,17,15,10

output.txt - Notepad
File Edit Format View Help
Price: 36.5
Path:
(1,1) (2,1) (3,1) (4,1) (5,1) (6,1) (7,2) (8,3) (9,3) (

```



Test 4:

```

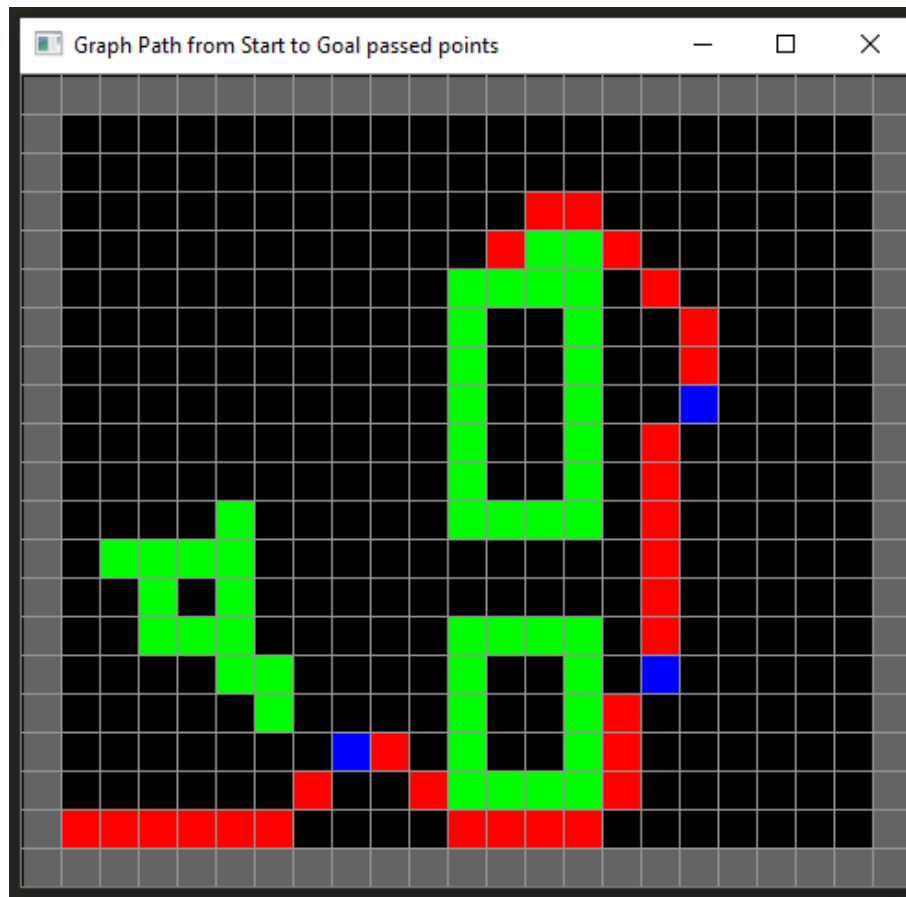
input.txt - Notepad
File Edit Format View Help
22, 20
1,1, 12,16,16,5,8,3, 17, 12
3
3,9,6,10,7,5
12,3,12,7,15,7,15,3
12,10,12,16,15,17,15,10

```

```

output.txt - Notepad
File Edit Format View Help
Price: 36.5
Path:
(1,1) (2,1) (3,1) (4,1) (5,1) (6,1) (7,2) (8,3) (9,3) (

```



Test 5:

```
input.txt - Notepad
File Edit Format View Help
22, 20
1,1, 12,16,16,5,8,3, 21,12
3
3,9,6,10,7,5
12,3,12,7,15,7,15,3
12,10,12,16,15,17,15,10
```

```
C:\Users\NGUYỄN ANH NAM\Downloads>b.py

C:\Users\NGUYỄN ANH NAM\Downloads>b.py
Traceback (most recent call last):
  File "C:\Users\NGUYỄN ANH NAM\Downloads\b.py", line 331, in <module>
    res, dis=FindPath(start, goal, list_diemdon, maze)
  File "C:\Users\NGUYỄN ANH NAM\Downloads\b.py", line 303, in FindPath
    array, dis = FindPriority(start, end, list, graph)
  File "C:\Users\NGUYỄN ANH NAM\Downloads\b.py", line 269, in FindPriority
    path, dis, endpos=AStarSearch(closed_path[j], list[i], graph)
  File "C:\Users\NGUYỄN ANH NAM\Downloads\b.py", line 239, in AStarSearch
    raise RuntimeError("Can not find a path !!!")
RuntimeError: Can not find a path !!!

C:\Users\NGUYỄN ANH NAM\Downloads>
```


4. Mức độ 4: muc4.py

a. Yêu cầu:

Robot di chuyển đến đích trong không gian có các vật cản hình đa giác có thể di động được với tốc độ h tọa độ/s. Cách thức di động có thể ở mức đơn giản nhất là tới lui một khoảng nhỏ để đảm bảo không đề lên đa giác khác.

b. Input:

Như chuẩn đề bài và thêm 1 dòng để chứa tốc độ h

- test1.txt

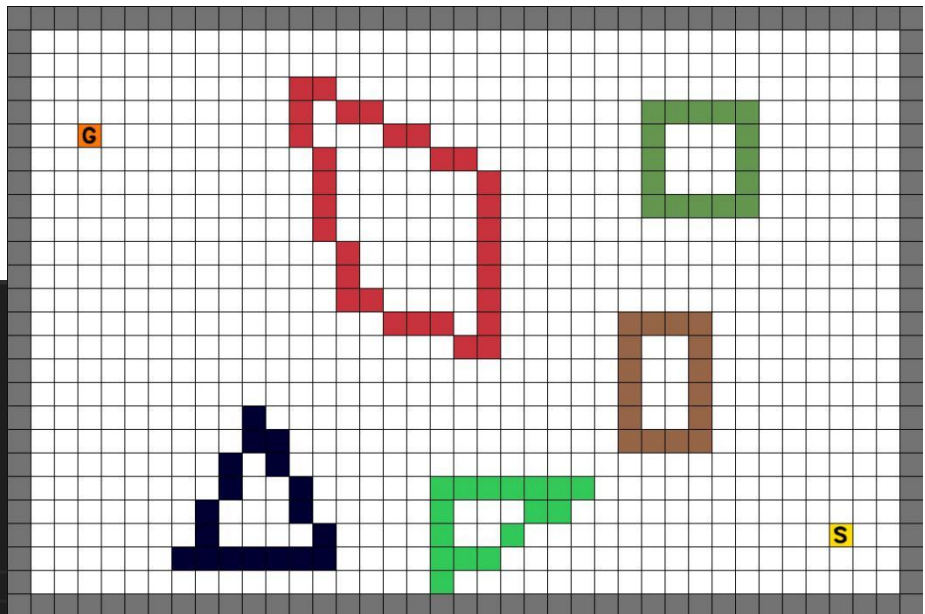
```
test1.txt
1 22,18
2 2,2,19,16
3 4
4 4,4,5,9,8,10,9,5
5 8,12,8,17,13,12
6 11,1,11,6,14,6,14,1
7 17,14,20,14,17,11
8 3
```

- test2.txt

```
test2.txt
1 22,20
2 6,19,21,1
3 3
4 3,9,6,10,7,5
5 7,13,10,17,11,9
6 12,2,12,7,15,7,15,2
7 3
```

- test3.txt

```
test3.txt
1 38,25
2 35,3,3,20
3 5
4 7,2,10,8,13,2
5 14,13,20,11,20,18,12,22
6 26,7,26,12,29,12,29,7
7 27,21,31,21,31,17,27,17
8 18,5,24,5,18,1
9 3
```



c. Tổng thể:

Các vật cản sẽ di chuyển bất kỳ theo 8 hướng, robot tìm đường đi ban đầu và đi theo, nếu gặp vật cản không thể đi nữa thì tìm đường đi mới.

d. Chi tiết cách thực hiện:

- Chỉ xét các tọa độ nguyên; robot có thể đi chéo
- Đọc tất cả giá trị từ input và đánh dấu tương ứng vào mảng 2 chiều **maze**, chỉ số 1 ô của mảng (i, j) tương ứng với tọa độ (j, i) trên đồ thị thực tế

Giá trị	Nội dung biểu thị
-1	border
0	empty spot
1	start point
2	goal point
3	pickup points (trong mức độ này ta không xét)
4	robot spot
>= 5	polygons

- Dùng giải thuật BFS để tìm kiếm đường đi ngắn nhất cho robot: hàm *bfs(maze, start)* trả về một list các cặp tọa độ là điểm mà robot có thể đi để đến được đích
- Ta cho biến route chứa đường đi của frame gốc ban đầu – đọc từ input
- Sau đó nhích robot lên 1 bước theo route, sau đó thực hiện xử lý các frame sau
- Tạo vòng lặp while để xử lý từng frame

- Mỗi frame tương ứng với vị trí các vật thể tại một thời điểm
- Thời gian chờ của mỗi frame là $1000(ms)/h$ để tương ứng h tọa độ/s
- Tại mỗi frame:
 - Cho các đa giác di chuyển đến 1 hướng bất kỳ trong bán kính 1 ô lân cận
 - Xét điểm tiếp theo trong route, nếu có thể đi được (không dính vật cản, không dính lề, không bị đi vào trong vật cản) thì ta cho robot nhích đến ô đó; ngược lại ta qua frame để các vật cản di chuyển, nếu đồ thị mới vẫn không đi được thì ta thực hiện tính toán lại route mới cho robot, bắt đầu từ điểm đang xét
 - Đồng thời ta cũng xét đến việc nếu vật cản di chuyển liên tục qua một số lượng frame nhất định nhưng robot vẫn không tìm được đường đi thì xuất ra màn hình không có đường đi, kết thúc while.
 - Robot được xem là đến đích khi route chỉ còn chứa 1 điểm là điểm đích.
 - Với mỗi bước di chuyển được của robot, ta cộng vào tổng chi phí, các bước di chuyển ngang/dọc có chi phí 1, chéo có chi phí 1.5

e. Output:

- **Đồ họa biểu diễn:** video demo output cho file input: test3.txt
https://www.youtube.com/watch?v=zclbM8Rpl_Y
- File output.txt chứa một số là tổng chi phí để đến được đích của robot

IV. Tài liệu tham khảo

1. https://rosettacode.org/wiki/A*_search_algorithm#Python
2. https://docs.opencv.org/2.4/modules/core/doc/drawing_functions.html