**CƠ SỞ TRÍ TUỆ NHÂN TẠO**

Bài thực hành số 1

RoBot Tìm Đường

1712542 – Hoàng Anh Kiệt

1712525 - Nguyễn Ngọc Minh Khánh

1712543 – Huỳnh Trần Anh Kiệt

C:\Users\tdqua_000\Dropbox\SS-Slides\DeCuong-CDIO\Template CDIO v4.2\Templates\Hinh anh\LogoTruong.png

Khoa Công nghệ Thông tin

Đại học Khoa học Tự nhiên TP HCM

Tháng 10/2019

**CƠ SỞ TRÍ TUỆ NHÂN TẠO**

Bài thực hành số 1

RoBot Tìm Đường

1712542 – Hoàng Anh Kiệt

1712525 - Nguyễn Ngọc Minh Khánh

1712543 – Huỳnh Trần Anh Kiệt

GVHD: Ths. Lê Ngọc Thành

C:\Users\tdqua_000\Dropbox\SS-Slides\DeCuong-CDIO\Template CDIO v4.2\Templates\Hinh anh\LogoTruong.png

Khoa Công nghệ Thông tin

Đại học Khoa học Tự nhiên TP HCM

Tháng 10/2019

Mục lục

[**I.** **Những yêu cầu đã hoàn thành.** 4](#_Toc22378111)

[**II. Chi tiết các mức độ hoàn thành và thuật toán.** 4](#_Toc22378112)

[**1. Chi tiết mức 1.** 4](#_Toc22378113)

[**2. Chi tiết mức 2.** 6](#_Toc22378114)

[**3. Chi tiết mức 3.** 9](#_Toc22378115)

[**III.** **Các hàm chức năng.** 11](#_Toc22378116)

# **Những yêu cầu đã hoàn thành.**

* Hoàn thành đồ án ở mức 1, mức 2, mức 3.
* **Mức 1**: Cài đặt thành công 1 thuật toán tìm đường đi từ S -> G: thuật toán A\*.
* **Mức 2**: Cài đặt thành công 3 thuật toán khác nhau bao gồm: A\*, tham lam ( greedy ), UCS ( Uniform cost search ). Báo cáo nhận xét sự khác nhau khi chạy thử 3 thuật toán.
* **Mức 3**: Cài đặt thành công thuật toán tìm đường đi có tổng chi phí từ S -> đi qua các điểm đón tới -> G là nhỏ nhất.
* Những phần chưa làm được:
* Mức 4
* Mức 5
* Ngôn ngữ lập trình : Python
* Hàm thư viện hỗ trợ: Matplotlib ( hỗ trợ đồ họa ), numpy ( làm việc với mảng ).

# **II. Chi tiết các mức độ hoàn thành và thuật toán.**

**(\* Các từ đỉnh, điểm, nút đều cùng 1 ý nghĩa )**

## **1. Chi tiết mức 1.**

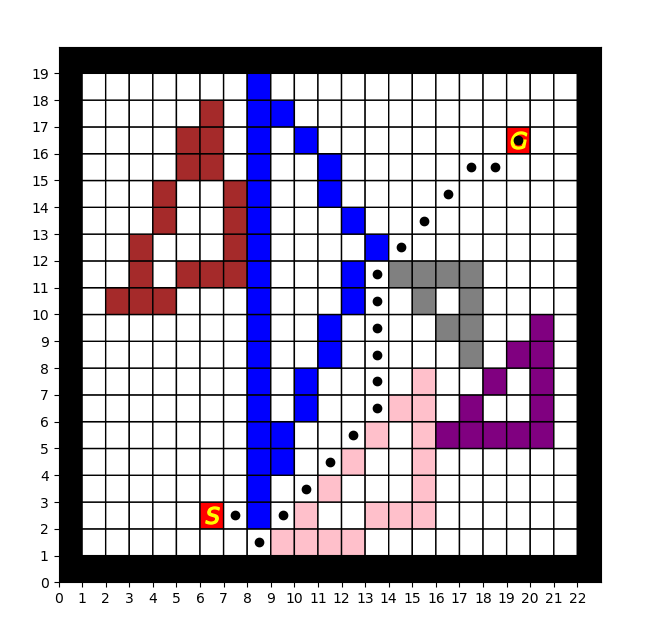
1. Thuật toán.

* Áp dụng và cài đặt thành công thuật toán A\*: Heuristic áp dụng vào thuật toán là khoảng cách **Euclidean** từ đỉnh đang xét tới đỉnh goal ( **hvalue(n)** ).
* Hàm đánh giá 1 nút ( điểm ) dựa trên chi phí đi từ nút gốc ( start ) tới điểm đó **gvalue(n)** , cộng với chi phí từ nút đó đến đích hvalue(n). Tìm lới giải tối ưu là lấy nút có giá **trị fvalue(n)= gvalue(n) + hvalue(n)** nhỏ nhất.
* Các hàm phụ: insideThePolygon(neighbor, current, grid), heuristic(p, goal), distance(p1, p2).

|  |  |  |
| --- | --- | --- |
| **Tên hàm** | **Chức năng** | **Mô tả** |
| def astar( grid, start, goal ) | * Tìm kiếm đường đi có chi phí tối ưu từ điểm start tới goal. | * Input: Tọa độ điểm start, goal. * Output: Trả về danh sách các điểm đi qua và tổng chi phí đường đi tìm được. * closeList[]: danh sách các đỉnh đã đi qua. * prev: lưu vết đường đi. * Sử dụng priority queue (openList) để lưu các đỉnh chưa xét và chi phí đỉnh đó. Các đỉnh trong queue được sắp xếp dựa trên chi phí **fvalue(n)**. * Lấy đỉnh có chi phí nhỏ nhất trong queue. Nếu là goal thì trả về đường đi ngược lại mở rộng các đỉnh liền kề vào queue ( trạng thái tiếp theo ). |

1. Nhận xét:

* Cho lời giải tối ưu.

1. Kết quả chạy thử:
   * **Chi phí:**

## **2. Chi tiết mức 2.**

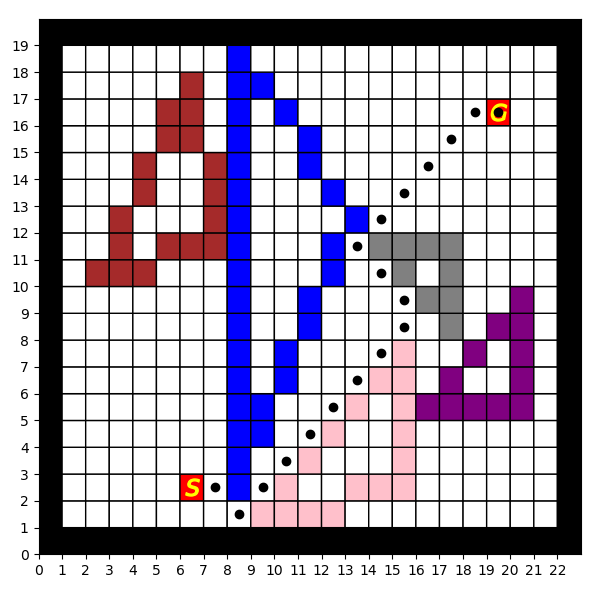
1. Thuật toán.

* Cài đặt thành công 3 thuật toán: A\*, \*, tham lam ( greedy ), UCS ( Uniform cost search)
* Thuật toán A\* như đã trình bày ở phần trên.
* *Thuật toán tìm kiếm tham lam*:
* Heuristic áp dụng vào thuật toán là khoảng cách **Euclidean** từ đỉnh đang xét tới đỉnh goal **hvalue(n)** .
* Mở rộng nút gần đích nhất, chi phí dựa trên hàm heuristic **fvalue(n)=hvalue(n)**.
* *Thuật toán UCS* :
* Hàm chi phí: là khoảng cách **Euclidean** từ đỉnh đang xét tới đỉnh start **gvalue(n)** .
* Sử dụng hàng đợi ưu tiên **( priority queue** ) để lưu nút chưa xét.
* Mở rộng các nút trong hàng đợi có hàm có hàm chi phí là nhỏ nhất.
* Các hàm phụ: insideThePolygon(neighbor, current, grid), heuristic(p, goal), distance(p1, p2).

|  |  |  |
| --- | --- | --- |
| **Tên hàm** | **Chức năng** | **Mô tả** |
| def greedy( grid, start, goal ) | * Tìm kiếm đường đi có đường đi từ điểm start tới goal. | * Input: Tọa độ điểm start, goal. * Output: Trả về danh sách các điểm đi qua và tổng chi phí đường đi tìm được. * closeList[]: danh sách các đỉnh đã đi qua. * prev: lưu vết đường đi. * Sử dụng priority queue (openList) để lưu các đỉnh chưa xét và chi phí đỉnh đó. Các đỉnh trong queue được sắp xếp dựa trên chi phí **hvalue(n).** * Lấy đỉnh có chi phí nhỏ nhất trong queue. Nếu là goal thì trả về đường đi ngược lại mở rộng các đỉnh liền kề vào queue ( trạng thái tiếp theo ). |
| def UCS(grid, start, goal) | * Tìm kiếm đường đi tối ưu từ start tới goal. | * Input: Tọa độ điểm start, goal. * Output: Trả về danh sách các điểm đi qua và tổng chi phí đường đi tìm được. * closeList[]: danh sách các đỉnh đã đi qua. * prev: lưu vết đường đi. * Sử dụng priority queue (openList) để lưu các đỉnh chưa xét và chi phí đỉnh đó. Các đỉnh trong queue được sắp xếp dựa trên chi phí **gvalue(n).** * Lấy đỉnh có chi phí nhỏ nhất trong queue. Nếu là goal thì trả về đường đi ngược lại mở rộng các đỉnh liền kề vào queue ( trạng thái tiếp theo ). |

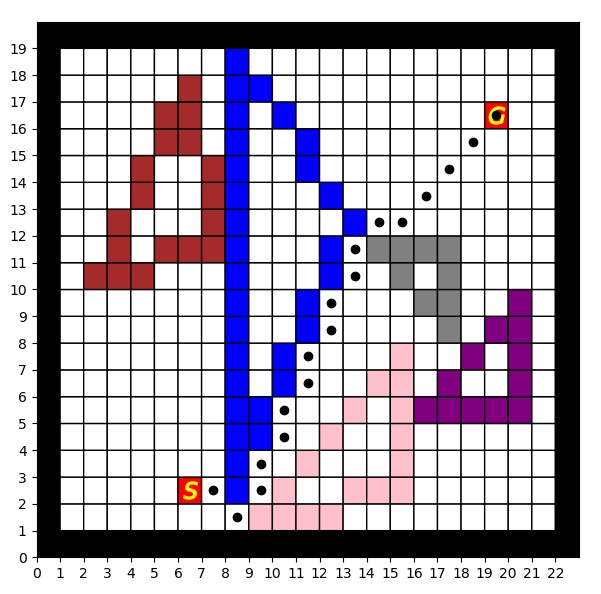
1. Nhận xét: ????
2. Kết quả chạy thử nghiệm.

* Thuật toán A\* : như đã trình bày ở phần trên
* Thuật toán greedy.





* + **Chi phí:**
* Thuật toán UCS.



* + Chi phí:

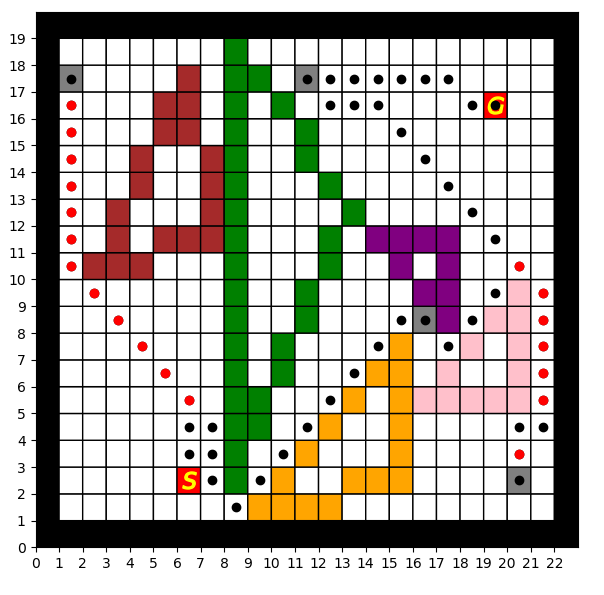
## **3. Chi tiết mức 3.**

1. Thuật toán.

* Cài đặt thành công thuật toán tìm đường đi từ điểm start -> qua các điểm đón -> goal có chi phí tối ưu.
* Thuật toán áp dụng:
* Heuristic áp dụng vào thuật toán là khoảng cách **Euclidean** từ đỉnh đang xét tới đỉnh goal **hvalue(n).**
* Hàm chi phí: dựa trên chi phí đi từ nút hiện hành tới điểm đó **gvalue(n)** trừ cho chi phí từ nút đó đến đích **hvalue(n)**.
* Coi các điểm là như nhau. Mục tiêu là từ điểm hiện hành ta biết được điểm đón tiếp theo cần đi tới mà chưa nằm trong tập visited.
* Các hàm phụ: costOfPath(path, start), heuristic(p, goal), greedy( grid, start, goal )

|  |  |  |
| --- | --- | --- |
| **Tên hàm** | **Chức năng** | **Mô tả** |
| def shorttestPath ( grid, start, goal ) | * Tìm kiếm đường đi có chi phí tối ưu từ điểm start tới goal. | * Input: Tọa độ điểm start, goal. * Output: Trả về danh sách các điểm đi qua và tổng chi phí đường đi tìm được. * visited[] là danh sách các điểm đón đã đi qua. * Dùng priority queue để lưu chi phí từ điểm hiện hành cho tới các điểm đón chưa nằm trong tập visited . Điểm đón tiếp theo được đi tới là điểm có chi phí nhỏ nhất ( độ ưu tiên cao nhất ). |

1. Nhận xét: ????
2. Kết quả chạy thử nghiệm.





* + Chi phí:

# **Các hàm chức năng.**

|  |  |  |
| --- | --- | --- |
| **Tên hàm** | **Chức năng** | **Mô tả** |
| def heuristic(p, goal) | * Tính khoảng cách từ điểm đang xét tới điểm đích | * Input: Tọa độ điểm p, goal. * Output: khoảng cách đường đi từ p tới goal. * Áp dụng công thức Euclidean cho tính khoảng cách. |
| def costOfPath(path, start) | * Hàm tính chi phí đường đi. | * Input: danh sách các điểm đi qua, tọa độ điểm start. * Output: tổng chi phí đường đi. * Đi thẳng thì tính chi phí bằng 1, đi chéo tính chi phí bằng sqrt(2). |
| def distance(p1, p2): | * Khoảng cách giữa 2 điểm trên mặt phẳng tọa độ | * Input: Tọa độ điểm p1, p2. * Output: khoảng cách giữa 2 điểm. * Áp dụng công thức Euclidean cho tính khoảng cách. |
| def matchTwoPoint(p1, p2): | * Trả về danh sách các điểm để nối 2 điểm p1, p2 | * Input: Tọa độ điểm p1, p2. * Output: danh sách các điểm nối từ p1 tới p2. * Nếu 2 điểm có cùng tọa độ x hoặc y thì thêm các điểm nằm trên đường đi vào danh sách. * Áp dụng thuật toán Bresenham để nối 2 điểm tạo thành đường chéo. |
| def bresenham(p1, p2): | * Trả về danh sách các điểm để nối 2 điểm p1, p2 | * Input: Tọa độ điểm p1, p2 * Output: danh sách các điểm nối từ p1 tới p2 * Nối 2 điểm nằm trên đường chéo. |
| def drawing(grid, fi, quantity): | * Vẽ hình ( tạo bản đồ ) | * Input: mảng grid, con trỏ file fi, số lượng hình ( chướng ngại vật ). |

# **Tham khảo.**

* Tài liệu trên moodle Khoa
* <http://qiao.github.io/PathFinding.js/visual/>
* <https://www.analytics-link.com/single-post/2018/09/14/Applying-the-A-Path-Finding-Algorithm-in-Python-Part-1-2D-square-grid>