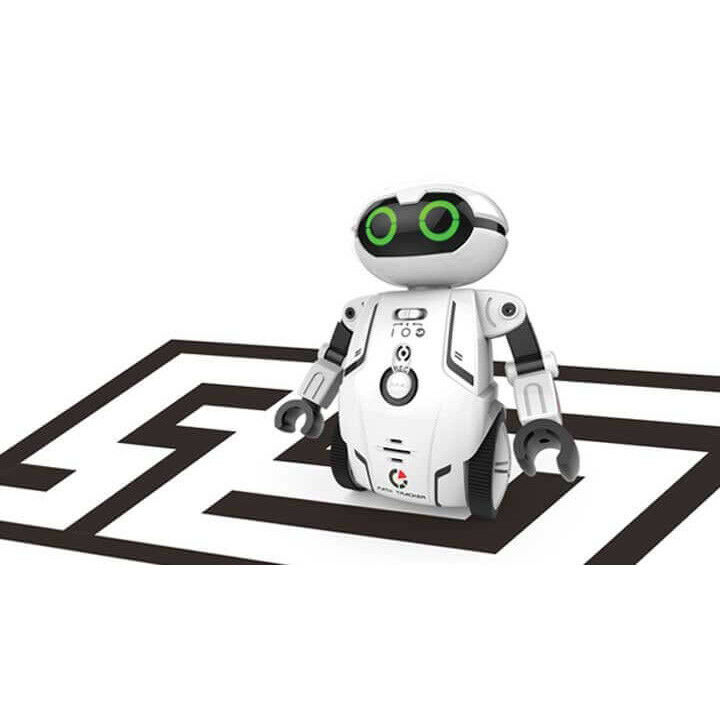
**Đại Học Quốc Gia Thành Phố Hồ Chí Minh**

**Trường Đại Học Khoa Học Tự Nhiên**

**Khoa Công Nghệ Thông Tin**

🙥🙥🙥🟍🙧🙧🙧

**BÁO CÁO ĐỒ ÁN**



***Môn học: Nhập môn Cơ sở trí tuệ nhân tạo***

***Đồ án 1 : Robot tìm đường***

*GVHD : Th.S Lê Ngọc Thành*

*Nhóm : “Ai?”*

*Lớp : Cơ sở trí tuệ nhân tạo 2017/21*

***Năm học: 2019 – 2020***

**Thông tin nhóm**

|  |  |  |
| --- | --- | --- |
| STT | MSSV | Họ Tên |
| 1 | 1712117 | Nguyễn Huỳnh Thảo Nhi |
| 2 | 1712145 | Âu Dương Tấn Sang |
| 3 | 1712171 | Phạm Hoàng Nhật Thông |

MỤC LỤC

[I. Giới thiệu đồ án: 4](#_Toc22247156)

[**1.** **Phát biểu bài toán:** 4](#_Toc22247157)

[**2.** **Dữ liệu đầu vào:** 4](#_Toc22247158)

[**3.** **Dữ liệu đầu ra:** 4](#_Toc22247159)

[II. Giới hạn bài toán: 5](#_Toc22247160)

[III. Tổ chức: 5](#_Toc22247161)

[IV. Phân công công việc: 5](#_Toc22247162)

[V. Phân tích bài toán: 5](#_Toc22247163)

[**1.** **Xử lý dữ liệu đầu vào:** 6](#_Toc22247164)

[**2.** **Ứng dụng thuật toán:** 6](#_Toc22247165)

[a. **Tìm kiếm BFS:** 6](#_Toc22247166)

[b. **Tìm kiếm UCS:** 7](#_Toc22247167)

[**c.** **Tìm kiếm A\*:** 8](#_Toc22247168)

[**d.** **Giải thuật di truyền:** 9](#_Toc22247169)

[**e.** **Các chi tiết được tối ưu để hoàn thiện bài toán:** 10](#_Toc22247170)

[VI. Hướng dẫn cài đặt và chạy chương trình: 10](#_Toc22247171)

[VII. Kết quả thực hiện: 11](#_Toc22247172)

[**1.** **Mức độ 1: Hoàn thiện** 11](#_Toc22247173)

[**2.** **Mức độ 2:** 11](#_Toc22247174)

[**a.** **Test case 1: Input mẫu** 11](#_Toc22247175)

[**b.** **Test case 2: Không có đa giác:** 12](#_Toc22247176)

[**c.** **Test case 3: Không có đường đi** 13](#_Toc22247177)

[**d.** **Nhận xét:** 13](#_Toc22247178)

[***3.*** ***Mức độ 3:*** 13](#_Toc22247179)

[**a.** **Test 1:** 13](#_Toc22247180)

[**b.** **Test 2:** 14](#_Toc22247181)

[**c.** **Test 3:** 15](#_Toc22247182)

[**d.** **Nhận xét:** 16](#_Toc22247183)

[VIII. Đánh giá: 16](#_Toc22247184)

[**1.** **Mức độ hoàn thành đồ án:** 16](#_Toc22247185)

[**2.** **Mức độ hoàn thành công việc của các thành viên:** 16](#_Toc22247186)

[IX. Phụ lục tham khảo: 17](#_Toc22247187)

1. **Giới thiệu đồ án:**
2. **Phát biểu bài toán:**

* Cho một bản đồ phẳng xOy (góc phần tư I), trên đó người ta đặt một điểm bắt đầu S(xs, ys) và một điểm đích đến G(xG,yG). 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. Đường đi có thể men theo cạnh của đa giác nhưng không được đè lên cạnh của nó. Biểu diễn đồ họa có thể ở mức đơn giản nhất để người sử dụng thấy được các đa giác và đường đi.
* Mức độ thực hiện được chia theo các mức như sau:
  + ***Mức 1 (40%):*** 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. Lưu ý, chạy thử trường hợp không có đường đi.
  + ***Mức 2 (30%):*** 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.
  + ***Mức 3 (30%):*** 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.
  + ***Mức 4 (điểm cộng 10%):*** các 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 đè lên đa giác khác. Chạy ít nhất 1 thuật toán trên đó. Quay video và đính kèm trực tiếp/link vào báo cáo.
  + ***Mức 5 (điểm cộng 10%):*** thể hiện mô hình trên không gian 3 chiều (3D).

1. **Dữ liệu đầu vào:**

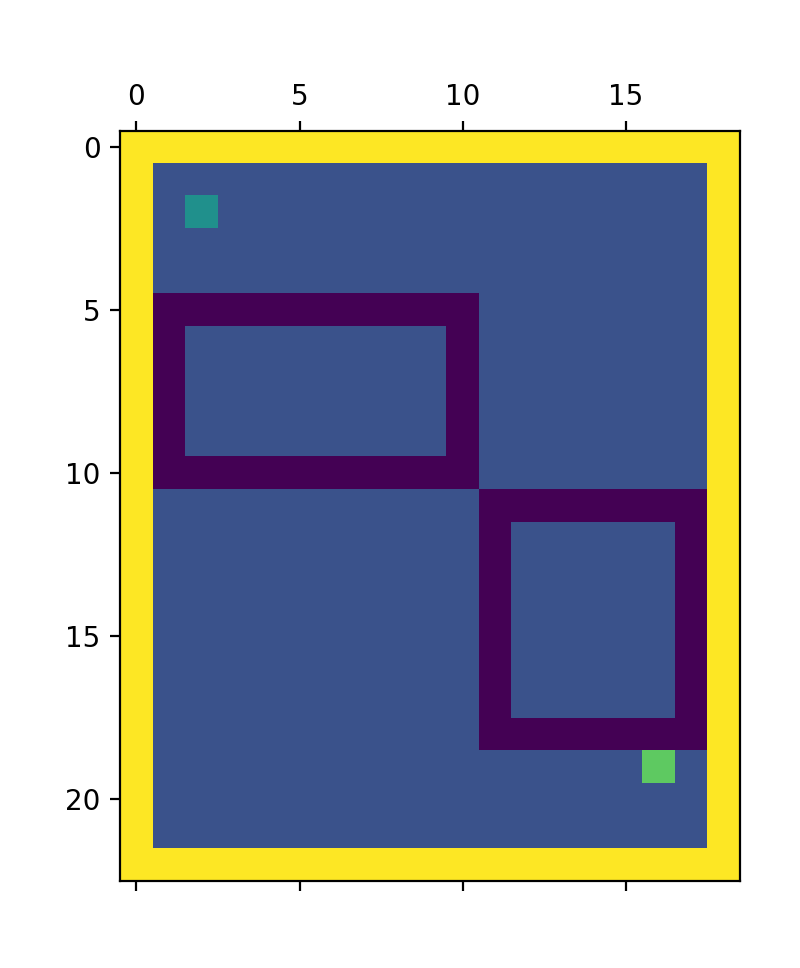
* Dòng đầu là giới hạn của không gian, được mô tả lần lượt bởi kích thước ngang, kích thước dọc và kích thước cao (nếu có)
* Dòng thức hai lần lượt là tọa độ điểm bắt đầu, tọa đọa điểm kết thúc và tập hợp các điểm đón (nếu có).
* Dòng thứ ba là số lượng đa giác có trong không gian.
* Các dòng tiếp theo, mỗi dòng chứa một đa giác theo quy tắc:

o Đa giác là tập hợp các điểm kế nhau theo chiều kim đồng hồ. Điểm cuối cùng sẽ được hiểu ngầm là sẽ được nối đến điểm đầu tiên để tạo thành một đa giác lồi hợp lệ.

* Mỗi số trong dữ liệu input cách nhau bởi dấu phẩy.

1. **Dữ liệu đầu ra:**

* Đồ hoạ biểu diễn đa giác và đường đi
* Chi phí đường đi



1. **Giới hạn bài toán:**

* Mảng chi phí đường đi thuộc kiểu số thực.
* Cho phép đi chéo với chi phí 1.5 lần đi ngang/ dọc. Cụ thể:
  + Chi phí 1 bước đi ngang/ dọc là 1.0 đơn vị
  + Chi phí 1 bước đi chéo là 1.5 đơn vị.
* Đối với trường hợp 2 đa giác chạm đỉnh với nhau, không cho phép robot đi xuyên qua điểm giao nhau. *Ví dụ: Trường hợp như hình bên sẽ không tồn tại đường đi.*

1. **Tổ chức:**

* Môi trường lập trình : Visual Studio Code
* Ngôn ngữ lập trình : Python
* Thư viện đồ hoạ : matplotlib
* Thư viện hỗ trợ :
  + heapq – Cấu trúc heap hỗ trợ thuật toán UCS.
  + random – Hỗ trợ việc sinh ngẫu nhiên dữ liệu.

1. **Phân công công việc:**

* Thời gian thực hiện : 4 tuần (23/09/2019 – 20/10/2019)
* Bảng phân công công việc :

|  |  |  |  |
| --- | --- | --- | --- |
| STT | Thời gian | Công việc | Thành viên thực hiện |
| 1 | **Tuần 1** | * Tìm hiểu yêu cầu đồ án * Thiết kế tổ chức dữ liệu * Lựa chọn phương án giới hạn bài toán | Cả nhóm |
| 2 | **Tuần 2** | * Cài đặt yêu cầu 1, 2 | Nguyễn Huỳnh Thảo Nhi |
| * Cài đặt yêu cầu 3 | Âu Dương Tấn Sang |
| * Tìm hiểu biểu diễn trực quan bằng đồ thị, tạo test case | Phạm Hoàng Nhật Thông |
| 3 | **Tuần 3** | * Test với những testcase đã thiết kế | Phạm Hoàng Nhật Thông |
| * Điều chỉnh lại code cho phù hợp | Âu Dương Tấn Sang |
| * Hoàn thiện báo cáo | Nguyễn Huỳnh Thảo Nhi |
| 4 | **Tuần 4** | * Hoàn thiện, bổ sung những thiếu sót | Cả nhóm |

1. **Phân tích bài toán:**
2. **Xử lý dữ liệu đầu vào:**

* Với dữ liệu bài toán đã cho, ta sẽ tổ chức 1 số cấu trúc dữ liệu như sau:
  + Điểm: Toạ độ của 1 điểm bao gồm (x, y). Sử dụng 1 mảng 2 phần tử với phần tử [0] là toạ độ x, phần tử [1] là toạ độ y.
  + Đa giác: Mỗi đa giác sẽ gồm n điểm (n > 2). Sử dụng 1 mảng n phần tử kiểu dữ liệu điểm.
* Từ 1 điểm có toạ độ (x, y) trên đồ thị Oxy, ta sẽ chuyển thành vị trí tương ứng trong 1 ma trận có kích thước hình chữ nhật.
* Từ tập những đa giác đã cho, t xét từng đa giác. Với mỗi đa giác, ta có tập những điểm theo chiều kim đồng hồ. Dùng thuật toán Bresenham để vẽ những đoạn thẳng từ các cặp điểm đôi một theo thứ tự kề nhau.Ta được hình dạng của các đa giác.
* Dùng 1 mảng 2 chiều để đánh dấu lại vị trí của từng dữ liệu trên ma trận như viền hình chữ nhật, đa giác, điểm bắt đầu, điểm kết thúc. Sau đó dùng thư viện matplot để mô phỏng ma trận một cách trực quan.

1. **Ứng dụng thuật toán:**
   1. ***Tìm kiếm BFS:***

* Sau khi đã xử lý dữ liệu đầu vào và với giới hạn bài toán, ta áp dụng BFS như sau:
  + Sử dụng ***queue*** để lưu trữ toạ độ được đã ghé thăm và chi phí đường đi sau khi ghé thăm toạ độ đó. Đẩy điểm bắt đầu vào queue với chi phí đi đến điểm bắt đầu là 0.
  + Dùng một mảng hai chiều ***distance[currentPoint[x][currentPoint[y]]*** tương ứng với chi phí đi từ toạ độ điểm bắt đầu Start đến điểm currentPoint. Khởi tạo mảng với gía trị 0.
  + Truy vết bằng mảng 1 chiều ***trace[point[x]][point[y]]*** tương ứng với toạ độ ghé thăm trước point. Khởi tạo mảng với giá trị 0. Sau khi duyệt qua ma trận, nếu ***trace[goal[0]][goal[1]]*** != 0 có nghĩa là có đường đi có thể đi được đến điểm ***Goal,*** ta truy vết ngược dần đến điểm ***Start*** sẽ tìm được đường đi kết quả.
  + Khi lấy ra 1 toạ độ từ ***queue***, ta kiểm tra:
    - Nếu là điểm đích thì kết thúc. Trả về giá trị của chí phí đường đi tương ứng với distance[goal[0]][goal[1]].
      * Nếu không, từ toạ độ lấy ra, loang đến 8 hướng xung quanh.
      * Kiểm tra toạ độ loang đến: Tương ứng với việc không chạm vào cạnh biên của hình chữ nhật, không đi vào cạnh của đa giác và không đi xuyên qua 2 đa giác tiếp xúc nhau.
        + Nếu có thể đi đến được, thì đưa vào ***queue*** thì lưu truy vết, cập nhật mảng chi phí ***distance***. Tiếp tục vòng lặp.
        + Nếu không thì bỏ qua và loang đến điểm kề tiếp theo.
  + Nếu đến khi duyệt hết queue mà không đi đến được điểm goal, tương ứng với distance[goal[0]][goal[1]] = 0 thì ta thông báo không tồn tại đường đi và không truy vết.
* Pseudo code:

|  |
| --- |
| **def BFS(mat, start, goal, trace):**  *Khởi tạo queue rỗng*  *Khởi tạo mảng distance[][] = 0 , trace[][] = 0*  *Đẩy (0, start) vào queue*    ***While*** *len(queue) > 0:*  *Lấy (du, u) từ queue ra*  ***If*** *(u == goal) :*  ***break***  ***For*** *v in [8 ô kề u]:  KT đi được đến u và u chưa được thăm:  Cập nhật chi phí mảng distance*  *Lưu vết mảng trace[v] = u*  *Đẩy (du+trọng số, v) vào queue*  ***return*** *distance[goal[0]][goal[1]]* |

* 1. ***Tìm kiếm UCS:***
* Thuật toán trên được áp dụng vào bài toán như sau:
  + Khá tương tự với BFS, nhưng thay vì sử dụng queue, ta sử dụng min heap để tối ưu chi phí đường đi.
  + Những cài đặt khác tương tự 2.a.
* Pseudo code:

|  |
| --- |
| **def BFS(mat, start, goal, trace):**  *Khởi tạo heap rỗng*  *Khởi tạo mảng distance[][] = 0 , trace[][] = 0*  *Đẩy (0, start) vào heap*    ***While*** *len(heap) > 0:*  *Lấy (du, u) từ heap ra*  ***If*** *(u == goal) :*  ***break***  ***For*** *v in [8 ô kề u]:  KT đi được đến u và u chưa được thăm:  Cập nhật chi phí mảng distance*  *Lưu vết mảng trace[v] = u*  *Đẩy (du+trọng số, v) vào heap*  ***return*** *distance[goal[0]][goal[1]]* |

* 1. ***Tìm kiếm A\*:***
* Sử dụng min heap tương tự như UCS để lưu chi phí để đến toạ độ đang xét.
* Ngoài chi phí như đã áp dụng ở 2 thuật toán trên, ta sử dụng thêm hàm tính khoảng cách Euclid để tối ưu hoá đường đi kết quả tìm đường.
* Pseudo code:

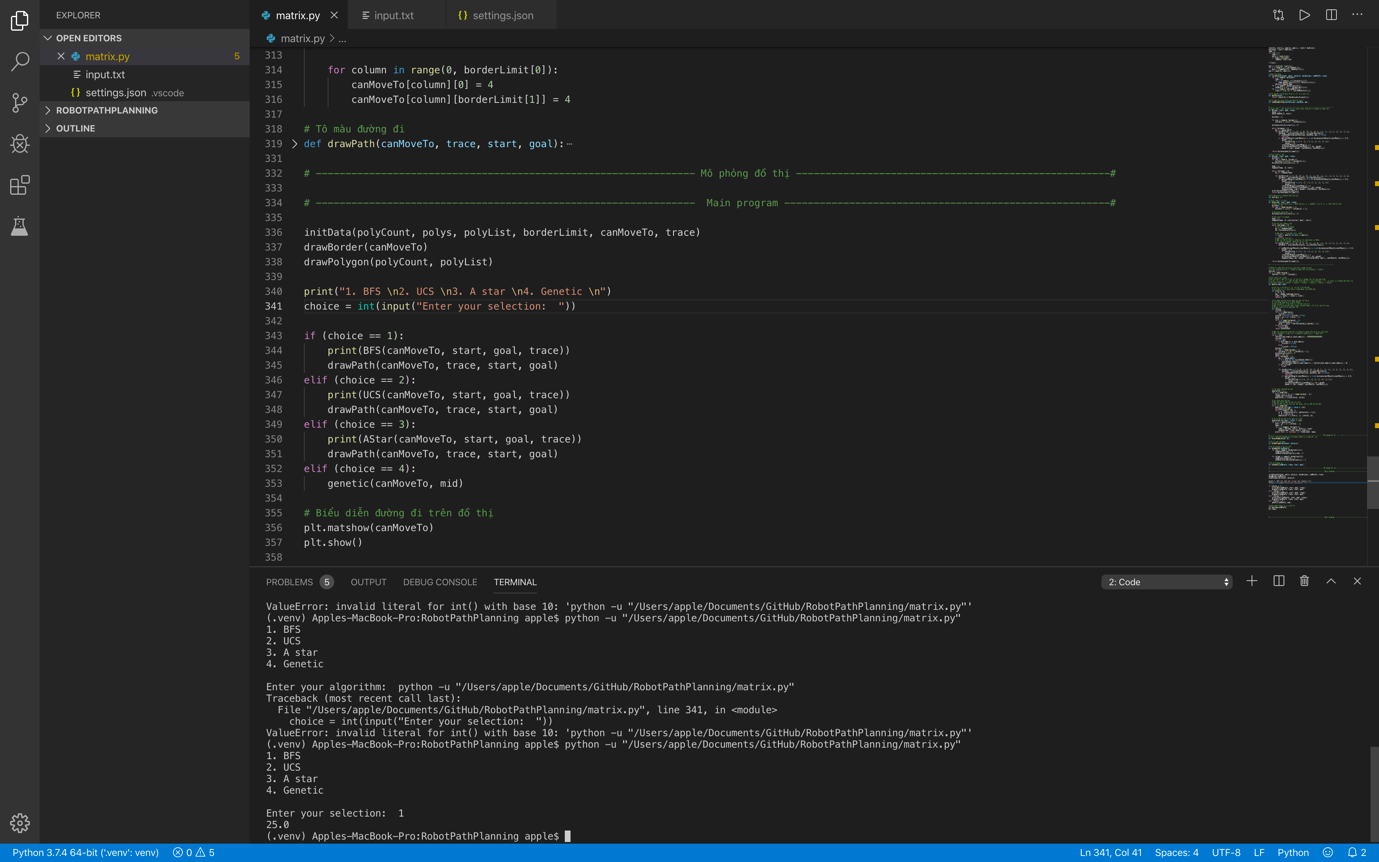
|  |
| --- |
| **def Euclid(u, v):  return** *kết quả tính khoảng cách euclid với công thức:*  **def BFS(mat, start, goal, trace):**  *Khởi tạo heap rỗng*  *Khởi tạo mảng distance[][] = 0 , trace[][] = 0*  *Đẩy (0 + euclid(u, goal), start) vào heap*    ***While*** *len(heap) > 0:*  *Lấy (du, u) từ heap ra  Cập nhật du = distance[u[0]][u[1]]*  ***If*** *(u == goal) :*  ***break***  ***For*** *v in [8 ô kề u]:  KT đi được đến u và u chưa được thăm:  Cập nhật chi phí mảng distance*  *Lưu vết mảng trace[v] = u*  *Đẩy (du+trọng số+euclid(v, goal), v) vào heap*  ***return*** *distance[goal[0]][goal[1]]* |

* 1. ***Giải thuật di truyền:***
* Ta áp dụng giải thuật để giải quyết bài toán ma trận có điểm đón như sau:
  + Đọc giá trị các điểm đón vào mảng mid từ dòng cuối cùng của file input.txt.
  + Khởi tạo mảng shortest[][] là đường đi ngắn nhất giữa mọi cặp đỉnh trong tập các điểm đón trên đồ thị.
  + Mỗi cá thể là một lộ trình, có thứ tự và không lặp lại của các đỉnh.
  + Áp dụng thuật toán BFS với queue đơn gỉan để tìm đường đi ngắn nhất giữa mọi cặp đỉnh.
  + Tạo quần thể 600 cá thể. Sử dụng hàm đánh giá độ thích nghi của một cá thể a. Tính bằng tổng độ dài đoạn đường. Hàm nếu có giá trị càng thấp tương đương với cá thể càng tốt. Nếu có hai đỉnh giống nhau trong cùng một gene, trả về số lớn vô cùng. Do mỗi đỉnh chỉ đi qua một lần.
  + Lai ghép qua 200 đời, mỗi đời bỏ đi 200 cá thể xấu nhất. 400 cá thể còn lại cho lai với nhau, tạo ra 200 cá thể mới. Lai 2 cái thể a và b, trả về 1 cá thể mới. Chọn 1 vị trí ngẫu nhiên trong gene rồi hoán đổi.
  + In ra cá thể thích nghi tốt nhất tương ứng với đường đi kết quả cần tìm. Bắt đầu từ điểm start, đi qua các điểm đón và đến điểm đích với chi phí thấp nhất có thể.
  + Pseudo Code:

|  |
| --- |
| ***def Genetic(mat, mid):  # Hàm đánh giá độ thích nghi của một cá thể***  ***def rate(a):***  *Khởi tạo mảng visited với giá trị False;  Chèn False vào đầu mảng visited, True vào cuối mảng visited   Khởi tạo đoạn đường với mảng moreA  Khởi tạo giá trị độ dài đường đi total = 0*  *f****or*** *i in range (len(moreA) – 1):  Đánh dấu đã đi qua moreA[i]  Cập nhập độ dài đường đi vào total += shortest[moreA[i]][moreA[i+1]]* ***return INF\_MAX***  ***# Hàm lai 2 cá thể a và b, trả về cá thể mới***  ***def cross(a, b):***  *Gán c,d = a, b*  *Gán pos = vị trị ngẫu nhiên trong gene*  *Hoán đổi c, d = d, c*  ***return c, d***  ***# BFS + queue để tìm đường đi ngắn nhất giữa mỗi cặp đỉnh*** ***# Cập nhật giá trị đường đi ngắn nhất giữa mọi cặp đỉnh vào mảng shortest***  ***# Tạo quần thể gồm 600 cá thể  # Lưu giá trị độ thích nghi của cá thể và cá thể vào mảng population[]*** *Dùng hàm random ngẫu nhiên cá thể  population += [[rate(child), child]]* ***#Lai ghép qua 200 đời***  ***For******i in range (200):***  *Sắp xếp population theo thứ tự độ thích nghi không tăng*  *Loại bỏ 200 cá thể xấu nhất*  ***For j in range(0, 399, 2):***  *Lai 400 cá thể còn lại với nhau*  *Tạo 200 cá thể mới* ***# In ra cá thể thích nghi tốt nhất*** |

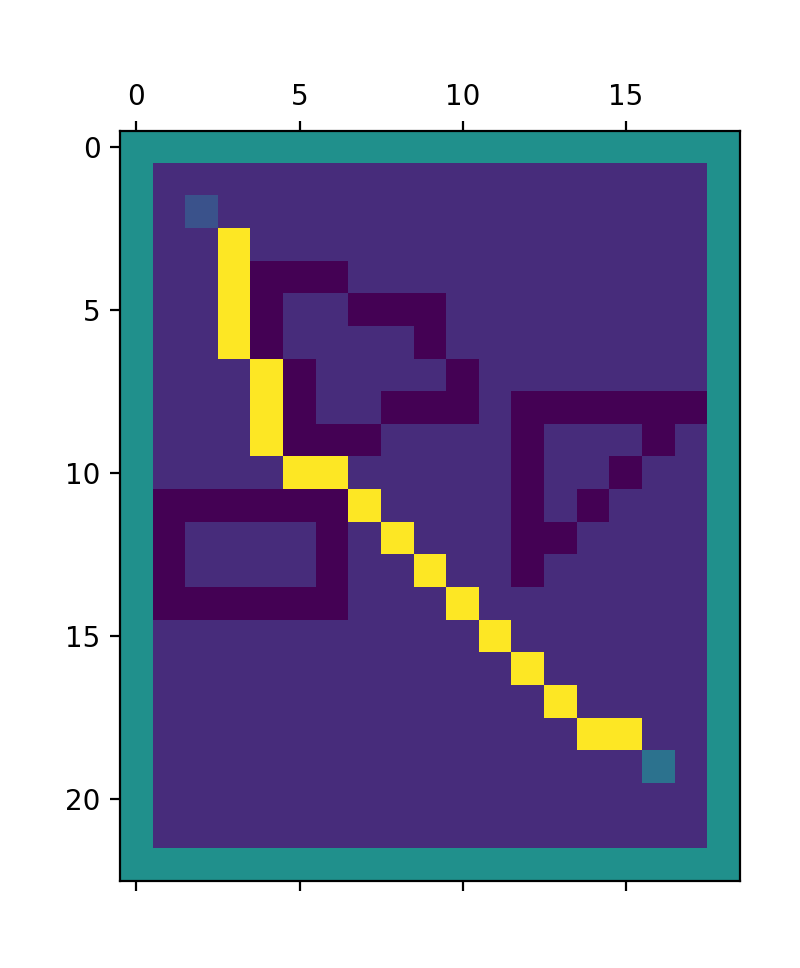
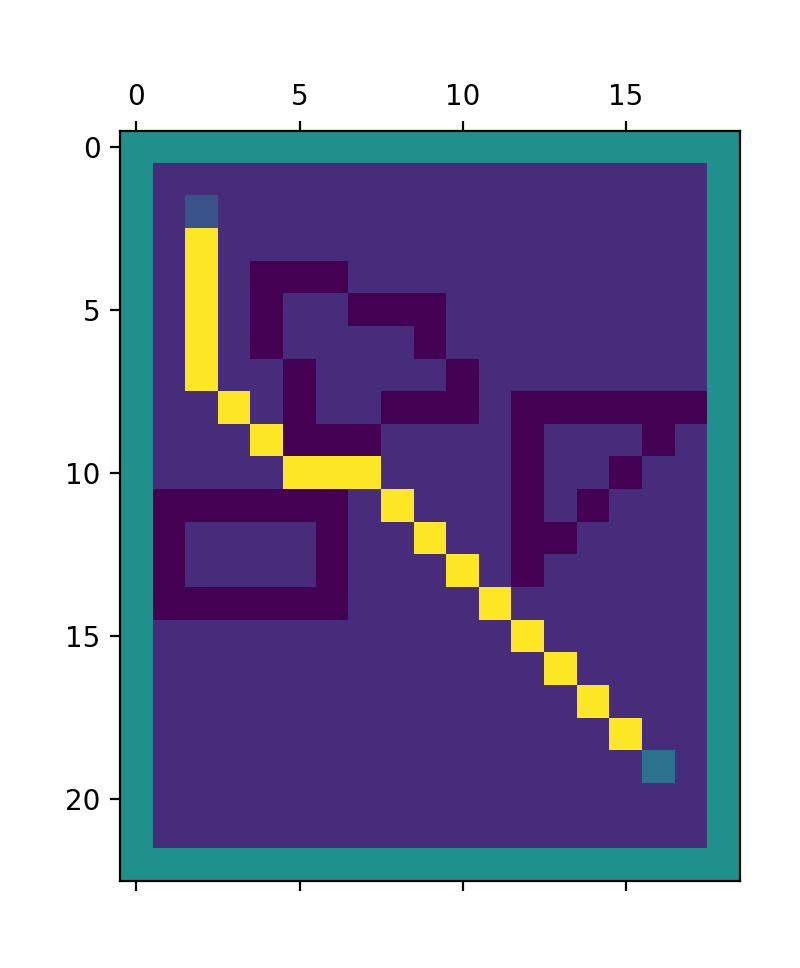
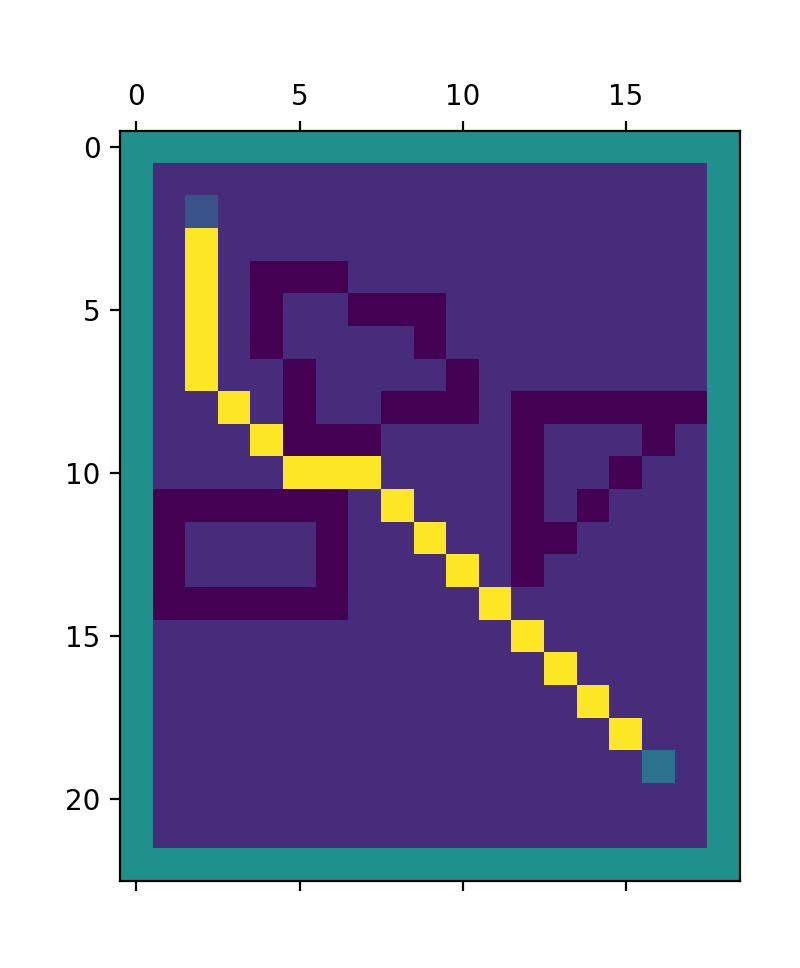
* 1. **Các chi tiết được tối ưu để hoàn thiện bài toán:**
* Kiểm tra được trường hợp không tồn tại đường đi từ điểm bắt đầu đến điểm kết thúc do các đa giác ngăn chặn.
* Kiểm tra trường hợp điểm bắt đầu và kết thúc nằm ngay trên cạnh của đa giác.
* Kiểm tra trường hợp các điểm đón nằm trên cạnh của đa giác.
* Kiểm tra 4 trường hợp có thể đi chéo được từ 4 hướng.
* Kiểm tra trường hợp điểm bắt đầu trùng với điểm kết thúc.

1. **Hướng dẫn cài đặt và chạy chương trình:**

* Tạo file input.txt có dữ liệu theo như định dạng đã nêu trên.
* Run file python
* Lựa chọn yêu cầu từ màn hình console, nhập vào số nguyên tương ứng với thuật toán được gợi ý.
* Chương trình sẽ hiển thị đồ thị minh hoạ cho ma trận và đường đi, màn hình console sẽ xuất kết quả chi phí đường đi tương ứng.
* Nếu không có đường đi, kết quả trả về 0.0.

1. **Kết quả thực hiện:**
2. **Mức độ 1: Hoàn thiện**
3. **Mức độ 2:** 
   1. ***Test case 1: Input mẫu***

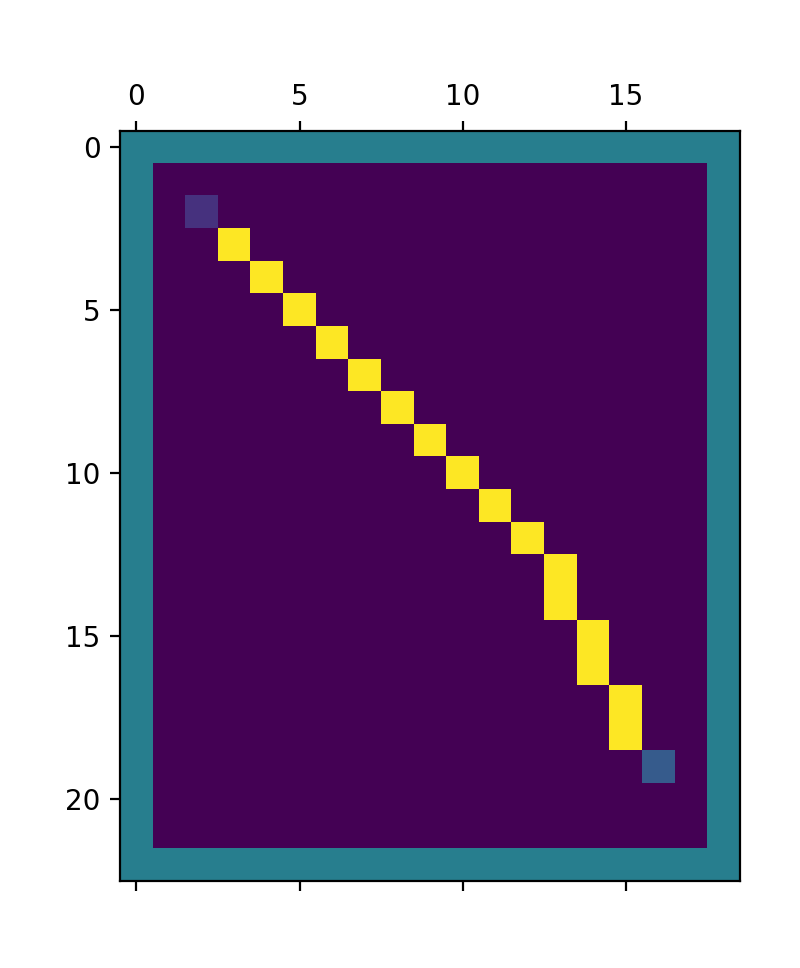
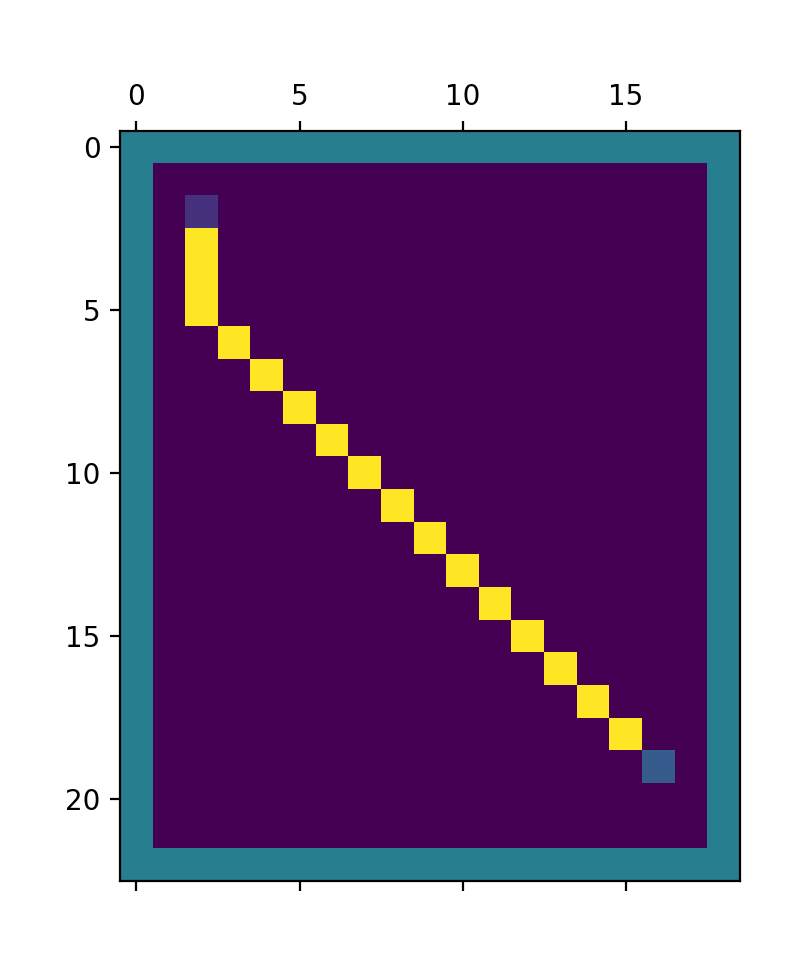
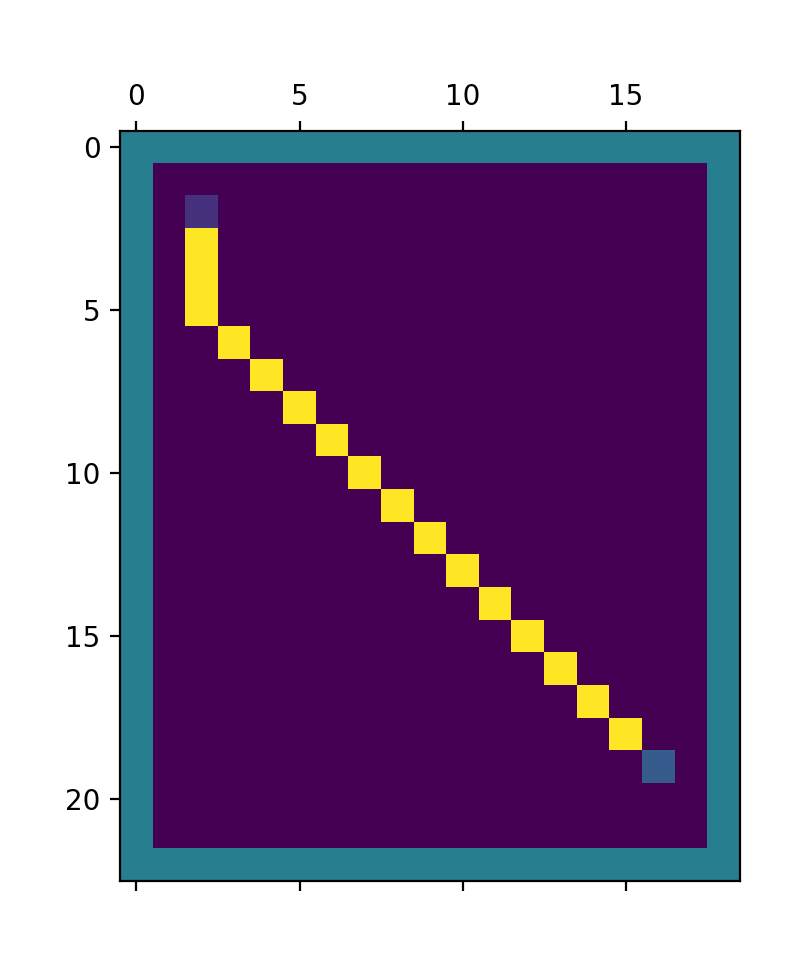
|  |  |
| --- | --- |
| ***Input.txt*** | ***Output*** |
| 22,18  2,2,19,16  3  4,4,5,9,8,10,9,5  8,12,8,17,13,12  11,1,11,6,14,6,14,1 | **25.0** |



***BFS UCS A star 25.0 25.0 25.0***

* 1. ***Test case 2: Không có đa giác:***

|  |  |
| --- | --- |
| ***Input.txt*** | ***Output*** |
| 22,18  2,2,19,16  0 | **24.0** |

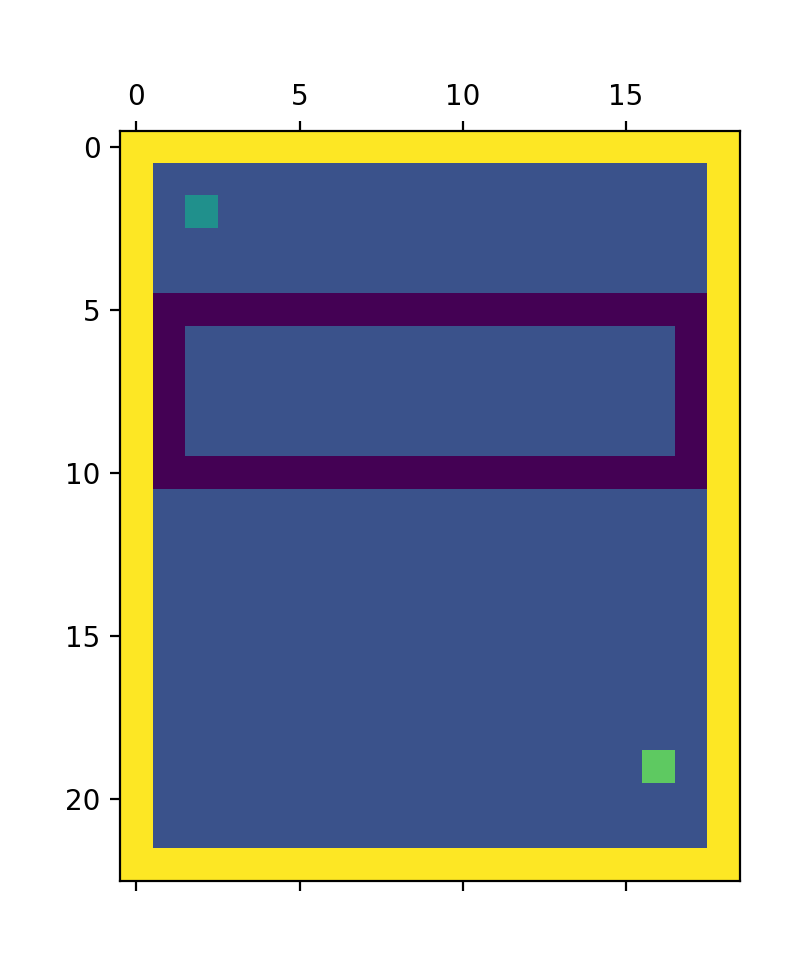
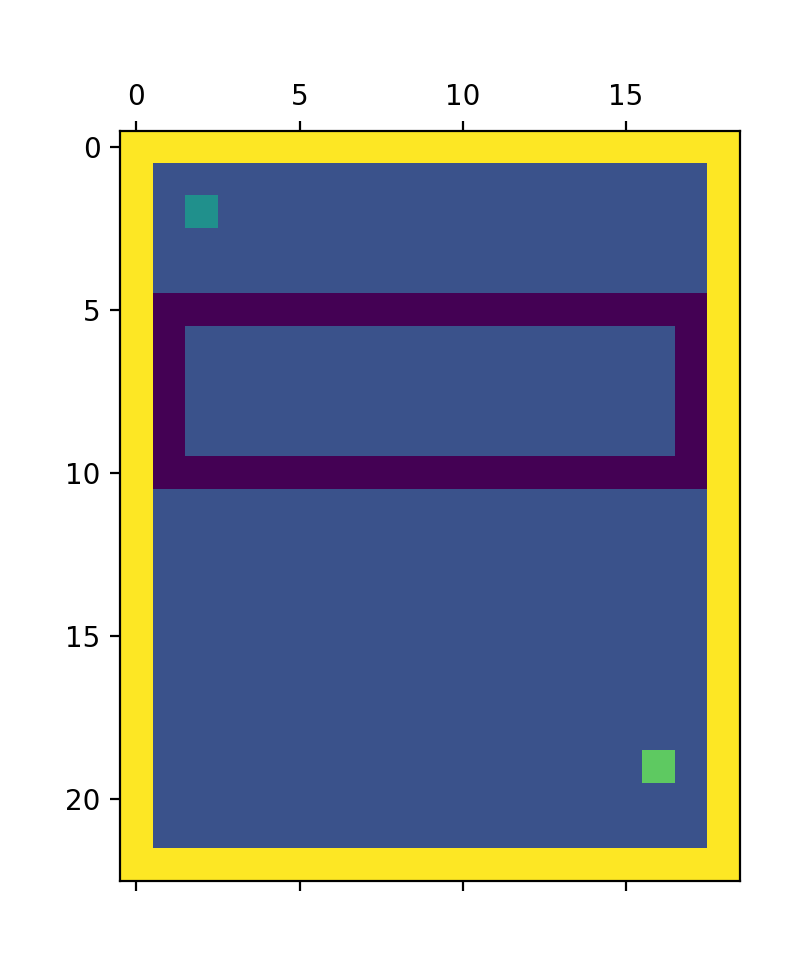
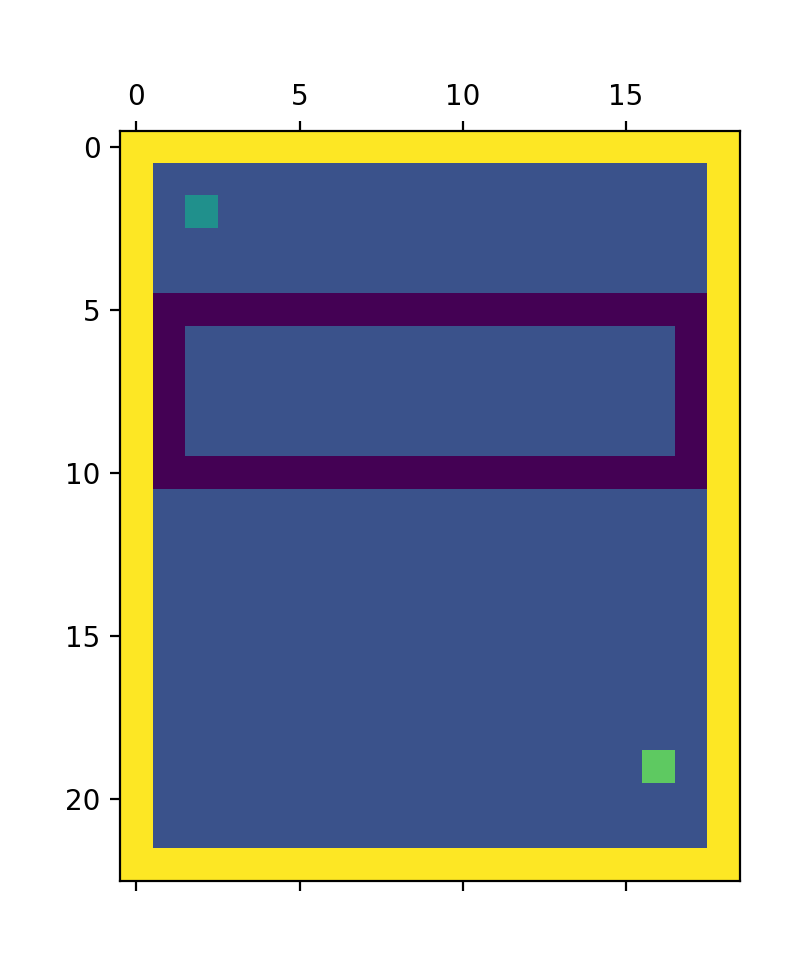


***BFS UCS A star***

***24.0 24.0 24.0***

* 1. ***Test case 3: Không có đường đi***

|  |  |
| --- | --- |
| ***Input.txt*** | ***Output*** |
| 22,18  2,2,19,16  1  10,17,5,17,5,1,10,1 | **0.0** |



***BFS UCS A star***

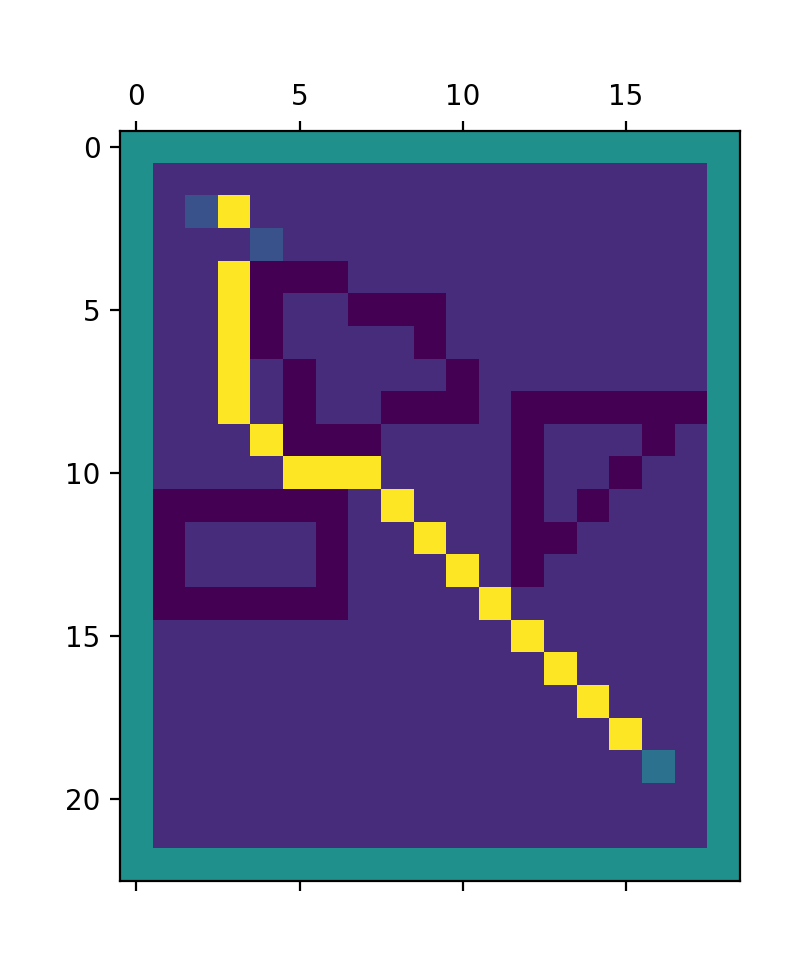
1. ***0.0 0.0***
   1. ***Nhận xét:***

* Qua thực nghiệm 3 thuật toán với 3 trường hợp dữ liệu input khác nha, ta rút ra được 1 số nhận xét:
  + Đối với bài toán tìm đường đi ngắn nhất trên ma trận 2 chiều, BFS và UCS cho ra kết quả đường đi và chi phí khá tương tự nhau vì cách tính mảng distance tương tự, chỉ khác nhau ở cấu trúc dữ liệu lưu trữ là queue và heap. Còn thuật toán A sao sử dụng heuristic để định hướng đi nên bị “thu hút” về phía toạ độ đích trong từng bước đi.

1. ***Mức độ 3:***

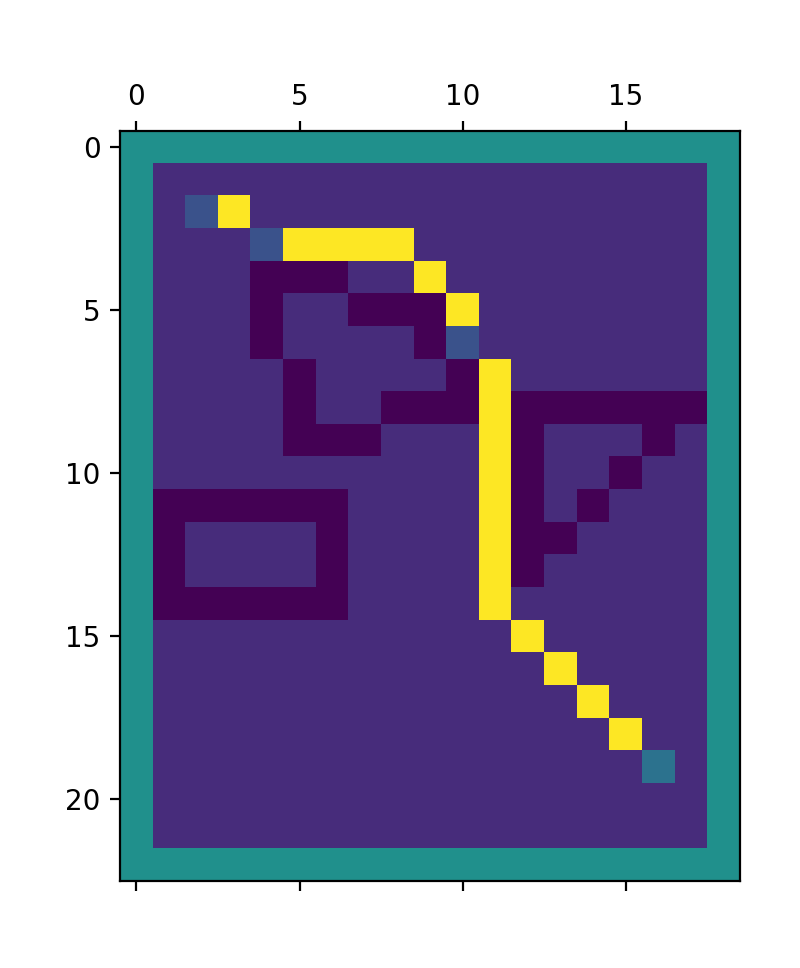
* Dữ liệu đầu vào bổ sung thêm tập những điểm đón với dạng x1,y1,x2,y2,….. vào dòng cuối cùng của file input.txt
* Dữ liệu đầu ra vẫn là đồ hoạ biểu diễn và chi phí đường đi.
  1. ***Test 1:***

|  |  |
| --- | --- |
| ***Input.txt*** | ***Output*** |
| 22,18  2,2,19,16  3  4,4,5,9,8,10,9,5  8,12,8,17,13,12  11,1,11,6,14,6,14,1  3,4 | **26.5** |

******

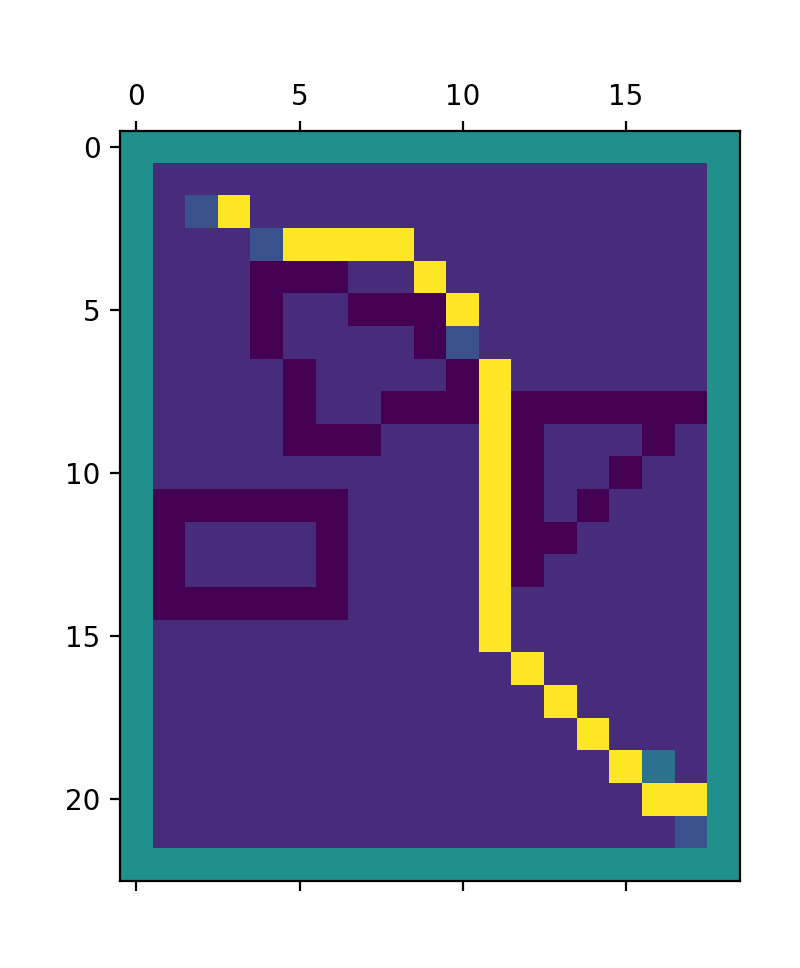
* 1. ***Test 2:***

|  |  |
| --- | --- |
| ***Input.txt*** | ***Output*** |
| 22,18  2,2,19,16  3  4,4,5,9,8,10,9,5  8,12,8,17,13,12  11,1,11,6,14,6,14,1  3,4,6,10 | **26.5** |

******

* 1. ***Test 3:***

|  |  |
| --- | --- |
| ***Input.txt*** | ***Output*** |
| 22,18  2,2,19,16  3  4,4,5,9,8,10,9,5  8,12,8,17,13,12  11,1,11,6,14,6,14,1  3,4,6,10,21,17 | **31.5** |

******

* 1. ***Nhận xét:***
* Đối với bài toán cập nhật thêm những điểm đón trên ma trận, sau khi áp dụng thuật gỉai di truyền ta có 1 số nhận xét như sau:
  + Có thể không tối ưu khi số đỉnh lớn (khoảng nửa số cá thể), nhưng luôn dừng và luôn ra được đường đi tốt nhất có thể (nếu có tồn tại đường đi).
  + Chưa code đột biến, có thể thêm đột biến để cải thiện kết quả hơn.
  + Do số lượng cá thể sinh ra và số lượng đời lai được cài đặt cố định nên không tối ưu với số đỉnh lớn. Nếu tuỳ chỉnh được số lượng cá thể và đời lai theo số lượng đỉnh ( số đỉnh tăng thì số cá thể phải tăng, phải lai nhiều hơn) thì tỉ lệ kết quả sai sẽ thấp hơn. Tuy nhiên, việc cài đặt cố định số cá thể và đời lai lại được lợi về mặt thời gian.

1. **Đánh giá:**
2. **Mức độ hoàn thành đồ án:**

|  |  |  |  |
| --- | --- | --- | --- |
| STT | Mức độ yêu cầu | Mức độ hoàn thành | Ghi chú |
| 1 | 1 | 100% |  |
| 2 | 2 | 100% |  |
| 3 | 3 | 100% |  |
| 4 | 4 | Không thực hiện |  |
| 5 | 5 | Không thực hiện |  |

1. **Mức độ hoàn thành công việc của các thành viên:**

|  |  |  |
| --- | --- | --- |
| STT | Thành Viên | Mức độ hoàn thành |
| 1 | Nguyễn Huỳnh Thảo Nhi | 100% |
| 2 | Âu Dương Tấn Sang | 100% |
| 3 | Phạm Hoàng Nhật Thông | 100% |

1. **Phụ lục tham khảo:**

<https://github.com/daancode/a-star/blob/master/source/AStar.cpp>

<https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/>

<https://eddmann.com/posts/depth-first-search-and-breadth-first-search-in-python/>

<https://towardsdatascience.com/matplotlib-tutorial-learn-basics-of-pythons-powerful-plotting-library-b5d1b8f67596>

<https://www.geeksforgeeks.org/python-introduction-matplotlib/>

<https://docs.python.org/2/library/heapq.html>

Cùng tài liệu và bài giảng tham khảo môn học của khoa công nghệ thông tin, trường đại học khoa học tự nhiên TP.HCM.

***-------- END --------***