

**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN KHOA CÔNG NGHỆ THÔNG TIN**



🙝🙝🙝🕮🙟🙟🙟

# BÀI TẬP CÁ NHÂN

**MÔN CƠ SỞ TRÍ TUỆ NHÂN TẠO**

**ĐỀ TÀI:**

# NGHIÊN CỨU VỀ THUẬT TOÁN TÌM KIẾM ĐỒ THỊ

**Giảng viên phụ trách : Bùi Duy Đăng**

**Lê Nguyễn Nhựt Trường**

|  |  |
| --- | --- |
| **Stt** | **Thành viên** |
| **1** | **21120060-Nguyễn Long Giang** |
| **2** | **21120063-Hà Thanh Hải** |
| **3** | **21120064-Lê Quốc Hân** |
| **4** | **21120082-Phan Quốc Huy (nhóm trưởng)** |

**Thành phố Hồ Chí Minh – tháng 04 năm 2024**

MỤC LỤC

[I. GIỚI THIỆU 3](#_Toc163852240)

[II. PHÂN LOẠI 3](#_Toc163852241)

[1. UNINFORMED SEARCH 3](#_Toc163852242)

[2. INFORMED SEARCH 4](#_Toc163852243)

[III. NGHIÊN CỨU THUẬT TOÁN 5](#_Toc163852244)

[1. THUẬT TOÁN DEPTH-FIRST SEARCH (DFS) 5](#_Toc163852245)

[A. GIỚI THIỆU 5](#_Toc163852246)

[B. Ý TƯỞNG THỰC HIỆN 6](#_Toc163852247)

[C. MÃ GIẢ 6](#_Toc163852248)

[D. MINH HỌA 7](#_Toc163852249)

[2. THUẬT TOÁN BREADTH-FIRST SEARCH (BFS) 7](#_Toc163852250)

[A. GIỚI THIỆU 7](#_Toc163852251)

[B. Ý TƯỞNG THỰC HIỆN 7](#_Toc163852252)

[C. MÃ GIẢ 8](#_Toc163852253)

[D. MINH HỌA 9](#_Toc163852254)

[3. THUẬT TOÁN UNIFORM COST SEARCH (UCS) 9](#_Toc163852255)

[A. GIỚI THIỆU 9](#_Toc163852256)

[B. Ý TƯỞNG THỰC HIỆN 9](#_Toc163852257)

[C. MÃ GIẢ 10](#_Toc163852258)

[D. MINH HỌA 11](#_Toc163852259)

[4. THUẬT TOÁN ASTAR (A\*) 11](#_Toc163852260)

[A. GIỚI THIỆU 11](#_Toc163852261)

[B. Ý TƯỞNG THỰC HIỆN 11](#_Toc163852262)

[C. MÃ GIẢ 12](#_Toc163852263)

[D. MINH HỌA 13](#_Toc163852264)

[E. HÀM HEURISTIC 13](#_Toc163852265)

[IV. MỨC ĐỘ THỰC HIỆN 16](#_Toc163852266)

[1. MỨC 1 16](#_Toc163852267)

[2. MỨC 2 18](#_Toc163852268)

[3. MỨC 3 23](#_Toc163852269)

[V. TÀI LIỆU THAM KHẢO 27](#_Toc163852270)

# GIỚI THIỆU

* Tưởng tượng bạn đang lạc vào một mê cung khổng lồ, nơi ẩn chứa vô số kho báu và bí mật. Để chinh phục mê cung này, bạn cần một bản đồ và một chiến lược thông minh. Giống như mê cung, thế giới đồ thị ẩn chứa vô số thông tin và mối liên hệ giữa các đối tượng. Khám phá thế giới này là chìa khóa để giải quyết nhiều vấn đề phức tạp trong khoa học máy tính và toán học.
* Bài viết này sẽ dẫn dắt bạn bước vào hành trình khám phá thế giới đồ thị đầy thú vị. Chúng ta sẽ cùng tìm hiểu 4 thuật toán quan trọng: DFS, BFS, UCS và AStar, đóng vai trò như những bản đồ giúp bạn định hướng và tìm kiếm kho báu trong mê cung. Bên cạnh đó, bạn cũng sẽ được giới thiệu thêm về các thuật toán khác như Greedy Search và Dijsktra, mở rộng thêm kho tàng bí kíp chinh phục thế giới đồ thị.

# PHÂN LOẠI

* Có nhiều cách phân loại các thuật toán tìm kiếm trên đồ thị, trong đó phổ biến hơn cả ta chia ra làm hai loại: Uninformed Search (Blind Search) và Informed Search (Heuristic Search).

## UNINFORMED SEARCH

* Các thuật toán thuộc phân loại này có đặc điểm nổi bật là không chứa bất cứ dữ liệu hay thông tin bổ sung nào về cấu trúc của đồ thị hay vị trí của đối tượng đang xét đối với mục tiêu tìm kiếm. Thay vào đó, nó chỉ tiến hành tìm kiếm dựa trên thông tin cơ bản được cung cấp từ yêu cầu đặt ra.
* Loại thuật toán này hoạt động bằng cách duyệt qua tất cả các nút trong đồ thị một cách tổng quát mà không xét đến độ ưu tiên cho các nút đó. Do đó, chúng cần tốn nhiều tài nguyên, thời gian và số lần thử hơn để tìm được tới mục tiêu.
* Tuy vậy, Uninformed Search luôn đảm bảo duyệt qua tất cả các vị trí nút trong đồ thị và luôn hoàn thành nhiệm vụ tìm kiếm nếu giải giải pháp tìm kiếm cho bài toán đó có tồn tại.
* Một số thuật toán tiêu biểu bao gồm: DFS, BFS, UCS, DLS, …

## INFORMED SEARCH

* Trái ngược với Uninformed Search, thuật toán Informed Search sử dụng thêm thông tin đặc biệt để hỗ trợ quá trình tìm kiếm mục tiêu. Chẳng hạn như còn cách mục tiêu bao xa, chi phí đường dẫn, độ ưu tiên, cách tiếp cận đến mục tiêu, … Nhờ vậy, quá trình tìm kiếm và tài nguyên sử dụng sẽ được tối ưu hơn thuật toán Uninformed Search.
* Thuật toán Informed Search sử dụng một hàm số gọi là Heuristic, hàm này được sử dụng để ước tính khoảng cách hoặc chi phí tìm kiếm cho các khả năng ở hiện tại và chọn ra kết quả tối ưu nhất. Đây là một hàm rất quan trọng trong thuật toán Informed Search vì nó quyết định đến việc thuật toán có tìm được mục tiêu hay không. Cho nên, việc xác định một hàm Heuristic đủ tốt và thông mình rất quan trọng, nếu có sai sót trong hàm Heuristic, nó có thể dẫn đến việc thuật toán bỏ sót các khả năng và giải pháp, từ đó dẫn đến không hoàn thành nhiệm vụ tìm kiếm.
* Một số thuật toán tiêu biểu bao gồm: AStar, Best-First Search, Greedy Search…

1. SO SÁNH UNINFORMED SEARCH VÀ INFORMED SEARCH

|  |  |  |
| --- | --- | --- |
| **Tiêu chí** | **Informed Search** | **Uninformed Search** |
| Chiến lược | Tìm kiếm có thông tin, sử dụng hàm Heuristics | Tìm kiếm theo một hệ thống |
| Hiệu suất | Tìm ra giải pháp nhanh hơn khi hàm Heuristics hiệu quả | Tìm ra giải pháp chậm hơn |
| Độ hoàn thành | Có thể không hoàn thành hoặc đi xa khỏi lời giải nếu hàm Heuristics không phù hợp.  Có thể lặp vô tận | Luôn hoàn thành |
| Chi phí | Chi phí thấp | Chi phí cao |
| Kích thước bài toán tìm kiếm | Có thể xử lý các bài toán tìm kiếm lớn | Tìm ra giải pháp cho một bài toán tìm kiếm lớn là khó khi phải thời gian tìm kiếm lâu. |

# NGHIÊN CỨU THUẬT TOÁN

## THUẬT TOÁN DEPTH-FIRST SEARCH (DFS)

### GIỚI THIỆU

* Thuật toán Depth-First Search - Tìm kiếm theo chiều sâu, là một thuật toán tìm kiếm trên đồ thị dựa trên việc duyệt qua hết 1 nhánh các đỉnh theo chiều sâu trước khi quay lại kiểm tra các nhánh khác.

### Ý TƯỞNG THỰC HIỆN

* Thuật toán bắt đầu từ đỉnh gốc, tiếp tục đi đến các đỉnh kề theo một nhánh bằng các cạnh của đồ thị. Quá trình này lặp lại cho tới khi tìm thấy mục tiêu hoặc tất cả các đỉnh đều được duyệt qua.
* Nếu tại một đỉnh nào đó không còn đỉnh nào kề chưa duyệt qua nữa thì quay trở lại tiếp tục tìm con đường khác của đồ thị.
* Thuật toán sử dụng cấu trúc dữ liệu ngăn xếp (stack) lưu trữ các đỉnh đã duyệt qua để đảm bảo các đỉnh chỉ được duyệt một lần duy nhất.

### MÃ GIẢ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Depth-First-Search (DFS)** | | | | |
|  | ***Input:*** *trạng thái ban đầu (****S0****)* | | | |
|  | ***Output:*** *trạng thái kết thúc (****Sg****)* | | | |
| ***1:*** | ***add S0 to OPEN*** | | | |
| ***2:*** | ***loop*** | | | |
| ***3:*** |  | ***get the first element from OPEN and assign to current\_state*** | | |
| ***4:*** |  | ***remove current\_state from OPEN*** | | |
| ***5:*** |  | ***if current\_state is Sg*** | | |
| ***6:*** |  |  | ***return path*** | |
| ***7:*** |  | ***add current\_state to CLOSED*** | | |
| ***8:*** |  | ***foreach neighbour in get\_neighbours(current\_state)*** | | |
| ***9:*** |  |  | ***if neighbour not in OPEN and neighbour not in CLOSED*** | |
| ***10:*** |  |  |  | ***add neighbour to OPEN as the first element*** |
| ***11*** |  |  |  | ***path[neighbour] = current\_state*** |
| ***12:*** | ***end loop*** | | | |
| ***13:*** | ***return “No path found”*** | | | |

### MINH HỌA

A diagram of a tree with an empty stack

Description automatically generated

## THUẬT TOÁN BREADTH-FIRST SEARCH (BFS)

### GIỚI THIỆU

* BFS xác định không gian tìm kiếm bằng cách duyệt theo chiều rộng, mà duyệt tất cả các trạng thái trong cùng một độ sâu trước khi chuyển sang trạng thái ở mức khác.

### Ý TƯỞNG THỰC HIỆN

* Thuật toán này viếng thăm các đỉnh theo thứ tự tăng dần mức (level) của đỉnh.
* Từ đỉnh đầu, thuật toán đi qua các đỉnh mức 1 sau đó đến các đỉnh mức 2…Quá trình này được tiếp tục cho đến khi tất cả các đỉnh đã được viếng thăm.
* Thuật toán sử dụng cấu trúc dữ liệu hàng đợi (queue) lưu trữ các đỉnh cùng mức
* trước khi đi tới các mức tiếp theo. Hơn nữa, hàng đợi lưu trữ các đỉnh đã viếng thăm, đảm bảo các đỉnh chỉ được duyệt một lần duy nhất.

### MÃ GIẢ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Breadth-First-Search (DFS)** | | | | |
|  | ***Input:*** *trạng thái ban đầu (****S0****)* | | | |
|  | ***Output:*** *trạng thái kết thúc (****Sg****)* | | | |
| ***1:*** | ***add S0 to OPEN*** | | | |
| ***2:*** | ***loop*** | | | |
| ***3:*** |  | ***get the first element from OPEN and assign to current\_state*** | | |
| ***4:*** |  | ***remove current\_state from OPEN*** | | |
| ***5:*** |  | ***if current\_state is Sg*** | | |
| ***6:*** |  |  | ***return path*** | |
| ***7:*** |  | ***add current\_state to CLOSED*** | | |
| ***8:*** |  | ***foreach neighbour in get\_neighbours(current\_state)*** | | |
| ***9:*** |  |  | ***if neighbour not in OPEN and neighbour not in CLOSED*** | |
| ***10:*** |  |  |  | ***add neighbour to OPEN as the last element*** |
| ***11:*** |  |  |  | ***path[neighbour] = current\_state*** |
| ***12:*** | ***end loop*** | | | |
| ***13:*** | ***return “No path found”*** | | | |

### MINH HỌA

A diagram of numbers and circles

Description automatically generated

## THUẬT TOÁN UNIFORM COST SEARCH (UCS)

### GIỚI THIỆU

* Thuật toán Uniform Cost Search - Tìm kiếm chi phí đều, là một thuật toán duyệt hay tìm kiếm trên một cây hoặc đồ thị có trọng số. Việc tìm kiếm bắt đầu tại đỉnh gốc và tiếp tục bằng cách duyệt các đỉnh tiếp theo sao cho tính từ đỉnh gốc, tổng trọng số hay chi phí của đường đi là thấp nhất.

### Ý TƯỞNG THỰC HIỆN

* Thuật toán này viếng thăm các đỉnh theo thứ tự ưu tiên trọng số (chi phí) tích lũy nhỏ nhất từ đỉnh gốc đến mỗi đỉnh, đảm bảo tìm ra đường đi có tổng chi phí thấp nhất mà không quan tâm đến số bước phải đi.
* Thuật toán sử dụng cấu trúc dữ liệu hàng đợi ưu tiên (priority queue) lưu trữ các đỉnh với độ ưu tiên là trọng số của cạnh nối với đỉnh kế tiếp trên đường đi.

### MÃ GIẢ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Uniform-Cost-Search (UCS)** | | | | | |
|  | ***Input:*** *trạng thái ban đầu (****S0****)* | | | | |
|  | ***Output:*** *trạng thái kết thúc (****Sg****)* | | | | |
| ***1:*** | ***add (0, S0)to OPEN*** | | | | |
| ***2:*** | ***loop*** | | | | |
|  |  | ***sort OPEN in ascending order based on cost*** | | | |
| ***3:*** |  | ***get the first element from OPEN and assign to cost, current\_state*** | | | |
| ***4:*** |  | ***remove current\_state from OPEN*** | | | |
| ***5:*** |  | ***if current\_state is Sg*** | | | |
| ***6:*** |  |  | ***return path*** | | |
| ***7:*** |  | ***add current\_state to CLOSED*** | | | |
| ***8:*** |  | ***foreach neighbour in get\_neighbours(current\_state)*** | | | |
| ***9:*** |  |  | ***if neighbour not in CLOSED*** | | |
| ***10*** |  |  |  | ***tentative\_cost = cost + distance\_between(neighbor, current\_state)*** | |
| ***11*** |  |  |  | ***if neighbour not in OPEN*** | |
| ***12:*** |  |  |  |  | ***Add (tentative\_cost, neighbour) to OPEN as the last element*** |
| ***13:*** |  |  |  |  | ***path[neighbour] = current\_state*** |
| ***14:*** |  |  |  | ***if tentative\_cost < [cost of neighbour in OPEN]*** | |
| ***15:*** |  |  |  |  | ***change the cost of neighbour in OPEN to tentative\_cost*** |
| ***16:*** |  |  |  |  | ***path[neighbour] = current\_state*** |
| ***17:*** | ***end loop*** | | | | |
| ***18:*** | ***return “No path found”*** | | | | |

### MINH HỌA

A diagram of a diagram of a diagram

Description automatically generated

## THUẬT TOÁN ASTAR (A\*)

### GIỚI THIỆU

* A\* là thuật toán tìm đường ngắn nhất kết hợp giữa việc tính toán chi phí thực sự đã di chuyển và ước lượng chi phí còn lại. Bằng cách này, nó tối ưu hóa việc tìm kiếm đường đi, làm cho nó trở thành công cụ quan trọng trong lập trình trí tuệ nhân tạo và trò chơi.

### Ý TƯỞNG THỰC HIỆN

* A\* sử dụng một hàm heuristic để đánh giá đường đi tốt nhất có thể đi. Điều này làm cho A\* trở nên “thông minh” hơn nhiều so với các thuật toán Uninformed Search khác. Vì vậy mà thuật toán này được sử dụng rộng rãi trong lĩnh vực trí tuệ nhân tạo, giải thuật tìm kiếm và ứng dụng tìm đường đi trong bản đồ.

### MÃ GIẢ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **AStar Search (A\* Search)** | | | | | |
|  | ***Input:*** *trạng thái ban đầu (****S0****)* | | | | |
|  | ***Output:*** *trạng thái kết thúc (****Sg****)* | | | | |
| ***1:*** | ***add (0, S0)to OPEN*** | | | | |
| ***2:*** | ***g\_cost = list n elements with [100,000] value*** | | | | |
| ***3:*** | ***loop*** | | | | |
| ***4:*** |  | ***sort OPEN in ascending order based on cost*** | | | |
| ***5:*** |  | ***get the first element from OPEN and assign to node\_cost, current\_state*** | | | |
| ***6:*** |  | ***remove current\_state from OPEN*** | | | |
| ***7:*** |  | ***if current\_state is Sg*** | | | |
| ***8:*** |  |  | ***return path*** | | |
| ***9:*** |  | ***add current\_state to CLOSED*** | | | |
| ***10:*** |  | ***foreach neighbour in get\_neighbours(current\_state)*** | | | |
| ***11:*** |  |  | ***if neighbour not in CLOSED*** | | |
| ***12:*** |  |  |  | ***tentative\_cost = g\_cost[current\_state] + distance\_between(neighbor, current\_state)***  ***+ heuristic(neighbour, Sg)*** | |
| ***13:*** |  |  |  | ***if neighbour not in OPEN*** | |
| ***14:*** |  |  |  |  | ***Add (tentative\_cost, neighbour) to OPEN as the last element*** |
| ***15:*** |  |  |  |  | ***path[neighbour] = current\_state*** |
| ***16*** |  |  |  |  | ***g\_cost[neighbour] = g\_cost[current\_state] + distance\_between(neighbor, current\_state)*** |
| ***17:*** |  |  |  | ***if tentative\_cost < [cost of neighbour in OPEN]*** | |
| ***18:*** |  |  |  |  | ***change the cost of neighbour in OPEN to tentative\_cost*** |
| ***19:*** |  |  |  |  | ***path[neighbour] = current\_state*** |
| ***20:*** |  |  |  |  | ***g\_cost[neighbour] = g\_cost[current\_state] + distance\_between(neighbor, current\_state)*** |
|  |  | ***end for*** | | | |
| ***21:*** | ***end loop*** | | | | |
| ***22:*** | ***return “No path found”*** | | | | |

### MINH HỌA

A black text on a white background

Description automatically generated

### HÀM HEURISTIC

1. **Manhattan**

* Hàm heuristic Manhattan thường được sử dụng trong các bài toán như tìm đường đi ngắn nhất từ một điểm tới một điểm khác trên một bản đồ lưới, ví dụ như trong trò chơi điện tử, tìm đường đi trong bản đồ thành phố, hoặc trong lập kế hoạch di chuyển của robot.
* Hàm này tốt nhất được sử dụng trong các tình huống khi không thể di chuyển theo đường chéo (đường ngắn nhất giữa hai điểm không phải là đường thẳng), và giả sử mỗi bước di chuyển từ một ô lưới sang một ô khác đều có chi phí là một đơn vị.

A screen shot of a computer program

Description automatically generated

1. **Chebyshev**

* Hàm heuristic Chebyshev có thể được sử dụng trong các bài toán tương tự như hàm heuristic Manhattan, bao gồm tìm đường đi ngắn nhất từ một điểm tới một điểm khác trên một bản đồ lưới, trong các trò chơi điện tử hoặc trong lập kế hoạch di chuyển của robot. Tùy thuộc vào bản chất của bài toán cụ thể, lựa chọn giữa hàm heuristic Manhattan và hàm heuristic Chebyshev có thể phụ thuộc vào cách mà các điểm trên lưới được kết nối và sự ưu tiên về hướng di chuyển.

A computer screen shot of a program

Description automatically generated

1. **Diagonal**

* Được sử dụng khi di chuyển 8 hướng, hướng chéo có chi phí lớn hơn hướng thẳng, thông thường sẽ là gấp căn bậc 2.
* Hàm heuristic Diagonal thường được sử dụng trong các bài toán tìm kiếm đường đi ngắn nhất khi không gian di chuyển không bị hạn chế theo các hướng đặc biệt, và khi mục tiêu là tối ưu hóa đường đi theo cả hướng ngắn nhất và đường chéo nhất. Cụ thể, nó có thể được sử dụng trong các trò chơi điện tử, bài toán lập kế hoạch di chuyển của robot hoặc trong các hệ thống điều khiển tự động khác.

A computer screen shot of a program

Description automatically generated

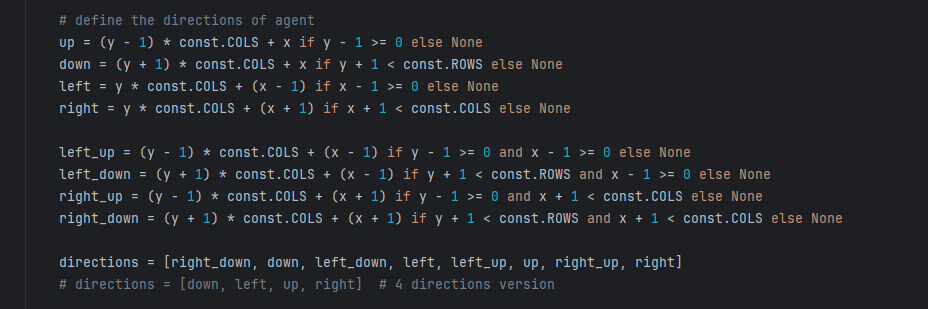
# MỨC ĐỘ THỰC HIỆN

## MỨC 1

* **Mức độ hoàn thành: 100%**
* **Nhận xét:**

1. **DFS**

* Cho kết quả khá nhanh khi điểm đích gần với đường đi. Di chuyển phụ thuộc vào thứ tự ưu tiên của hướng đi mà lập trình viên quy định trong quá trình phát triển thuật toán.



|  |  |
| --- | --- |
| **Chạy thử lần 1** | Không có mô tả. |
| **Chạy thử lần 2** |  |
| **Chạy thử lần 3** |  |

## MỨC 2

* **Mức độ hoàn thành: 100%**
* **Nhận xét:**

1. **BFS**

* Mở rộng xung quanh điểm bắt đầu sau đó lan rộng ra cho tới điểm kết thúc.
* Phải duyệt tất cả các node khi điểm kết thúc ở xa.

|  |  |
| --- | --- |
| **Chạy thử lần 1** |  |
| **Chạy thử lần 2** |  |
| **Chạy thử lần 3** |  |

1. **UCS**

* Cách thức mở rộng giống như BFS, tuy nhiên hay hướng tới điểm kết thúc nhiều hơn, đến khi gặp điểm đích thì dừng lại.

|  |  |
| --- | --- |
| **Chạy thử lần 1** |  |
| **Chạy thử lần 2** |  |
| **Chạy thử lần 3** |  |

1. **ASTAR**

* Duyệt các điểm gần với điểm đích, ít duyệt cái ô không liên quan.

|  |  |
| --- | --- |
| **Chạy thử lần 1** |  |
| **Chạy thử lần 2** |  |
| **Chạy thử lần 3** |  |

* **So sánh thuật toán:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TIÊU CHÍ** | **UCS** | **DFS** | **ASTAR** | **BFS** |
| Loại thuật toán | Informed Search | Uninformed Search | Informed Search | Uninformed Search |
| Chiến lược | Lựa chọn nút tiếp theo có chi phí thấp nhất tính từ nút gốc | Tìm kiếm theo chiều sâu, ưu tiên đi sâu vào các nhánh con trước khi quay lại để khám phá các nhánh khác | Kết hợp chi phí đã đi (g) và chi phí ước tính còn lại (h) để lựa chọn nút tiếp theo có giá trị f = g + h thấp nhất | Tìm kiếm theo chiều rộng, khám phá tất cả các nút ở cùng một độ sâu trước khi chuyển sang độ sâu tiếp theo |
| Hiệu suất | Chậm hơn A\*, nhưng đảm bảo tìm thấy đường đi tối ưu trong đồ thị có trọng số không âm | Nhanh hơn UCS, nhưng có thể bị kẹt trong các nhánh con dài | Nhanh hơn UCS và có khả năng tìm thấy đường đi gần tối ưu trong nhiều trường hợp | Chậm hơn DFS, nhưng đảm bảo tìm thấy đường đi ngắn nhất trong đồ thị không có chu kỳ âm |
| Độ hoàn thành | Đảm bảo hoàn thành nếu đồ thị hữu hạn và có trọng số không âm | Đảm bảo hoàn thành nếu đồ thị hữu hạn | Đảm bảo hoàn thành nếu đồ thị hữu hạn, có trọng số không âm và hàm heuristic admissibile | Đảm bảo hoàn thành nếu đồ thị hữu hạn và không có chu kỳ âm |
| Tối ưu | Đảm bảo đường đi tối ưu trong đồ thị có trọng số không âm | Không đảm bảo đường đi tối ưu | Đảm bảo đường đi tối ưu nếu hàm heuristic admissibile và consistent | Đảm bảo đường đi tối ưu trong đồ thị không có chu kỳ âm |
| Phụ thuộc Heuristic | Không phụ thuộc vào heuristic | Không phụ thuộc vào heuristic | Phụ thuộc vào heuristic. Hiệu suất của A\* phụ thuộc vào chất lượng của hàm heuristic | Không phụ thuộc vào heuristic |
| Độ phức tạp thời gian | O(b^d) trong trường hợp xấu nhất | O(b^d) trong trường hợp xấu nhất | O(b^d) trong trường hợp xấu nhất | O(b^d) trong trường hợp xấu nhất |
| Độ phức tạp không gian | O (độ sâu của cây tìm kiếm) trong trường hợp xấu nhất | O (độ sâu của cây tìm kiếm) trong trường hợp xấu nhất | O (độ sâu của cây tìm kiếm) trong trường hợp xấu nhất | O (số lượng nút con trung bình của mỗi nút) trong trường hợp xấu nhất |

* **Chú thích:**
  + b: là số lượng nút con trung bình của mỗi nút.
  + d: là độ sâu của cây tìm kiếm.

## 3. MỨC 3

* **Mức độ hoàn thành: 100%**
* **Thuật toán áp dụng:**
* Từ node xuất phát ta dùng hàm Heuristic để tính toán điểm gần với S nhất, bằng cách sử dụng Heuristic Manhatan để sắp xếp lại các điểm đón.

A screen shot of a computer code

Description automatically generated

* Đoạn mã phụ trách vẽ lại đường đi từ điểm xuất phát đến điểm đích.

Với mục tiêu tìm tổng đường đi nhỏ nhất thì sử dụng thuật toán A\* để thực hiện tìm đường đi từ điểm xuất phát đến điểm đích.

A screenshot of a computer program

Description automatically generated

* A screenshot of a computer

  Description automatically generated**Minh họa:**

**Chạy thử lần 1**

A screenshot of a computer

Description automatically generated

**Chạy thử lần 2**

**A screenshot of a computer

Description automatically generated**

**Chạy thử lần 3**

* **Hướng dẫn chạy code:**

B1: Mở terminal cd đến folder chứa file main.py.

B2: Nhập vào termial “pip install pygame”

* + - Enter

B3: Nhập vào terminal “ python main.py --algo DFS”

* + - Enter

Để thay đổi thuật toán, chỉ cần thay đổi “DFS” sang “BFS hoặc UCS hoặc A\*”

Nếu muốn thay đổi map, chỉ cần vào file main.py

Và thay đổi map cần chạy

**A black background with white text

Description automatically generated**

* **Link demo:**

[**https://youtu.be/mMF11oVYN9M?si=VKcvQei1YI9XzBCq**](https://youtu.be/mMF11oVYN9M?si=VKcvQei1YI9XzBCq)

# TÀI LIỆU THAM KHẢO

1. [Heuristics (stanford.edu)](http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html)
2. [[Algorithm] Các thuật toán tìm kiếm trong AI – FLINTERS Developer's Blog](https://labs.flinters.vn/algorithm/algorithm-cac-thuat-toan-tim-kiem-trong-ai/)
3. [Amit’s A\* Pages (stanford.edu)](http://theory.stanford.edu/~amitp/GameProgramming/)
4. [CTDL&GT - Graph Algorithms - Breadth First Search - Viblo](https://viblo.asia/p/data-structure-algorithm-graph-algorithms-breadth-first-search-bfs-gwd43kMM4X9)
5. [Breadth First Search or BFS for a Graph - GeeksforGeeks](https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/)
6. [Difference between Informed and Uninformed Search in AI - Testbook.com](https://testbook.com/key-differences/difference-between-informed-and-uninformed-search-in-ai#:~:text=An%20informed%20search%20provides%20the,information%20about%20the%20potential%20solution.)
7. Geeksforgeeks (2/2023), Difference between Informed and Uninformed Search in AI, [https://www.geeksforgeeks.org/difference-between-informed-and-](https://www.geeksforgeeks.org/difference-between-informed-and-uninformed-search-in-ai/) [uninformed-search-in-ai/](https://www.geeksforgeeks.org/difference-between-informed-and-uninformed-search-in-ai/)
8. Milos Simic (5/2023), The Informed vs. Uninformed Search Algorithms, <https://www.baeldung.com/cs/informed-vs-uninformed-search>
9. Katsu - Howkteam.vn, Đồ thị và cây, [https://howkteam.vn/course/cau-truc-du-](https://howkteam.vn/course/cau-truc-du-lieu-va-giai-thuat/do-thi-va-cay-4319) [lieu-va-giai-thuat/do-thi-va-cay-4319](https://howkteam.vn/course/cau-truc-du-lieu-va-giai-thuat/do-thi-va-cay-4319)
10. Vũ Quế Lâm (2/2022), Các giải thuật tìm kiếm trên đồ thị, <https://viblo.asia/p/cac-giai-thuat-tim-kiem-tren-do-thi-1Je5EBRGKnL>
11. Bùi Quang Hà (11/2020), [Algorithm] Các thuật toán tìm kiếm trong AI, <https://labs.flinters.vn/algorithm/algorithm-cac-thuat-toan-tim-kiem-trong-ai/>
12. Ashwin Ramachandran (9/2023), Depth-first Search (DFS) Algorithm, <https://www.interviewkickstart.com/learn/depth-first-search-algorithm>

--HẾT--