TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO ĐỒ ÁN 1 – ROBOT TÌM ĐƯỜNG

Bộ môn: Cơ sở trí tuệ nhân tạo

LỚP	**:**	**Cử nhân Tài năng – Khóa 2017**
NHÓM SINH VIÊN	**:**	**1712168 – Trần Lê Bá Thịnh**
	:	**1712237 – Đặng Tấn Tài**
	:	**1712606 – Nguyễn Thanh Nam**

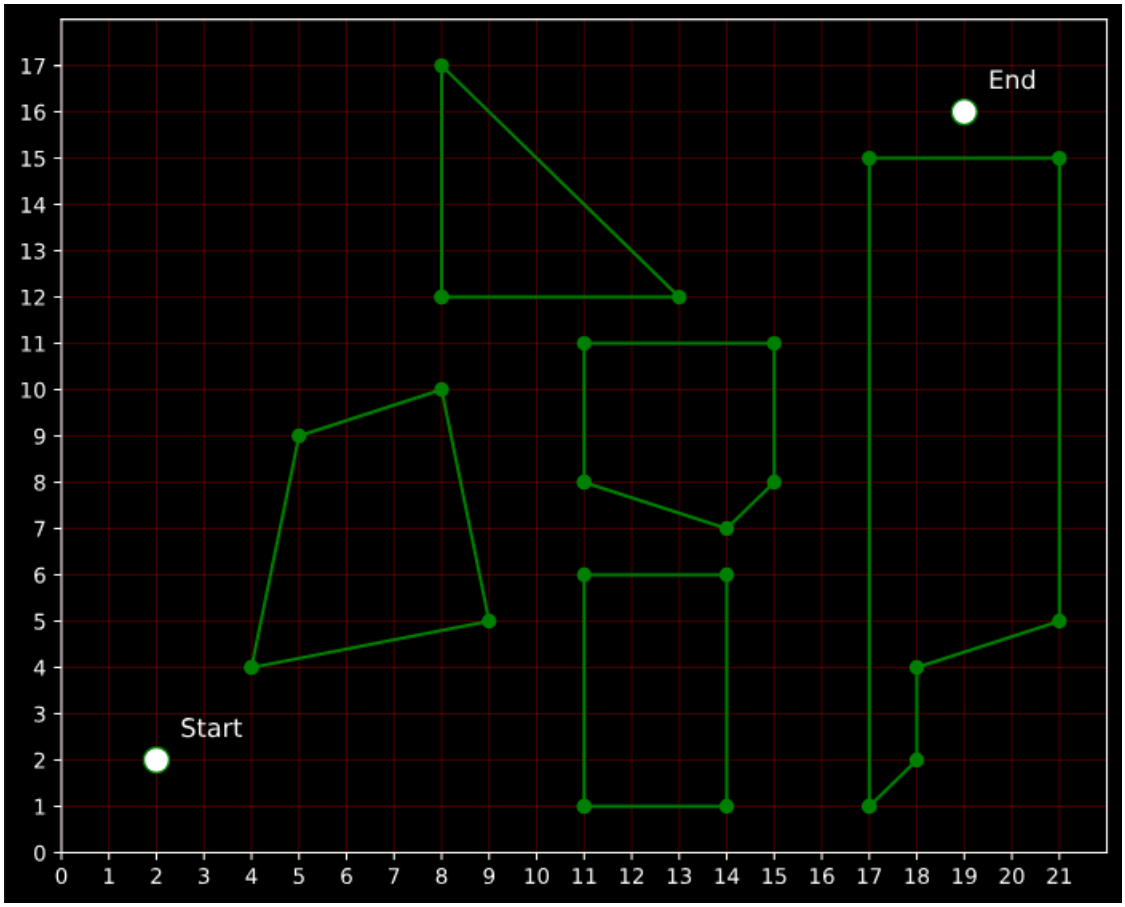
MỤC LỤC

I. Nội dung thực hiện

1. Phát biểu bài toán:

Cho một bản đồ phẳng xOy (góc phần tư I), trên đó người ta đặt một điểm bắt đầu S(x_S, y_S) và một điểm đích đến G(x_G,y_G). Đồng thời đặt các chướng ngại vật là các hình đa giác lồi sao cho các đa giác không được đặt chồng lên nhau hay có điểm chung. Không gian bản đồ được giới hạn trong một khung hình chữ nhật có góc trái dưới trùng với gốc tọa độ, độ dày của khung là 1 đơn vị. Không có điểm nào trong bản đồ được vượt hay đè lên khung này.

Chọn và cài đặt các thuật toán để tìm kiếm đường đi ngắn nhất từ S đến G sao cho đường đi không được cắt xuyên qua các đa giác.

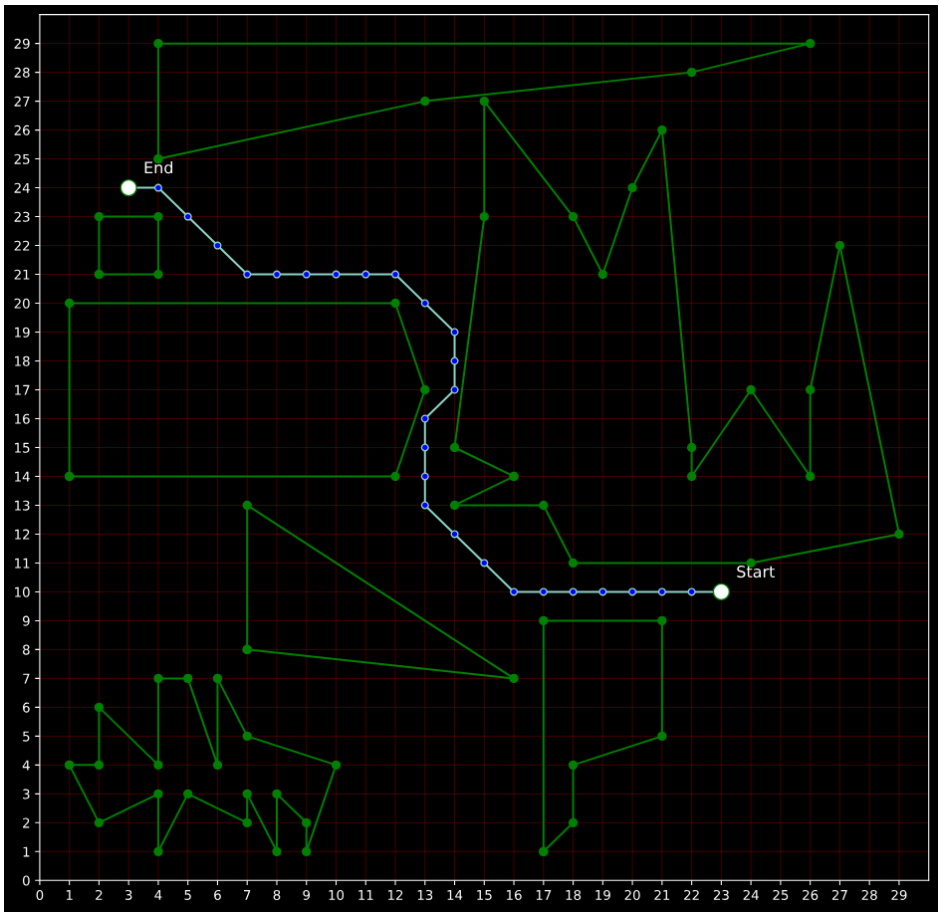


2. Các thuật toán sử dụng

Bao gồm: BFS (Breadth-First-Search), Dijkstra Heap, A Star (A*), Quy hoạch động trạng thái, Ray-casting.

a) Dijikstra Heap

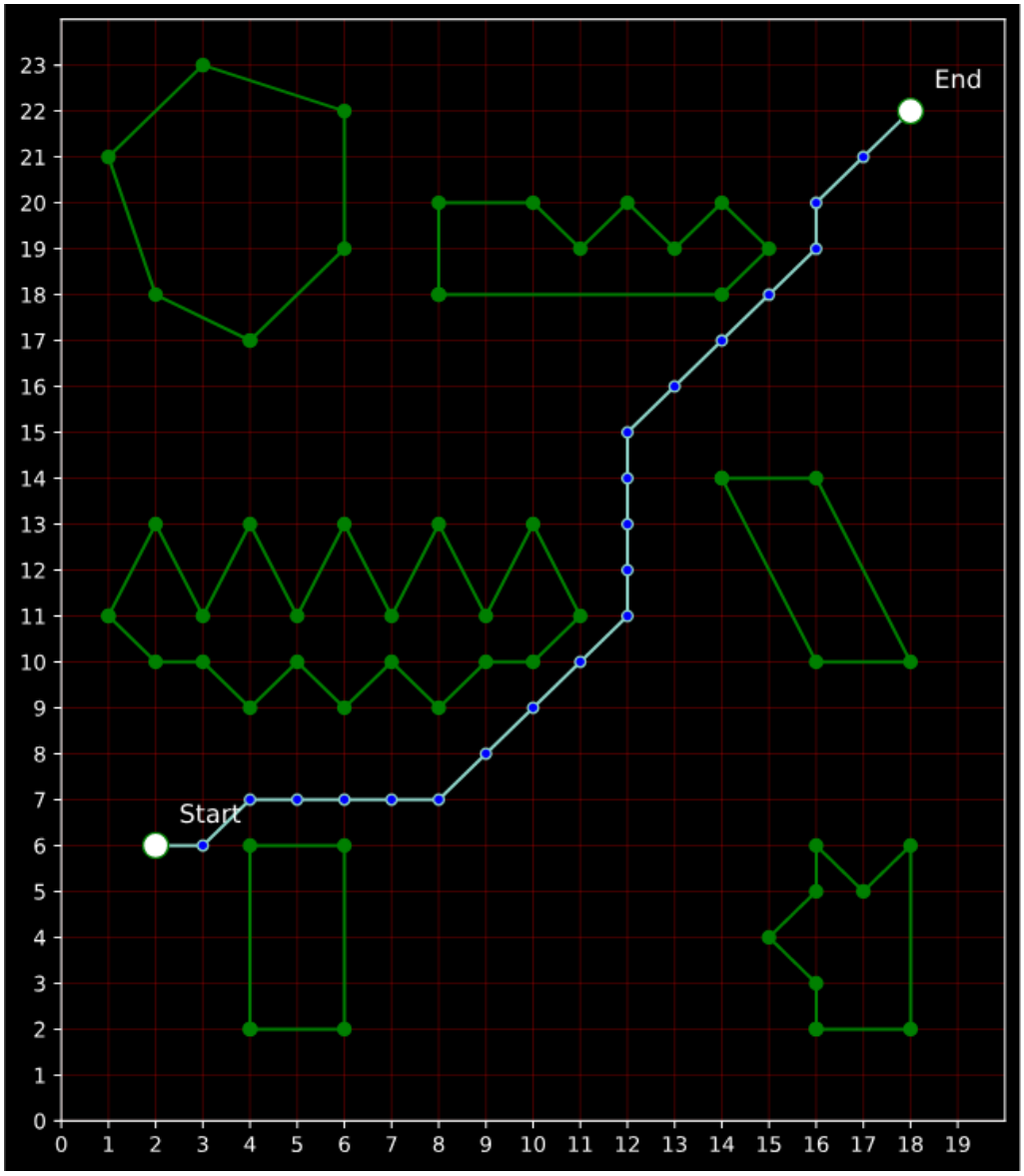
- Mô tả: Với mỗi đỉnh i, gọi distance[i] là khoảng cách ngắn nhất từ S → i. Mỗi bước lấy ra đỉnh i sao cho distance[i] nhỏ nhất và cố định đỉnh i (mỗi đỉnh chỉ được lấy 1 lần). Từ i, xây dựng lại distance[j] với j là đỉnh kề với i. Cho đến khi tìm được i = End thì kết thúc. Kết hợp với cấu trúc Heap để tối ưu hóa tốc độ xử lý.
- Độ phức tạp: O(mn.log(mn)) với m,n là kích thước bản đồ
- Run test: Hình ảnh khi chạy thuật toán



b) BFS (Breadth-First-Search)

- Mô tả: Gọi Queue là mảng chứa các điểm được sinh ra trong quá trình duyệt đường đi. Đầu tiên, Queue chỉ chứa điểm S, sau đó sinh ra được các điểm kề cận có thể đi (8 hướng) và bỏ vào trong Queue. Từ các điểm mới được sinh ra trong Queue, ta lại tiếp tục sinh ra được các điểm kề cận của những điểm đó và lại bỏ vào Queue (lưu ý mỗi điểm chỉ được bỏ duy nhất 1 lần). Cứ lặp lại quá trình cho đến khi đỉnh G được bỏ vào Queue thì kết thúc hoặc Queue không được sinh thêm bất kì điểm nào nữa (không tồn tại đường đi).
- Độ phức tạp: O(8mn) với m,n là kích thước bản đồ

- Run test: Hình ảnh khi chạy thuật toán



c) A Star (A*)

- Mô tả: Sử dụng thuật toán của DijkstraHeap và tối ưu hóa bằng việc dự đoán đường đi ngắn nhất từ điểm bất kì đến được đích End. Từ đó hạn chế việc di chuyển vào những điểm không cần thiết. Việc dự đoán đường đi ngắn nhất được thể hiện qua hàm Heristic tính khoảng cách Euclid (khoảng cách đường chim bay).

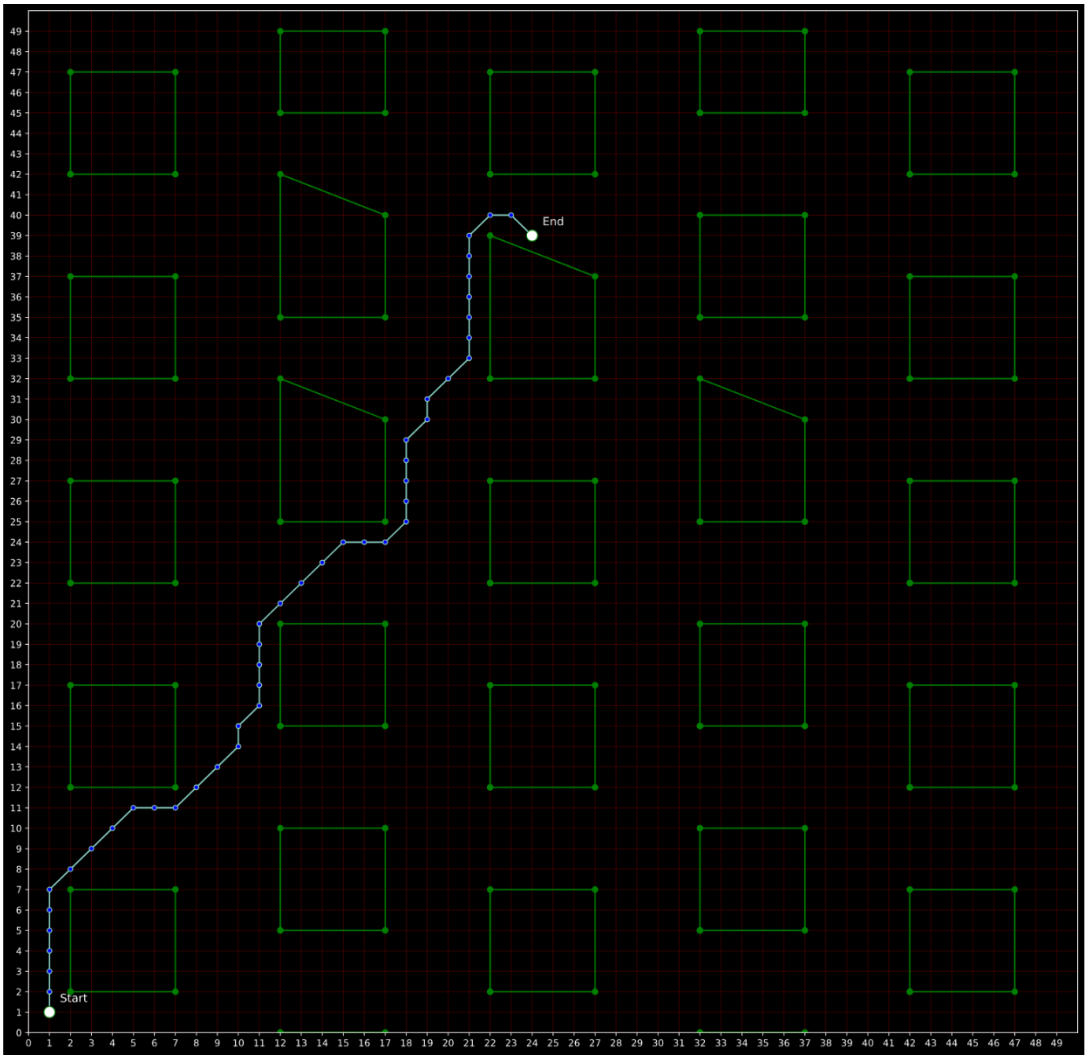
  Gọi hàm f(x) = g(x) + h(x) trong đó:

  g(x) là khoảng cách thực sự từ S → x.

  h(x) là khoảng cách dự đoán từ x → G.

  Xây dựng cấu trúc Heap chứa f(x) và sử dụng thuật toán Dijkstra vào để tìm đường đi ngắn nhất từ S → G.

- Độ phức tạp: O(mn.log(mn)) với m,n là kích thước bản đồ
- Run test: Hình ảnh khi chạy thuật toán



d) Quy hoạch động trạng thái

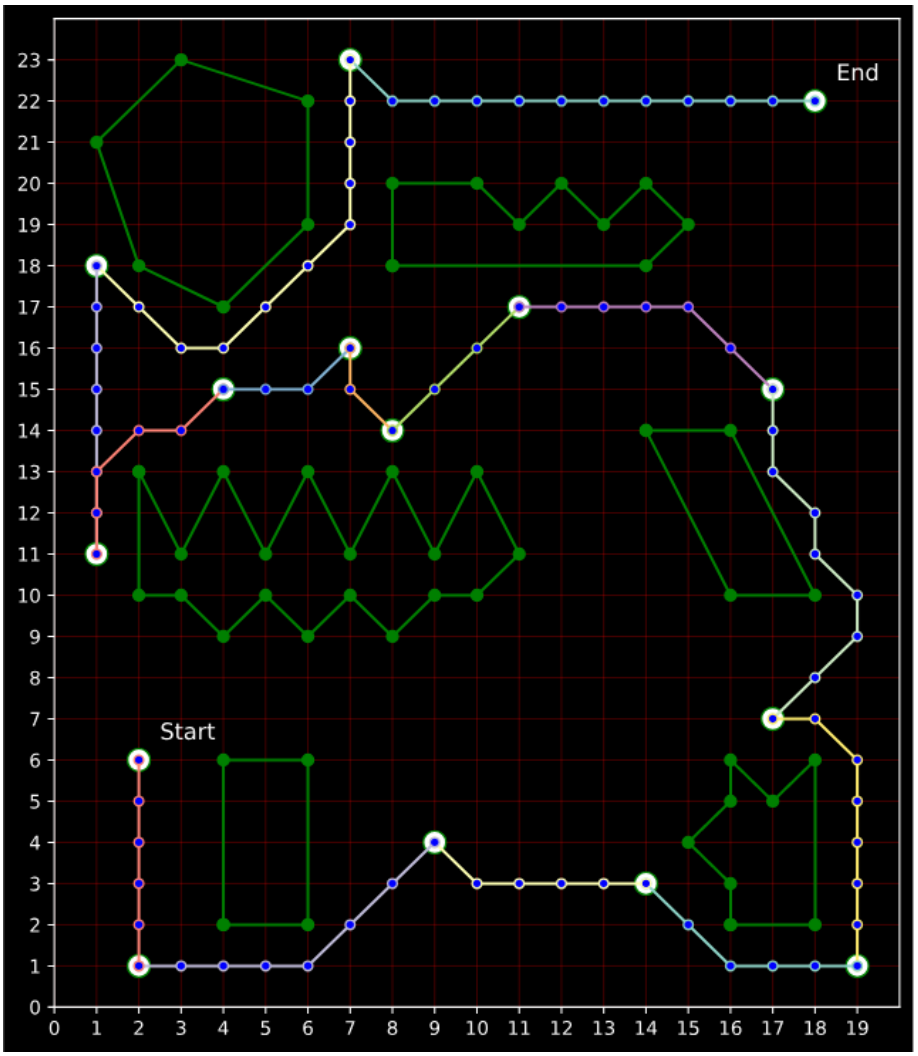
- Mô tả: Đây là thuật toán giải quyết việc có những điểm đón bắt cuộc cần đi qua bên cạnh điểm đầu S và điểm cuối G. Với mỗi trạng thái từ 0 → 2^{n-1} ta được một dãy n bit tương ứng với n đỉnh cần đi qua. Mỗi vị trí i trong dãy bit nhận giá trị 0 nếu đỉnh thứ i chưa đi qua và ngược lại nhận giá trị 1 nếu đã được đi qua. Nhận thấy rằng: mỗi khi ta bật 1 bit (từ 0 → 1) thì sẽ chuyển qua trạng thái mới với ý nghĩa ta đã đi thăm thêm 1 đỉnh chưa được thăm trước đó.

  Đặt F[x][i] là tổng giá trị đường đi sau khi đi qua k đỉnh của trạng thái x tương ứng với đỉnh i là đỉnh cuối cùng (tất nhiên bit thứ i trong x là 1). Từ đó ta được:

  F[x][i] = min(F[z][j] + distance(j, i))

với z là giá trị khi tắt bit thứ i trong x và j là các bit có giá trị 1 trong x (j<>i), còn distance là khoảng cách ngắn nhất để đi từ j tới i. Kết quả là giá trị F mang trạng thái 2^n-1 với G là đỉnh cuối cùng. (Lưu ý S là trạng thái đầu tiên).

- Độ phức tạp: $O(2^n.n^2)$ với n là số đỉnh cần đi qua (tính luôn cả điểm đầu và điểm cuối) → Thuật toán chạy khá tốt trong khoảng n nhỏ hơn 20.
- Run test: Hình ảnh khi chạy thuật toán



e) Ray-casting

- Mô tả: Đây là thuật toán để kiểm tra một điểm có nằm trong đa giác không. Ý tưởng của thuật toán: từ một điểm A đang xét, kẻ 1 tia Ax song song trục Ox, nếu tia này cắt đa giác với số giao điểm là chẵn thì điểm nằm ngoài đa giác, ngược lại điểm nằm trong đa giác. Để hiệu quả hơn cũng như không chạy sai kết quả, ta sẽ xét trước trường hợp điểm thuộc cạnh đa giác, cũng như điểm trùng đỉnh của đa giác.

- Độ phức tạp: O(n) với n là số đỉnh của đa giác, nếu có m đa giác thì độ phức tạp sẽ là m*O(n)

3. Các vấn đề giải quyết:

a) Mức 1: Cài đặt thành công 1 thuật toán

- Thuật toán: BFS
- Mức độ hoàn thành: 100% (5/5 test)

b) Mức 2: Cài đặt ít nhất 3 thuật toán khác nhau

- Thuật toán: A Star - Mức độ hoàn thành: 100% (5/5 test)
- Thuật toán: Dijikstra Heap - Mức độ hoàn thành: 100% (5/5 test)

Bảng so sánh 3 thuật toán:

- Thời gian: đơn vị ms
- Quy ước: mỗi bước đi ngang 1, mỗi bước đi chéo 1.5
- Cấu hình máy test: ram 4GB, core i5-3210M

Test	Tiêu chí	BFS	Dijikstra Heap	A Star
22x18 5 đa giác Không có đường đi	Thời gian	17.988	29.981	34.978
	Chi phí	0	0	0
30x30 7 đa giác Có đường đi	Thời gian	60.960	105.930	52.962
	Chi phí	31.5	31.5	31.5
20x24 6 đa giác Có đường đi	Thời gian	33.981	67.960	30.981
	Chi phí	26.5	26.5	26.5
50x50 25 đa giác Có đường đi	Thời gian	326.808	608.626	259.839
	Chi phí	54	54	54
100x100 4 đa giác Có đường đi	Thời gian	1108.316	2099.707	1954.798
	Chi phí	200	200	200

- Hình ảnh kết quả chứa trong thư mục OutPut12

c) Mức 3: trên bản đồ xuất hiện thêm một số điểm đón. Tìm ra cách để tổng đường đi là nhỏ nhất

- Thuật toán: Quy hoạch động trạng thái
- Mức độ hoàn thành: 100% (3/3 test)

Test	Tiêu chí	Quy hoạch động trạng thái
20x24	Thời gian	14495.080

6 đa giác 15 điểm cần đi, kể cả S và G	Chi phí	97
22x18 5 đa giác 4 điểm cần đi, kể cả S và G	Thời gian	624.618
	Chi phí	Không có đường đi
100x100 25 đa giác 18 điểm cần đi, kể cả S và G	Thời gian	89650.833
	Chi phí	212

- Hình ảnh kết quả chứa trong thư mục OutPut3

d) Mức 4: các hình đa giác có thể di động được với tốc độ h tọa độ/bước. Tìm đường đi từ S đến G.

- Thuật toán: BFS
- Quy ước bộ test:
  - Dòng đầu tiên: Không gian bản đồ
  - Dòng thứ hai: tọa độ S và G
  - Dòng thứ ba: số đa giác
  - Các dòng tiếp theo, thể hiện thông tin đa giác gồm:
    - 1 kí tự đầu tiên: hướng di chuyển đa giác (L,R,U,D)
    - 1 kí số tiếp theo: tốc độ di chuyển (h tọa độ/s)
    - 4 kí số tiếp theo: khu vực giới hạn hoạt động của đa giác, là một hình chữ nhật được xác định bằng tọa độ ($x_{\text{bottom left}}$, $y_{\text{bottom left}}$) và ($x_{\text{top right}}$, $y_{\text{top right}}$)
    - Các kí số còn lại: tọa độ (x,y) các đỉnh của đa giác

  Ví dụ:

  26,26
  2,4,22,22
  4
  R,2,1,1,25,7,4,2,4,6,8,6,10,5,12,6,16,6,16,2,12,2,10,3,8,2
  U,1,1,8,14,25,4,10,6,14,8,12,10,16,12,10
  D,1,15,16,25,25,16,22,17,23,19,23,20,22,19,21,17,21
  L,1,15,8,25,15,22,10,22,12,24,12,24,10

- Mức độ hoàn thành: 100% (4/4 test)
- Hình ảnh kết quả (các file gif) chứa trong thư mục OutPut4

4. Hướng dẫn sử dụng source code

a) File Polygon.py: chứa class cài đặt đa giác

```
__init__(self,numOfVer=3,listPoint=[],area=[],directMove=1,speed=1)
```

Khởi tạo đa giác gồm các tham số: số đỉnh, danh sách đỉnh, khu vực hoạt động (đối với mức độ 4), hướng di chuyển (đối với mức độ 4), tốc độ (đối với mức độ 4)

Các hàm bao gồm:

- isInside(Point): hàm kiểm tra một điểm có nằm trong đa giác
- getMovedListPoint(step): hàm trả về danh sách đỉnh sau khi di chuyển *step* bước, với mỗi bước có tốc độ *self.speed* đã có.
- isInsideMoved(Point,movedListPoint): hàm kiểm tra một điểm có nằm trong đa giác (đã cập nhật danh sách đỉnh sau khi di chuyển).

b) File ToolFunction.py: chứa các hàm hỗ trợ đọc file, vẽ hình, vẽ chuyển động

c) File AlgorithmFunction.py: chứa các class thuật toán BFS, Dijikstra Heap, A Star, Quy hoạch động trạng thái

d) File Main.py: chứa các hàm giải quyết các câu trả lời

```
main_12(filename,choose=1):
```

Hàm giải quyết mức độ 1, 2

choose=1,2,3 lần lượt là lựa chọn các thuật toán BFS, DijikstraHeap, A Star

Các bộ test có tên dạng: input[i].txt

Ví dụ: main_12('input1.txt',1)

```
main_3(filename):
```

Hàm giải quyết mức độ 3

Các bộ test có tên dạng: inputMultiRoute[i].txt

Ví dụ: main_3('inputMultiRoute3.txt')

```
main_4(filename):
```

Hàm giải quyết mức độ 4

Các bộ test có tên dạng: inputMove[i].txt

Ví dụ: main_4('inputMove2.txt')

II. Đánh giá kết quả của nhóm

1. Đối với nhóm

Thành viên	Nhiệm vụ	Mức độ hoàn thành
Trần Lê Bá Thịnh 1712168	Thiết kế class đa giác và thuật toán liên quan đa giác Xử lí hình ảnh và chuyển động Tổng hợp code thành hàm main	100%

Đặng Tấn Tài 1712237	Thiết kế thuật toán Dijikstra Heap, A Star và Quy hoạch động trạng thái Tester và fix bug	100%
Nguyễn Thanh Nam 1712606	Thiết kế thuật toán BFS Tạo các bộ test	100%

2. Đối với đồ án

Mức độ yêu cầu	Mức độ hoàn thành
1	100%
2	100%
3	100%
4	100%