TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN, ĐHQG-HCM

KHOA CÔNG NGHỆ THÔNG TIN



**BÁO CÁO ĐỒ ÁN 1**

**CÁC THUẬT TOÁN TÌM KIẾM**

Môn học: Cơ sở trí tuệ nhân tạo

Lớp: 20TN

**Danh sác sinh viên thực hiện:**

**20120081 – Nguyễn Mậu Trọng Hiếu**

**20120351 – Phạm Hữu Phúc**

**20120599 - Phù Thị Kim Trang**

**TP.HCM, tháng 10 năm 2022**

# MỤC LỤC

[MỤC LỤC 2](#_Toc117156785)

[I. Thông tin chung và phân chia công việc 4](#_Toc117156786)

[II. Mức độ hoàn thành 4](#_Toc117156787)

[1. Đánh giá tổng thể 4](#_Toc117156788)

[2. Đánh giá chi tiết từng yêu cầu 4](#_Toc117156789)

[III. Giới thiệu tổng quan 5](#_Toc117156790)

[1. Tổng quan về đồ án 5](#_Toc117156791)

[2. Tổng quan về kỹ thuật 5](#_Toc117156792)

[IV. Trình bày thuật toán 6](#_Toc117156793)

[1. Biểu diễn trong không gian trạng thái 6](#_Toc117156794)

[a. Không gian trạng thái 6](#_Toc117156795)

[b. Xây dựng đồ thị 6](#_Toc117156796)

[2. Thuật toán tìm kiếm DFS (Depth First Search) 6](#_Toc117156797)

[3. Thuật toán tìm kiếm BFS (Breadth First Search) 8](#_Toc117156798)

[4. Thuật toán tìm kiếm UCS (Uniform-Cost Search) 9](#_Toc117156799)

[5. Thuật toán tìm kiếm tham lam (Greedy Best First Search) 10](#_Toc117156800)

[6. Thuật toán tìm kiếm A\* 10](#_Toc117156801)

[V. Kết quả thử nghiệm và nhận xét 11](#_Toc117156802)

[1. Hướng dẫn sử dụng chương trình 11](#_Toc117156803)

[2. Thử nghiệm 12](#_Toc117156804)

[**a)** **Bản đồ không có điểm thưởng:** 12](#_Toc117156805)

[**b)** **Bản đồ có điểm thưởng** 18](#_Toc117156806)

[c) Bản đồ có điểm đóng 21](#_Toc117156807)

[1. Cách tổ chức chương trình và ghi chú lập trình 25](#_Toc117156808)

[*a.* *Thư mục bài nộp bao gồm:* 25](#_Toc117156809)

[*b.* *Mô tả file run.sh* 26](#_Toc117156810)

[2. Tài liệu tham khảo 26](#_Toc117156811)

# Thông tin chung và phân chia công việc

* **Bộ môn**: Cơ sở trí tuệ nhân tạo
* **Đồ án 1**: Các thuật toán tìm kiếm
* **Giảng viên lý thuyết**: Thầy Lê Hoài Bắc
* **Trợ giảng phụ trách đồ án**:
  + Nguyễn Duy Khánh ([duykhanhnguyen360@gmail.com](mailto:duykhanhnguyen360@gmail.com))
  + Nguyễn Ngọc Băng Tâm ([bangtamnguyenn@gmail.com](mailto:bangtamnguyenn@gmail.com))

|  |  |  |  |
| --- | --- | --- | --- |
| STT | MSSV | Họ tên | Nội dung công việc |
| 1 | 20120081 | Nguyễn Mậu Trọng Hiếu | Cài đặt |
| 2 | 20120351 | Phạm Hữu Phúc | Cài đặt |
| 3 | 20120599 | Phù Thị Kim Trang | Báo cáo |

# Mức độ hoàn thành

## Đánh giá tổng thể

Nhóm đã hoàn thành 100% yêu cầu của đồ án

## Đánh giá chi tiết từng yêu cầu

|  |  |  |
| --- | --- | --- |
| STT | Tên công việc | Độ hoàn thành |
| 1 | Mức 1a: Thiết kế đủ 5 bản đồ và hoàn thành 3 thuật toán tìm kiếm không có thông tin (bao gồm cài đặt thuật toán, báo cáo, hình vẽ minh họa). | 100% |
| 2 | Mức 1b: Hoàn thành tiếp 2 thuật toán tìm kiếm có thông tin | 100% |
| 3 | Mức 2a: Đề xuất thuật toán và các hàm heuristic phù hợp để giải quyết, thiết kế đủ 3 bản đồ và mô phỏng cách thuật toán chạy trên các bản đồ. | 100% |
| 4 | Mức 2b: Cài đặt thành công thuật toán (đảm bảo kết quả giống với mô phỏng ở mức 2a). | 100% |
| 5 | Mức 3: Đề xuất thuật toán và cài đặt thành công trường hợp có các điểm đón với tối thiểu 3 bản đồ. | 100% |
| 6 | Điểm cộng: Kịch bản nâng cấp | 100% |
| 7 | Điểm cộng: Video minh họa | 100% |
|  | Tổng điểm | 11.5/10 |

# Giới thiệu tổng quan

## Tổng quan về đồ án

Bài toán tìm đường đi ngắn nhất là một trong những bài toán phổ biến và có nhiều ứng dụng thực tế (như Google Maps, bài toán Networking, …); cũng như thường xuyên xuất hiện trong các kì thi lập trinh. Đây là một dạng bài kinh điển, được hiểu đơn giản là tìm đường đi giữa hai trạng thái hoặc nhiều trong đồ thị sao cho tổng trọng số của các cạnh tạo nên đường đi đó là nhỏ nhất.

Trong phạm vi đồ án, nhóm cần nghiên cứu và cài đặt các thuật toán tìm đường đi ngắn nhất trên một bản đồ là ma trận có kích thước cho trước với dữ liệu được đọc từ file. Các thuật toán bao gồm:

• Thuật toán tìm kiếm không có thông tin:

– Thuật toán tìm kiếm DFS (Depth First Search).

– Thuật toán tìm kiếm BFS (Breadth First Search).

– Thuật toán tìm kiếm UCS (Uniform-Cost Search).

• Thuật toán tìm kiếm có thông tin:

– Thuật toán tìm kiếm tham lam (Greedy Best First Search).

– Thuật toán tìm kiếm A\*.

* **Mục tiêu:** Cài đặt các thuật toán tìm kiếm đường đi từ vị trí xuất phát đến vị trí đích (vị trí thoát khỏi mê cung) cho tác nhân, và nếu tác nhân di chuyển vào ô điểm thưởng sẽ được giảm chi phí đường đi.
* **Yêu cầu ràng buộc:**

+ Tác nhân chỉ có thể di chuyển lên, xuống, trái, phải với chi phí bằng nhau.

+ Vị trí thoát khỏi mê cung là vị trí duy nhất trên biên của các bức tường mà tác nhân có thể thoát khỏi mê cung.

+ Đối với các ô trong bản đồ có thể di chuyển đến, chỉ có thể đi qua tối đa 1 lần.

+ Trường hợp bản đồ có điểm đón thì tác nhân bắt buộc phải đi qua tất cả các điểm đón.

## Tổng quan về kỹ thuật

* Ngôn ngữ lập trình: Python
* Phần mềm lập trình: Visual Studio Code
* Các thư viện hỗ trợ: Pygame
* Yêu cầu kỹ thuật: Python 3.5 trở lên, bao gồm công cụ cài đặt pip
* Có file requirements.txt để thuận tiện cài đặt thư viện thông qua pip

# Trình bày thuật toán

## Biểu diễn trong không gian trạng thái

### Không gian trạng thái

* Các trạng thái: Ứng với mỗi vị trí (x, y) trên ma trận
* Hành động: Di chuyển lên, xuống, trái, phải
* Mô hình chuyển tiếp: Cập nhật vị trí
* Mục tiêu: Di chuyển từ vị trí xuất phát đến vị trí đích

Kích thước không gian trạng thái của bài toán: M \* N.

### Xây dựng đồ thị

Với ma trận có kích thước M \* N, mỗi ô trong ma trận tương ứng với mỗi trạng thái của đồ thị. Ta sử dụng mảng 2 chiều để lưu giá trị của tất cả trạng thái có trong đồ thị.

Đối với mỗi trạng thái sẽ có tối đa 4 ô có thể di chuyển đến bằng hành động di chuyển lên, xuống, trái, phải. Biết rằng chi phí di chuyển đến các ô này là như nhau, đều bằng 1.

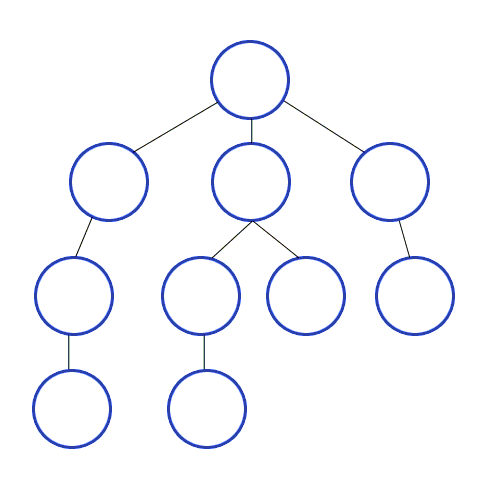
## Thuật toán tìm kiếm DFS (Depth First Search)

***Tìm kiếm theo chiều sâu*** là phương pháp tìm kiếm mà ưu tiên tìm kiếm những trạng thái xa với trạng thái xuất phát. Thuật toán DFS mở rộng tìm kiếm theo chiều sâu, nghĩa là thuật toán tìm đường theo hướng phát triển một nhánh duy nhất. Xét trên đồ thị ít vật cản hoặc không có vật cản, thuật toán sẽ tìm được đường đi hợp lệ với số lần thử ít hơn các thuật toán còn lại.

***Ý tưởng thuật toán*:** Từ trạng thái xuất phát ban đầu ta sẽ tìm được các trạng thái kề và duyệt qua các trạng thái này. Sau đó ta tìm các trạng thái kề của trạng thái vừa xét và tiếp tục duyệt đến khi đi qua hết tất cả các trạng thái có thể duyệt. Lưu ý, ta đi qua sâu vào từng nhánh của mỗi trạng thái đang xét. Thuật toán dừng khi đi qua hết tất cả các trạng thái có thể đi qua.

***Thuật toán tìm kiếm sâu:***

* Bước 1. Cho trạng thái xuất phát vào open. Lưu ý, open hoạt động theo quy tắc LIFO
* Bước 2. Nếu open rỗng thì tìm kiếm thất bại, kết thúc việc tìm kiếm.
* Bước 3. Lấy trạng thái đầu x trong open ra. Cho x vào closed.
* Bước 4. Nếu x là trạng thái đích thì tìm kiếm thành công, kết thúc việc tìm kiếm.
* Bước 5. Tìm tất cả các trạng thái con của x không thuộc open và closed cho vào đầu của open 6. Trở lại bước 2.



Hình : Minh họa thuật toán DFS (nguồn Wikipedia)

|  |  |
| --- | --- |
|  |  |
|  |  |

***Đánh giá thuật toán:***

Tổng số trạng thái cần xét tối đa là: 1 + b + b2+…+bd

Tổng số trạng thái tối đa trong open là: bd

Trong đó:

+ b: số nhánh tối đa của 1 trạng thái

+ d: độ sâu

Như vậy độ phức tạp về thời gian của thuật toán là O(bd ) và không gian (bộ nhớ) của thuật toán là O(bd).

Ta thấy thuật toán sẽ duyệt qua tất cả các trạng thái trong trường hợp đồ thị liên thông. Tuy nhiên, đây là thuật toán tìm kiếm mù, mang tính chất vét cạn và sẽ kém hiệu quả nếu tập trạng thái quá lớn. Các thuật toán tìm kiếm không thông tin không có thêm thông tin về trạng thái hoặc không gian tìm kiếm ngoài cách đi qua cây, vì vậy nó còn được gọi là tìm kiếm mù.

## Thuật toán tìm kiếm BFS (Breadth First Search)

***Tìm kiếm theo chiều rộng*** là phương pháp tìm kiếm mà ưu tiên những trạng thái gần với trạng thái xuất phát.

***Thuật toán tìm kiếm rộng:***

* Bước 1. Cho trạng thái xuất phát vào open.
* Bước 2. Nếu open rỗng thì tìm kiếm thất bại, kết thúc việc tìm kiếm. Ở giải thuật này, open hoạt động theo quy tắc FIFO.
* Bước 3. Lấy trạng thái đầu trong open ra và gọi đó là x.
* Bước 4. Nếu x là trạng thái đích thì tìm kiếm thành công, kết thúc việc tìm kiếm.
* Bước 5. Tìm tất cả các trạng thái con của x và cho vào cuối của open
* Bước 6. Trở lại bước 2

|  |  |
| --- | --- |
| Hình : Minh họa thuật toán BFS (nguồn Wikipedia) |  |
|  |  |

***Đánh giá thuật toán:***

Tổng số trạng thái cần xét tối đa là: 1 + b + b2+…+bd .

Trong đó:

+ b: số nhánh tối đa của 1 trạng thái

+ d: độ sâu

Như vậy độ phức tạp về thời gian và không gian (bộ nhớ) của thuật toán là O(bd)

Thuật toán sẽ duyệt qua tất cả các trạng thái nếu đồ thị liên thông. Tương tự như DFS, BFS cũng là thuật toán tìm kiếm mù, mang tính chất vét cạn và cũng gặp hạn chế khi tập trạng thái lớn.

Với những ưu, nhược điểm khác nhau của DFS và BFS thì việc áp dụng các giải thuật này để phục vụ cho việc giải quyết các yêu cầu bài toán phù hợp, trong đó:

* BFS: dùng để tìm đường đi ngắn nhất trên đồ thị.
* DFS: dùng để kiểm tra có tồn tại đường đi từ trạng nguồn đến trạng thái đích được hay không?

## Thuật toán tìm kiếm UCS (Uniform-Cost Search)

***Thuật toán UCS*** là một thuật toán duyệt và tìm kiếm trên một cấu trúc cây, hoặc đồ thị có trọng số (chi phí). Việc tìm kiếm bắt đầu tại nút gốc và tiếp tục bằng cách duyệt các nút tiếp theo với trọng số hay chi phí thấp nhất tính từ nút gốc. Thuật toán sẽ tiếp tục duyệt và kết thúc khi gặp trạng thái đích.

***Thuật toán tìm kiếm chi phí cực* *tiểu*:**

Bước 1. Cho đỉnh xuất phát vào open.

Bước 2. Nếu open rỗng thì tìm kiếm thất bại, kết thúc việc tìm kiếm.

Bước 3. Lấy đỉnh đầu trong open ra và gọi đó là x. Cho x vào closed

Bước 4. Nếu x là đỉnh đích thì tìm kiếm thành công, kết thúc việc tìm kiếm.

Bước 5. Tìm tất cả các đỉnh con của x không thuộc open và closed cho vào open theo thứ tự tăng dần về khoảng cách từ đỉnh xuất phát.

Bước 6. Trở lại bước 2

Ảnh có chứa bàn

Mô tả được tạo tự động

Hình . Minh họa thuật toán tìm kiếm UCS (Nguồn Wikipedia)

***Đánh giá thuật toán:***

+ Luôn luôn tìm được đường đi từ trạng thái nguồn đến trạng thái đích, đảm bảo tìm được đường đi có “chi phí thấp nhất”.

+ Hiệu quả với bản đồ nhiều vật cản như trong mê cung, ngược lại đối với bản đồ không hoặc ít vật cản thì kém hiệu quả.

+ Khá tốn chi phí về mặt bộ nhớ và thời gian thực thi.

## Thuật toán tìm kiếm tham lam (Greedy Best First Search)

Để khắc phục tính kém hiệu quả của các thuật toán mù, ta tìm hiểu đến phương pháp tìm kiếm có thông tin sử dụng hàm heuristic để đánh giá, trước hết là thuật toán Greedy Best First Search (Greedy BFS). Từ “tham lam” trong thuật toán hiểu là chọn giải pháp tối ưu tại mỗi bước đi với hi vọng tìm được giải pháp tối ưu cho cả bài toán.

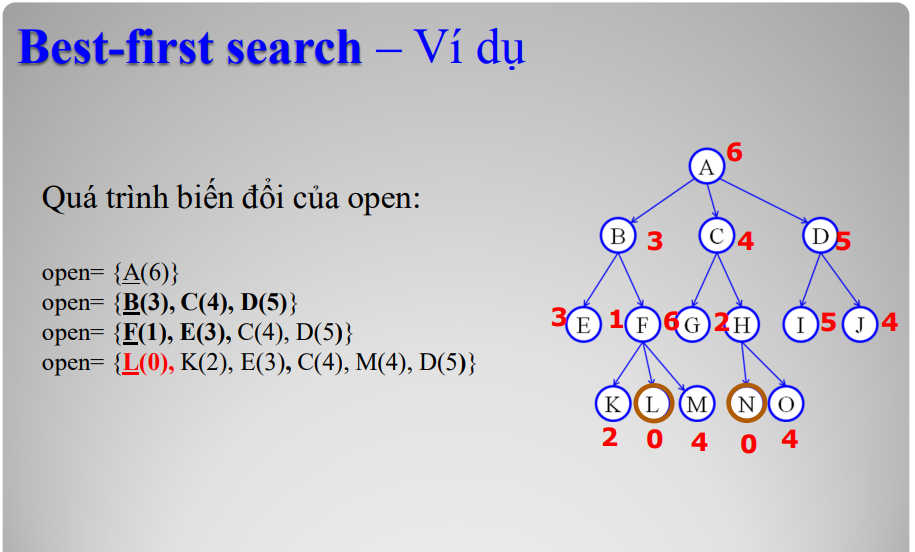
***Tư tưởng thuật toán***: Lựa chọn giải pháp nào được cho là tốt nhất ở thời điểm hiện tại và sau đó giải bài toán con nảy sinh từ việc thực hiện lựa chọn vừa rồi. Lựa chọn của thuật toán tham lam có thể phụ thuộc vào các lựa chọn trước đó. Nhưng nó không thể phụ thuộc vào một lựa chọn nào trong tương lai hay phụ thuộc vào lời giải của các bài toán con.

Thuật toán tiến triển theo kiểu thực hiện các chọn lựa theo một vòng lặp, cùng lúc đó thu nhỏ bài toán đã cho về một bài toán con nhỏ hơn. Đây cũng là sự khác biệt giữa thuật toán so với quy hoạch động.

Giải thuật tham lam quyết định sớm và thay đổi đường đi thuật toán theo quyết định đó, và không xét lại các quyết định cũ, vì vậy có thể là một thuật toán không tối ưu.

Thuật toán đánh giá chi phí các ô chỉ dựa trên chi phí từ ô đó đến ô đích:

|  |
| --- |
| f(n) = h(n) với f(n) là hàm đánh giá và h(n) là hàm heuristic. |



Hình . Mô tả thuật toán tìm kiếm tham lam

***Thuật toán tìm kiếm tham lam:***

Bước 1. Cho đỉnh xuất phát vào open.

Bước 2. Nếu open rỗng thì tìm kiếm thất bại, kết thúc việc tìm kiếm.

Bước 3. Lấy đỉnh đầu trong open ra và gọi đó là x. Cho x vào closed

Bước 4. Nếu x là đỉnh đích thì tìm kiếm thành công, kết thúc việc tìm kiếm.

Bước 5. Tìm tất cả các đỉnh con của x không thuộc open và closed cho vào open theo thứ tự tăng dần về khoảng cách ước lượng đến đích.

Bước 6. Trở lại bước 2.

***Đánh giá thuật toán:***

+ Tận dụng ưu điểm của BFS và DFS, Greedy BFS có thể chuyển đổi giữa 2 giải thuật trên, vì thế thuật toán cũng hiệu quả hơn nhưng độ hiệu quả cũng phụ thuộc lớn vào hàm Heuristic. Điểm đặc biệt là Greedy tập trung tối ưu hóa thời gian tìm đường đến đích.

+ Đường đi tìm được không đảm bảo có chi phí thấp nhất.

+ Hiệu quả với bản đồ trống, không nhiều vật cản; không hiệu quả với bản đồ nhiều vật cản và chật hẹp.

+ Tốn nhiều thời gian nếu bản đồ phức tạp nhưng tốn ít bộ nhớ

## Thuật toán tìm kiếm A\*

***Thuật toán A\*:***

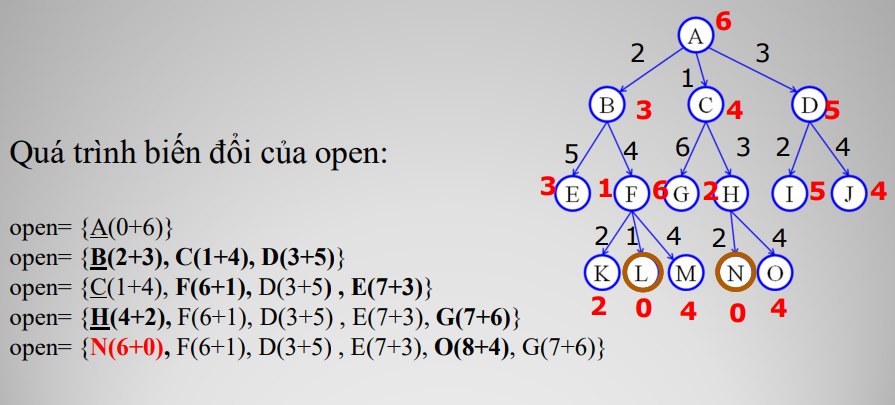
Thuật toán A\* là cũng là một trong những thuật toán Tìm kiếm có thông tin. A\* tìm đường đi từ một đỉnh hiện tại đến đỉnh đích có sử dụng hàm để ước lượng khoảng cách hay còn gọi là hàm Heuristic.

Thuật toán đánh giá chi phí các ô dựa trên tổng chi phí từ nút gốc đến nút đó và chi phí từ nút đó đến đích:

|  |
| --- |
| f(n) = g(n) + h(n) với f(n) là hàm đánh giá, g(n) là hàm chi phí thực và h(n) là hàm heuristic chi phí ước lượng. |

Biết rằng f(x) thường có giá trị càng thấp thì độ ưu tiên càng cao.

Điểm khác biệt giữa A\* và Greedy BFS là A\* tính khoảng cách đã đi qua, nên làm cho A\* tối ưu hơn và luôm tìm được đường đi ngắn nhất nếu tồn tại. Tuy nhiên, A\* không đảm bảo chạy nhanh hơn các thuật toán khác đơn giản hơn.



Hình . Mô tả thuật toán A\* (Nguồn slide thầy Phạm Minh Tuấn)

***Tư tưởng thuật toán:*** Xuất phát từ một ô và tính chi phí ước lượng của tất cả các ô lân cận (lưu ý chỉ tiếp tục duyệt đến ô kề có chi phí ước lượng tốt nhất). Lặp lại quá trình trên cho đến khi gặp ô đích hoặc không còn ô nào có thể đi.

***Thuật toán A\*:***

Bước 1. Cho đỉnh xuất phát vào open.

Bước 2. Nếu open rỗng thì tìm kiếm thất bại, kết thúc việc tìm kiếm.

Bước 3. Lấy đỉnh đầu trong open ra và gọi đó là x. Cho x vào closed

Bước 4. Nếu x là đỉnh đích thì tìm kiếm thành công, kết thúc việc tìm kiếm.

Bước 5. Tìm tất cả các đỉnh con của x không thuộc open và closed cho vào open theo thứ tự tăng dần đối với hàm f(n)=g(n)+h(n).

Bước 6. Trở lại bước 2.

***Đánh giá thuật toán:***

+ Thuật toán chắn chắn tìm được đường đi nếu có, nhưng đường đi tìm được chưa chắc là ngắn nhất (Phụ thuộc vào hàm Heuristic).

+ Hiệu quả với bản đồ trống, không nhiều vật cản và không hiệu quả với bản đồ nhiều vật cản và chật hẹp.

+ Tốn thời gian nếu bản đồ phức tạp nhưng tốn ít bộ nhớ.

+ Độ tối ưu của thuật toán phụ thuộc lớn vào hàm Heuristic.

## Thuật toán cho bản đồ có điểm thưởng – algo1

Thuật toán do chúng em đề xuất có tư tưởng giống với thuật toán A\*. Điểm khác biệt nằm trong hàm đánh giá heuristic. Với mọi đỉnh n được duyệt đến, ta không chỉ quan tâm đến f(n) = g(n) + h(n) mà còn quan tâm đến các điểm thưởng. Thực hiện duyệt qua lần lượt các điểm thưởng t, ta quan tâm đến chi phí heuristic đi từ điểm n đang xét đi đến t, cộng với giá trị thưởng tại t, rồi chi phí heuristic đi từ t đến điểm kết thúc. Tóm lại ta có công thức như sau:

|  |
| --- |
| f(n) = g(n) + min(h(n), h(n, t) + award(t) + h(t))  với mọi t là điểm thưởng chưa được duyệt, award(t) là  điểm thưởng của t , h(t) là chi phí heuristic từ t đến đích,  h(n, t) là chi phí heuristic đi từ n đến t |

Biết rằng f(x) có giá trị càng thấp thì độ ưu tiên càng cao.

***Đánh giá thuật toán:***

+ Thuật toán chắn chắn tìm được đường đi nếu có, nhưng đường đi tìm được chưa chắc là có chi phí tối ưu nhất.

+ Độ phức tạp về thời gian của thuật toán chính là độ phức tạp thời gian của thuật A\* nhân với số lượng điểm thưởng. Tuy nhiên trong giới hạn đồ án này, số điểm thưởng nằm trong khoảng vài chục và cao nhất là hàng trăm nên với giải pháp này là có thể chấp nhận được.

+ Độ tối ưu của thuật toán phụ thuộc lớn vào hàm Heuristic.

## Thuật toán cho bản đồ có điểm đón – algo2

“”

Phúc here ……………………………………………………………………..

””

Bước 6. Trở lại bước 2.

***Đánh giá thuật toán:***

+ ………………………………………………. Phúc…………………………………………..

## Thuật toán cho bản đồ có điểm dịch chuyển – algo3

Chúng em quy định đây là dịch chuyển (teleport) 2 chiều. Nếu từ (i,j) đi đến (u,v) được thì từ (u,v) cũng đến được (i,j). Ngoài ra khi chạm vào điểm teleport thì bắt buộc dịch chuyển. Chi phí để teleport là 0.

Thuật toán do chúng em đề xuất có tư tưởng giống với thuật toán A\*. Điểm khác biệt nằm trong hàm đánh giá heuristic. Cũng như algo1 – dành cho bản đồ có điểm thưởng, ta không chỉ quan tâm đến f(n) = g(n) + h(n) mà còn quan tâm đến các điểm teleport. Thực hiện duyệt qua lần lượt các điểm teleport t, ta quan tâm đến chi phí heuristic đi từ điểm n đang xét đi đến t, cộng với chi phí heuristic đi từ t’ (điểm teleport từ t dịch chuyển tới) đến điểm kết thúc. Với mọi đỉnh n được duyệt đến, ta kiểm tra xem điểm đó có phải là điểm teleport không. Nếu không ta thực hiện như thuật toán A\* bình thường với hàm heuristic trình bày như trên. Tuy nhiên nếu điểm đó là điểm teleport và vì ta quy ước bắt buộc dịch chuyển, ta cần phải lưu lại truy vết điểm trước t’ là t. Sau đó thực hiện việc đánh giá và cập heuristic như bình thường nhưng mà là cho điểm t’ chứ không phải t (vì ta đã dịch chuyển rồi). Tóm lại ta có công thức heuristic như sau:

|  |
| --- |
| f(n) = g(n) + min(h(n), h(n, t) + h(t’))  với mọi t là điểm teleport chưa được duyệt, t’ là  điểm dịch chuyển từ t , h(t’) là chi phí heuristic từ t’  đến đích, h(n, t) là chi phí heuristic đi từ n đến t |

Biết rằng f(x) có giá trị càng thấp thì độ ưu tiên càng cao.

***Đánh giá thuật toán:***

+ Thuật toán chắn chắn tìm được đường đi nếu có, nhưng đường đi tìm được chưa chắc là có chi phí tối ưu nhất.

+ Độ phức tạp về thời gian của thuật toán chính là độ phức tạp thời gian của thuật A\* nhân với số lượng điểm dịch chuyển. Tuy nhiên trong giới hạn đồ án này, số điểm thưởng nằm trong khoảng vài chục và cao nhất là hàng trăm nên với giải pháp này là có thể chấp nhận được.

+ Độ tối ưu của thuật toán phụ thuộc lớn vào hàm Heuristic.

## Các hàm đánh giá Heuristic cơ bản

Trong các thuật toán được dùng ở trên đều sử dụng các hàm heuristic hoặc từ các hàm heuristic cơ bản phát triển lên các hàm đánh giá phức tạp hơn. Trong giới hạn đồ án này, nhóm chúng em sử dụng 3 hàm heuristic cơ bản và được cài đặt trong file heuristic.py:

H1/ Euclidean distance: khoảng cách “đường chim bay” giữa 2 điểm a, b.

h(a,b) = sqrt((a1-b1)^2 + (a2-b2)^2)

H2/ Manhattan distance: hay còn gọi là “khoảng cách L1” giữa 2 điểm a, b.

h(a,b) = abs(a1-b1) + abs(a2-b2)

H3/ Chebyshev distance: còn gọi là “khoảng cách bàn cờ vua” giữa 2 điểm a, b.

h(a,b) = max(abs(a1-b1), abs(a2-b2))

Mỗi hàm heuristic đều có điểm mạnh và điểm yếu trong từng trường hợp cụ thể khác nhau sẽ được trình bày ở phần kết quả thực nghiệm của báo cáo. Với mỗi loại thuật toán cần dùng hàm heuristic đều sẽ được chạy với cả 3 hàm heuristic này.

## Hướng dẫn sử dụng chương trình

Ta chỉ cần làm việc với file main.py, nhận các argument làm tham số.

Giả sử ta đang đứng ở route, có cách tổ chức như sau:

route/

|--- input/

|--- source/

|--- run.sh

Nếu muốn, ta chỉ cần đổi thư mục input theo đúng format của giáo viên và cũng được trình bày ở phần bên dưới.

Sau đó đơn giản chạy lệnh:

bash run.sh

hoặc

sh run.sh

Chương trình sẽ tự động cài đặt các thư viện cần thiết cũng như chạy các thuật toán theo quy định, các kết quả (mỗi kết quả tương ứng gồm: 1 file .mp4 thể hiện đường đi, 1 file .jpg lưu kết quả cuối cùng, 1 file .txt chứa chi phí đường đi tốt nhất) sẽ lưu trong folder ouput như sau:

route/

|--- input/

|--- output/

|--- source/

|--- run.sh

Trong run.sh được cài đặt sẵn câu lệnh:

python source/main.py

Ngoài ra cũng có các ví dụ có sẵn để dung các tham số khác.

File main.py có các tham số:

--input-folder: đường dẫn đến input folder

--output-folder: đường dẫn đến output folder

--print-search-cost: có in ra thêm search cost không trong file txt (mặc định chỉ in best path cost)

--dont-show-video: có hiện video đường đi lên màn hình không (mặc định là có hiện lên)

--auto-close: nếu có hiện đường đi lên, thì đây là tham số số giây tự động tắt giữa các màn hình show lên.

--level: các level thực hiện, mặc định là 1, 2, 3, 4 (advance)

--path: chứa đường dẫn đến 1 file bản đồ (.txt), thay gì thực hiện tất cả các file trong thư mục input, chỉ thực hiện file này.

Cụ thể có thể coi cài đặt trong file main.py hoặc chạy: python source/main.py -h.

***Lưu ý:***

Do sự không ổn định của thư viện convert sang video, có thể trong quá trình chạy: python source/main.py có thể bị dừng giữa chừng (nhưng không báo lỗi) và không chạy hết các file trong folder input.

Hi vọng giáo viên chấm bài có thể chạy lại bằng command:

python source/main.py --dont-show-video

để quá trình chạy chương trình ổn định hơn, xin cảm ơn.

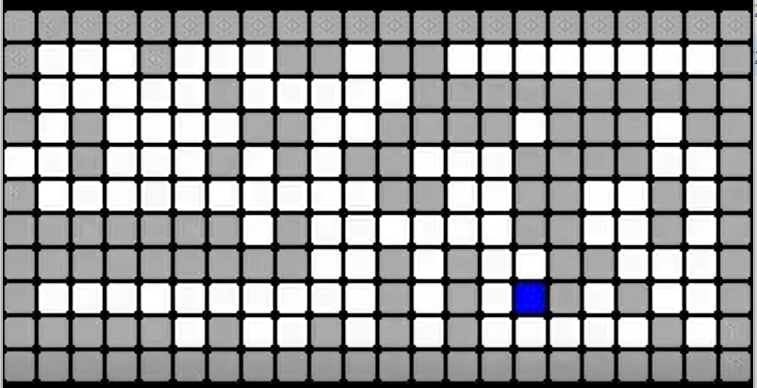
Mặc định vẫn show video đường đi lên do chúng em cũng tốn kha khá thời gian để code tính năng đó, cũng như sẽ đem lại hiệu ứng hình ảnh tốt hơn.

## Thử nghiệm

### **Bản đồ không có điểm thưởng:**

|  |
| --- |
| *Ghi chú:*   * *Ô màu xanh là ô xuất phát* * *Ô màu trắng trên tường là đích đến* * *cost: Chi phí đường đi từ nguồn đến đích, cost = NO nếu không tồn tại đường đi* * *Greedy BFS Heuristic 1, A\* Heuristic 1: Hàm Heuristic đánh giá theo* ***euclidean\_distance*** * *Greedy BFS Heuristic 2, A\* Heuristic 2: Hàm Heuristic đánh giá theo* ***manhattan\_distance*** * *Greedy BFS Heuristic 3, A\* Heuristic 3: Hàm Heuristic đánh giá theo* ***chebyshev\_distance*** |

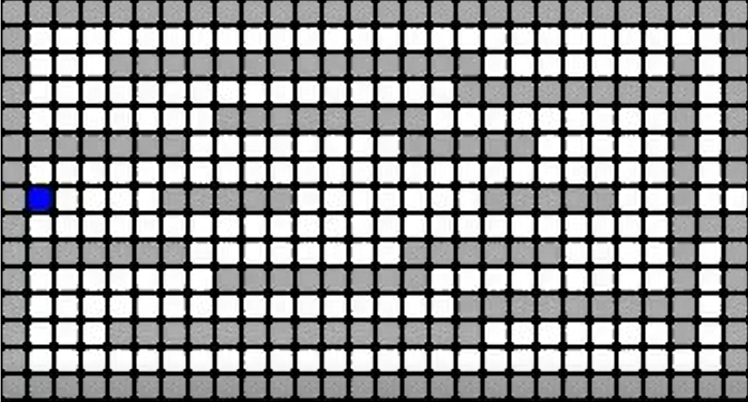
* **Bản đồ 1:**

****

|  |  |  |
| --- | --- | --- |
| Hình . DFS (cost = 19) | Hình . BFS (cost = 19) | Hình . UCS (cost = 19) |
| Hình . Greedy BFS Heuristic 1 (cost = 19) | Hình . Greedy BFS Heuristic 2 (cost = 25) | Hình . Greedy BFS Heuristic 3 (cost = 19) |
| Hình . A\* Heuristic 1 (cost = 19) | Hình . A\* Heuristic 2 (cost = 19) | Hình . A\* Heuristic 3 (cost = 19) |

***Nhận xét:***

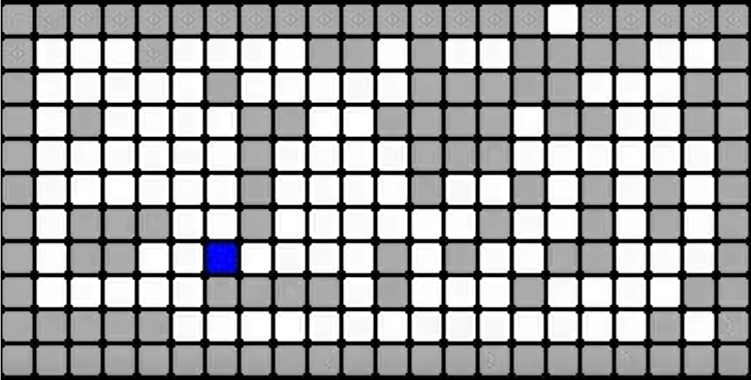
* Tính đầy đủ: Tất cả các giải thuật đều tìm được đường đi và đường đi chưa có sự khác biệt nhiều.
  + Tất cả giải thuật đều có chung tính chất sẽ tìm được đường đi từ nguồn đến đích, nếu tồn tại.
* Tính tối ưu: Nhìn chung, khi xét trên bản đồ này, các đường đi đều khá giống nhau và tối ưu về độ dài, ngoại trừ thuật toán Greedy Best First Search Heuristic 2 (manhattan\_distance) có đường đi dài nhất so với các thuật toán còn lại.
* Độ mở các nút: Đạt kết quả tối nhất là độ mở các nút của thuật toán Greedy Best First Search, hầu như tìm được đường đi trong lần duyệt đầu tiên. Ngược lại, khi xét trên 2 giải thuật BFS và UCS các nút mở tương đối nhiều gần như là tất cả, điều này thể hiện rõ việc hao tốn chi phí về bộ nhớ và thời gian chạy cho việc này. Riêng A\* có độ mở tương đối ổn định và chiếm một phần nhỏ trên cả 3 hàm Heuristic khác nhau.
* Chi phí đường đi: Trừ trường hợp thuật toán Greedy Best First Search Heuristic 2 có chi phí lớn nhất cost = 25 thì các giải thuật còn lại đều có cùng giá trị cost = 19.
* **Bản đồ 2:**

****

|  |  |  |
| --- | --- | --- |
| Hình . DFS (cost = 94) | Hình . BFS (cost = 46) | Hình . UCS (cost = 46) |
| Hình . Greedy BFS Heuristic 1 (cost = 62) | Hình . Greedy BFS Heuristic 2 (cost = 60) | Hình . Greedy BFS Heuristic 3 (cost = 82) |
| Hình . A\* Heuristic 1 (cost = 46) | Hình . A\* Heuristic 2 (cost = 46) | Hình . A\* Heuristic 3 (cost = 46) |

***Nhận xét:***

* Tính đầy đủ: Tất cả các giải thuật đều tìm được đường đi.
* Tính tối ưu: Trong bản đồ 2, các giải thuật dường như đã tỏ rõ sự khác biệt của chúng. Nếu DFS loang theo chiều sâu, phát triển một nhánh để đến được đích nên đường đi tìm được khá dài, kém hiệu quả. Ngược lại với DFS, BFS và UCS lại tìm kiếm loang theo chiều rộng nên trong trường hợp này có phần hiệu quả và tìm được tìm đi tối ưu nhất.
  + Sở dĩ, UCS tìm được đường đi khá giống với BFS trong các bản đồ là vì UCS tìm kiếm theo trọng số (chi phí) thấp của các ô kề nó, mà trong phạm vi đồ án ta xét tất cả các ô đều có trọng số bằng 1, nghĩa là chi phí bằng nhau ở mọi hướng. Mặt khác, BFS và UCS đều đảm bảo tìm được đường đi ngắn nhất nên vấn đề khá giống nhau về đường đi có thể giải thích được.
* Độ mở các nút: Các giải thuật có sự khác biệt
  + Trường hợp xấu nhất: BFS, UCS, A\* Heuristic 1 phải duyệt toàn bộ các nút để tìm được đường đi tối ưu.
  + Trường hợp tương đối xấu: DFS gần như duyệt toàn bộ các nút
  + Trường hợp còn lại: Ngoại trừ các thuật toán nêu trên, các thuật toán còn lại duyệt tương đối và hiệu quả hơn.
* Chi phí đường đi: Dựa trên từng tính chất riêng của từng loại thuật toán mà chi phí về đường đi của mỗi giải thuật có sự khác biệt:
  + Chi phí thấp nhất (cost = 46): BFS, UCS và các giải thuật A\*. Nghĩa là, các giải thuật này đảm bảo rằng nếu tồn tại đường đi từ nguồn đến đích thì đường đi tìm được là đường đi ngắn nhất, có chi phí thấp nhất.
* **Bản đồ 3:**

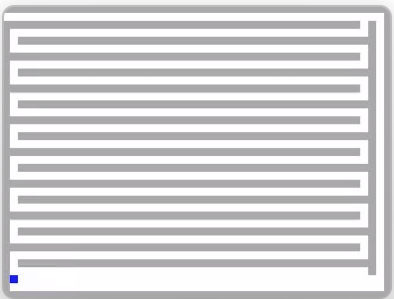


|  |  |  |
| --- | --- | --- |
| Hình . DFS (cost = NO) | Hình . BFS (cost = NO) | Hình . UCS (cost = NO) |
| Hình . Greedy BFS Heuristic 1 (cost = NO) | Hình . Greedy BFS Heuristic 2 (cost = NO) | Hình . Greedy BFS Heuristic 3 (cost = NO) |
| Hình . A\* Heuristic 1 (cost = NO) | Hình . A\* Heuristic 2 (cost = NO) | Hình . A\* Heuristic 3 (cost = NO) |

***Nhận xét:***

Bản đồ trên là trường hợp không tồn tại đường đi giữa nguồn và đích nên kết quả thu được của mọi giải thuật là như nhau:

* Tính đầy đủ: Tất cả các giải thuật đều không tìm được đường đi.
* Tính tối ưu: Các giải thuật rơi vào trường hợp xấu nhất nên không có giải thuật nào tối ưu.
* Độ mở các nút: Trừ những nút không thể duyệt đến thì toàn bộ nút còn lại đều được duyệt.
* Chi phí đường đi: cost = NO không tồn tại đường đi.
* **Bản đồ 4:**



|  |  |  |
| --- | --- | --- |
| Hình . DFS (cost = 126) | Hình . BFS (cost = 126) | Hình . UCS (cost = 126) |
| Hình . Greedy BFS Heuristic 1 (cost 738) | Hình . Greedy BFS Heuristic 2 (cost = 738) | Hình . Greedy BFS Heuristic 3 (cost = 738) |
| Hình . A\* Heuristic 1 (cost = 126) | Hình . A\* Heuristic 2 (cost = 126) | Hình . A\* Heuristic 3 (cost = 126) |

***Nhận xét:***

* Tính đầy đủ: Tất cả các giải thuật đều tìm được đường đi.
* Tính tối ưu: Bản đồ 4 xét về tính tối ưu chia rõ giữa:
  + Tối ưu: DFS, BFS, UCS, A\* Heuristic
  + Không tối ưu: Greedy Best First Search
    - Gready BFS chỉ quan tâm đến h(x) – khoảng cách heuristic từ vị trí hiện tại đến đích nên trong trường hợp bản đồ này, thuật toán đã bị “lừa”. Thuật toán chỉ cố gắng đi đến các điểm có vẻ gần với địch mà không nhận ra đó chính là đường dài, “vòng vèo” hơn rất nhiều so với đường tắt ngắn hơn nhưng chi phí để đi đến lúc đầu có vẻ nhiều hơn.
* Độ mở các nút: Trường hợp tốt nhất là DFS với số lượng nút được duyệt đến là ít nhất. Nhưng điều này không “ổn định”, phụ thuộc vào thứ tự nút ta chọn để duyệt, và may mắn trong trường hợp này thứ tự nút duyệt phù hợp để DFS có thể thuận lợi tìm ra đường tốt nhất. Kế đến là các giải thuật A\*, BFS, UCS tương đối tốt. Ngược lại, Greedy Best First Search khá xấu gần như mở tất cả nút của bản đồ.
* Chi phí đường đi:
  + Tối ưu (cost = 126): DFS, BFS, UCS, A\* Heuristic
  + Không tối ưu: Greedy Best First Search
* **Bản đồ 5:**



|  |  |  |
| --- | --- | --- |
| Hình . DFS (cost = 247) | Hình . BFS (cost = 83) | Hình . UCS (cost = 83) |
| Hình . Greedy BFS Heuristic 1 (cost = 83) | Hình . Greedy BFS Heuristic 2 (cost = 105) | Hình . Greedy BFS Heuristic 3 (cost = 83) |
| Hình . A\* Heuristic 1 (cost = 83) | Hình . A\* Heuristic 2 (cost = 83) | Hình . A\* Heuristic 3 (cost = 83) |

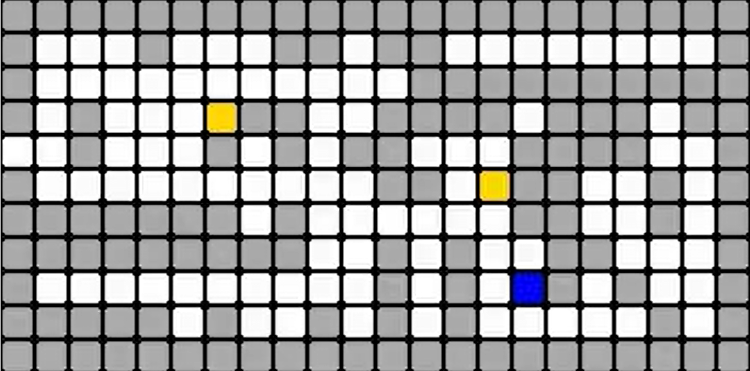
***Nhận xét:***

* Tính đầy đủ: Tất cả các giải thuật đều tìm được đường đi.
* Tính tối ưu: Các giải thuật dường như đều tìm được đường đi tối ưu nhất, trừ DFS và Greedy BFS 2. Trong số đó, DFS kém tối ưu nhất, nguyên nhân thì như đã đề cập ở, do việc chọn thứ tự đỉnh quyết định rất nhiều đến kết quả của DFS. Do trong việc cài đặt DFS nhóm chúng em chọn ưu tiên đi theo hướng trái phải rồi mới di chuyển lên xuống và thuật toán sẽ kết thúc ngay khi đến được đích nên mới sinh ra kết quả không tối ưu của DFS trong trường hợp này. Còn về Greedy BFS 2 (sử dụng độ đo manhattan) cũng không tìm được đường đi tối ưu.
* Độ mở các nút: Đổi lại từ chi phí “đắt đỏ” về đường đi của DFS thì thuật toán này đã có kết quả độ mở hết sức hiệu quả với số lượng các nút được mở khá ít ỏi. Ngược lại, BFS và UCS thì rơi vào trường hợp dường như duyệt đến tất cả các nút có trong bản đồ nên quá tốn kém về mặt thời gian cũng như bộ nhớ. Tiếp đến, với số lượng nút tương đối được duyệt là giải thuật A\* chiếm một nửa số nút. Còn lại các giải thuật Greedy BFS cũng khá hiệu quả.
* Chi phí đường đi: Gần như toàn bộ giải thuật đều tìm được đường đi có chi phí thấp nhất với cost = 83. Ngược lại, DFS có chi phí cao nhất cost = 247. Ngoài ra, Greedy BFS 2 cũng có chi phí đường đi không được tối ưu với cost = 105

### **Bản đồ có điểm thưởng**

|  |
| --- |
| *Ghi chú:*   * *Ô màu xanh: Ô xuất phát* * *Ô màu vàng: Ô điểm thưởng* * *Cost: Chi phí đường đi từ nguồn đến đích, cost = NO nếu không tồn tại đường đi* |

* **Trường hợp bản đồ có 2 điểm thưởng:**
  + **Bản đồ 1:**

****

|  |  |  |
| --- | --- | --- |
| Hình . A\* Heuristic 1 (cost = 16) | Hình . A\* Heuristic 2 (cost = 16) | Hình . A\* Heuristic 3 (cost = 16) |

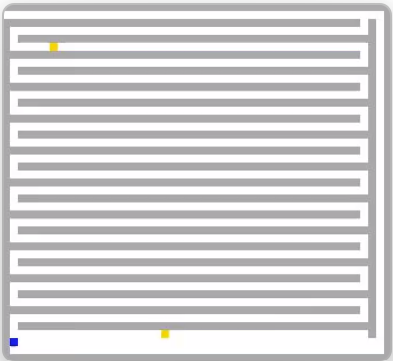
***Nhận xét:***

Với sự xuất hiện của điểm thưởng, trong trường hợp này bản đồ có 2 điểm thưởng nhưng ta có thể quan sát thấy chỉ 1 điểm thưởng được duyệt đến và đi qua, trong khi điểm thưởng thứ 2 chưa được xét đến. Bản đồ 1 này vẫn chưa thấy rõ được sự khác biệt.

* Tính đầy đủ: Tất cả các giải thuật đều tìm được đường đi.
* Tính tối ưu: Tất cả giải thuật đều tìm được đường đi tối ưu nhất theo tính chất của A\* đã được trình bày.
* Độ mở các nút: Một trong những tính chất khác biệt của 3 giải thuật A\* là sự khác biệt về các nút được duyệt.

//////////////////////////////////////////////////// giải thích

* Chi phí đường đi: Tất cả các giải thuật đều tìm được đường đi có chi phí đường đi thấp nhất cost = 16.
  + **Bản đồ 2:**



|  |  |  |
| --- | --- | --- |
| Hình . A\* Heuristic 1 (cost = 131) | Hình . A\* Heuristic 2 (cost = 131) | Hình . A\* Heuristic 3 (cost = 131) |

***Nhận xét:***

* Tính đầy đủ: Tất cả các giải thuật đều tìm được đường đi và cả 3 đều chưa duyệt đến ô chứa điểm thưởng.
* Tính tối ưu: Tất cả giải thuật đều tìm được đường đi tối ưu nhất, tuy nhiên, A\* Heuristic 2 lại có đường đi khác với 1 và 3

//////////////////////////////////////////////////// giải thích

* Độ mở các nút: Xét trên bản đồ 4, độ mở các nút của 3 loại A\* đều giống nhau.
* Chi phí đường đi: Tất cả các giải thuật đều tìm được đường đi có chi phí đường đi thấp nhất cost = 131.
* **Trường hợp bản đồ có 5 điểm thưởng:**
  + **Bản đồ 3:**

Ảnh có chứa bàn

Mô tả được tạo tự động

|  |  |  |
| --- | --- | --- |
| Hình . A\* Heuristic 1 (cost = -57) | Hình . A\* Heuristic 2 (cost = -57) | Hình . A\* Heuristic 3 (cost = -57) |

* **Trường hợp bản đồ có 10 điểm thưởng:**
  + **Bản đồ 5:**

****

|  |  |  |
| --- | --- | --- |
| Hình . A\* Heuristic 1 (cost = 71) | Hình . A\* Heuristic 2 (cost = 71) | Hình . A\* Heuristic 3 (cost = 71) |

***Nhận xét:***

Đường đi tìm được tối ưu với chi phí đường đi cost = 71 và 4/10 điểm thưởng được duyệt cho cả 3 trường hợp. Tuy nhiên, với sự khác biệt về tính chất của từng hàm Heuristic cho nên cả 3 đều có đường đi khác nhau dù cùng một chi phí. Độ mở các nút cũng khá tương đối nhiều, nhưng một số điểm thưởng vẫn chưa được duyệt đến, nếu trong trường hợp các điểm thưởng chưa duyệt giảm được chi phí khá lớn thì rõ ràng rằng đường đi tìm được chưa hẳn là tối ưu nhất. Qua ví dụ này, ta cũng kết luận được rằng giải thuật A\*chưa chắc sẽ tìm được đường đi ngắn nhất và tùy thuộc khá lớn vào hàm Heuristic.

### Bản đồ có điểm đón

////////////////////////////////

Phúc here

//////////////////////////////////

# Cách tổ chức chương trình và ghi chú lập trình

# *Thư mục bài nộp bao gồm:*

– File run.sh: file shell script chứa các câu lệnh để chạy toàn bộ các thuật toán cho tất cả các bản đồ.

– File report.pdf: báo cáo trình bày các vấn đề được yêu cầu trong đề bài.

– Folder source bao gồm:

+ Thư mục visual:

+ cell.py: chứa các thông tin hỗ trợ việc vẽ 1 cell.

+ map.py: có các hàm chứa thông tin của 1 map, các hàm hỗ trợ liên quan đến map cũng như việc thực hiện vẽ cũng như show map.

+ util.py: chứa các hàm hỗ trợ thường dùng.

+ path.py: chứa template của 1 class “thuật toán”, quy định các hàm chung nhất của các thuật toán tìm đường đi.

+ algo.py: chứa cài đặt của 5 thuật toán cơ bản.

+ award\_algo.py: chứa cài đặt của thuật toán cho bản đồ điểm thưởng và bản đồ điểm đón.

+ tele\_algo.py: chứa cài đặt của thuật toán cho bản đồ có điểm dịch chuyển.

+ heuristic.py: chứa cài đặt 3 hàm heuristic cơ bản.

+ main.py: chứa cài đặt tổng, cũng như là xử lý các ArgumentParser.

+ requirements.txt: chứa các thư viện cần cài thông qua pip.

– Folder input bao gồm các folder, file theo format và yêu cầu của giáo viên:

+ level\_1: folder chứa 5 map (.txt) của level\_1.

+ level\_2: folder chứa 3 map (.txt) của level\_2.

+ level\_3: folder chứa 5 map (.txt) của level\_3.

+ advance: folder chứa 4 map (.txt) của level advance.

# *Mô tả file run.sh*

- Thực hiện cài đặt các thư viện cần thiết thông qua pip từ requirements.txt.

- Chạy file main.py.

- Có các ví dụ sử dụng các argument với main.py.

# Tài liệu tham khảo

Các slide về giải thuật và tài liệu hướng dẫn được cung cấp trên Moodle.

https://vi.wikipedia.org/wiki/Bài\_toán\_đường\_đi\_ngắn\_nhất

<https://algs4.cs.princeton.edu/44sp/>

<https://viblo.asia/>

https://kcntt.duytan.edu.vn/