**TRƯỜNG ĐẠI HỌC CẦN THƠ**

**TRƯỜNG CÔNG NGHỆ THÔNG TIN & TRUYỀN THÔNG**



**BÁO CÁO NHÓM II**

**NHẬP MÔN TRÍ TUỆ NHÂN TẠO**

**CT190**

***ĐỀ TÀI***: **MỘT SỐ PHƯƠNG PHÁP TÌM KIẾM**

**ĐƯỜNG ĐI TRONG MÊ CUNG VÀ ỨNG DỤNG.**

**Giảng viên giảng dạy : *Th.s Huỳnh Gia Khương***

**Nhóm sinh viên thực hiện:**

1. Huỳnh Thanh Phong KHMT K48 MSSV:B2207555
2. Vũ Đức Thắng KHMT K48 MSSV:B2207567
3. Trần Văn Khỏe CNTT K47 MSSV:B2111848
4. Nguyễn Trịnh Trọng Chiến MMT&TTDL K48 MSSV:B2204924

**MỤC LỤC**

[LỜI NÓI ĐẦU 1](#_Toc150383695)

[CHƯƠNG I. KHÁI QUÁT VỀ BÀI TOÁN VÀ CÁC KHÁI NIỆM CƠ BẢN.2](#_Toc150383696)

[1.1 Mô tả Ma Trận: 2](#_Toc150383697)

[1.2 Điểm Xuất Phát và Điểm Đích 2](#_Toc150383698)

[1.3 Ràng Buộc và Hạn Chế 2](#_Toc150383699)

[1.4 Thuật Toán Tìm Đường Đi 2](#_Toc150383700)

[1.5 Quá Trình Tìm Kiếm 3](#_Toc150383701)

[1.6 Kết Quả Đường Đi 3](#_Toc150383702)

[CHƯƠNG II. KHÔNG GIAN TRẠNG THÁI VÀ CÁC HÀM LIÊN QUAN 4](#_Toc150383703)

[2.1 Xây dựng ma trận. 4](#_Toc150383704)

[2.2 Không gian trạng thái. 4](#_Toc150383705)

[2.3 Hàm đánh giá Heuristic. 5](#_Toc150383706)

[CHƯƠNG III. PHƯƠNG PHÁP TÌM KIẾM MÙ (UNINFORMED BLIND SEARCH) 6](#_Toc150383707)

[3.1 TÌM KIẾM THEO DUYỆT CHIỀU RỘNG (BREADTH FIRST SEARCH - BFS) 6](#_Toc150383708)

[3.1.1 Khái niệm BFS: 6](#_Toc150383709)

[3.1.2 Giải thuật BFS: 6](#_Toc150383710)

[3.2 TÌM KIẾM THEO DUYỆT CHIỀU SÂU (DEPTH FIRST SEARCH-DFS) 13](#_Toc150383711)

[3.2.1 Khái niệm DFS: 13](#_Toc150383712)

[3.2.2 Giải thuật DFS: 13](#_Toc150383713)

[3.3 SO SÁNH BFS VÀ DFS 17](#_Toc150383714)

[CHƯƠNG IV. TÌM KIẾM DỰA TRÊN KINH NGHIỆM (INFORMED/HEURISTIC SEARCH) 19](#_Toc150383715)

[4.1 TÌM KIẾM BẰNG PHƯƠNG PHÁP CỦA GIẢI THUẬT HÁU ĂN (GREEDY BEST-FIRST SEARCH) 19](#_Toc150383716)

[4.2 TÌM KIẾM BẰNG PHƯƠNG PHÁP CỦA GIẢI THUẬT A\* 27](#_Toc150383719)

[4.2.1 Tổng quát về giải thuật A\* 27](#_Toc150383720)

[4.2.2 Giải thuật của thuật toán A\* 29](#_Toc150383721)

[4.1 TÌM KIẾM BẰNG PHƯƠNG PHÁP CỦA GIẢI THUẬT LEO ĐỒI (HILL CLIMBING) 46](#_Toc150383722)

[4.3.1 Vài nét về giải thuật leo đồi (Hill Climbing) 46](#_Toc150383723)

[4.3.2 Đặc điểm của giải thuật leo đồi (Hill Climbing) 46](#_Toc150383724)

[4.3.3 Các loại của giải thuật leo đồi 46](#_Toc150383725)

[4.3.4 Sơ đồ không gian trạng thái của giải thuật leo đồi (Hill Climbing) 49](#_Toc150383726)

[CHƯƠNG V. CẤU TRÚC DỮ LIỆU VÀI GIẢI THUẬT 50](#_Toc150383727)

[CHƯƠNG VI. ỨNG DỤNG GIẢI THUẬT VÀO VIDEO GAME 57](#_Toc150383728)

[I. ỨNG DỤNG GIẢI THUẬT VÀO GAME GIẢI ĐỐ 57](#_Toc150383729)

[II. GIẢI THUẬT ĐƯỢC ỨNG DỤNG NHƯ THẾ NÀO? 57](#_Toc150383730)

[III. NGUYÊN LÝ HOẠT ĐỘNG CỦA GẢM DỰA TRÊN GIẢI THUẬT 57](#_Toc150383731)

[IV. SẢN PHẨM GAME 59](#_Toc150383732)

[CHƯƠNG VII. NGUỒN TÀI LIỆU THAM KHẢO 62](#_Toc150383733)

# LỜI NÓI ĐẦU

Trong kỷ nguyên công nghệ 4.0 phát triển nở rộ, thế giới đã và đang có những bước phát triển mạnh và công nghệ là một trong những lĩnh vực thay đổi đáng chú ý nhất. Rất nhiều công nghệ mới xuất hiện đã giúp đời sống chúng ta thay đổi theo chiều hướng tích cực hơn, khiến khoa học, kỹ thuật và công nghệ đã trở thành một phần thiết yếu trong hoạt động sản xuất và đời sống của mỗi người. Không chỉ dừng lại ở sự ảnh hưởng đến cá nhân, cách mạng công nghệ 4.0 còn tác động mạnh mẽ đến các quốc gia, chính phủ, doanh nghiệp và ngành nghề, tạo tiền đề cho kinh tế số phát triển với những bước tiến nhảy vọt.

Một trong số những xu thế phát triển của cuộc cách mạng công nghệ 4.0 đó là sự phát triển của Trí tuệ nhân tạo hay trí thông minh nhân tạo (Artificial Intelligence-viết tắt là AI). Trí tuệ nhân tạo là một ngành thuộc lĩnh vực Khoa học máy tính (Computer Science), là trí tuệ do con người lập trình tạo nên với mục tiêu giúp máy tính có thể tự động hóa các hành vi thông minh như con người. Và trong cuộc sống, nhiều bài toán được áp dụng trí tuệ nhân tạo như: Tìm kiếm đường đi trong mê cung, nhận diện và phân tích hình ảnh, game thông minh… giúp ích rất nhiều trong nghiên cứu và phát triển của đời sống – xã hội con người.

Bài toán tìm đường đi trong ma trận là một vấn đề quan trọng và phổ biến trong lĩnh vực trí tuệ nhân tạo và thị giác máy tính. Mục tiêu chính của bài toán này là tìm cách di chuyển từ một điểm xuất phát đến một điểm đích trong một mê cung hoặc ma trận với các ràng buộc và hạn chế cụ thể. Tùy vào các thuật toán tìm đường khác nhau sẽ cho ra những hướng di chuyển khác nhau với những ưu điểm và hạn chế cụ thể

Bài toán tìm đường trong mệ cung được quy về việc tìm đường trong một ma trận kích thước NxM (N, M thuộc N\*) sử dụng các phương pháp tìm kiếm để giải quyết, và đây là bài báo cáo lại chúng, bài báo cáo bao gồm nhiều chương. Nội dung sẽ là các khái niệm cơ bản, các thuật toán, ưu hạn chế của từng phương pháp.

# KHÁI QUÁT VỀ BÀI TOÁN VÀ CÁC KHÁI NIỆM CƠ BẢN

## Mô tả Ma Trận:

Bài toán bắt đầu bằng việc mô tả một ma trận, thường là một lưới hoặc bản đồ, trong đó mỗi ô của ma trận có thể có các giá trị khác nhau. Một số ô có thể là các ô bị cản trở (ví dụ: tường hoặc vật cản), và một số ô có thể là các ô trống hoặc điểm xuất phát và điểm đích.

## 1.2 Điểm Xuất Phát và Điểm Đích

Cần xác định điểm xuất phát ( nơi bắt đầu ) và điểm đích ( nơi muốn đến ). Thường thì chỉ có thể bắt đầu từ một ô cụ thể và muốn đến một ô khác.

## 1.3 Ràng Buộc và Hạn Chế

Ma trận có thể có các ràng buộc và hạn chế, chẳng hạn như các ô bị cản trở hoặc các ô có giá trị khác nhau, có thể tương ứng với các chi phí di chuyển khác nhau.

## 1.4 Thuật Toán Tìm Đường Đi

Cần áp dụng một thuật toán tìm đường đi để xác định đường đi tốt nhất từ điểm xuất phát đến điểm đích dựa trên mô tả của ma trận và các ràng buộc. Các thuật toán phổ biến bao gồm: Tìm kiếm mù ( Tìm kiếm theo chiều rộng, tìm kiếm theo chiều sâu), Tìm kiếm dựa trên kinh nghiệm ( Tìm kiếm A\*, Tìm kiếm háu ăn, Tím kiếm leo đồi,…) …

## 1.5 Quá Trình Tìm Kiếm

Thuật toán sẽ thực hiện một quá trình tìm kiếm từ điểm xuất phát, di chuyển qua các ô của ma trận dựa trên các quy tắc tìm đường đi, và tiếp tục cho đến khi nó đạt được điểm đích hoặc không thể tìm thấy đường đi hợp lệ.

## 1.6 Kết Quả Đường Đi

Kết quả cuối cùng của bài toán là đường đi từ điểm xuất phát đến điểm đích, cũng như các bước di chuyển hoặc các ô trung gian trên đường đi.

Bài toán tìm đường đi trong ma trận có nhiều ứng dụng thực tế, chẳng hạn như hệ thống dẫn đường GPS, robot tự hành, và trò chơi máy tính, ứng dụng chỉ đường ( Google maps,Định vị GPS…). Quá trình tìm kiếm đường đi phụ thuộc vào thuật toán được sử dụng và cách biểu diễn ma trận và ràng buộc cụ thể của vấn đề.

# KHÔNG GIAN TRẠNG THÁI VÀ CÁC HÀM LIÊN QUAN

## 2.1 Xây dựng ma trận.

* Matrix[5][5] là một mảng 2 chiều kích thước 5x5.(*Lưu với tên File Matrix.TXT*)

**A 0 0 0 0**

**1 1 1 1 0**

**1 0 0 0 0**

**1 0 1 0 1**

**1 0 0 0 B**

* Điểm xuất phát (Start) : Vị trí (0;0) trong ma trận.
* Điểm đích (Goal) : Vị trí (4;4) trong ma trận.
* Ràng buộc :

+ Vị trí hợp lệ được ký hiệu A hoặc 1 hoặc B.

+ Vị trí không hợp lệ ký hiệu 0.

## 2.2 Không gian trạng thái.

* N (Node) : Sự dịch chuyển của A thông qua các vị trí

hợp lệ để dến được B.

* S (Start) : A=Matrix[0][0].
* GD (Goal Description) : B=Matrix[4][4].
* Op (Operator) : A có thể dịch chuyển theo 04 hướng

+ Sang trái (Left).

+ Sang phải (Right).

+ Lê trên (Go up).

+ Xuống dưới (Down).

|  |
| --- |
| **F(n)=G(n)+H(n)** |

## 2.3 Hàm đánh giá Heuristic.

Trong đó:

* **F(n)** là hàm đánh giá Heuristic tại trạng thái n.
* **G(n)** là khoảng cách của trạng thái đầu (Start) đối với trạng thái n.
* **H(n)** là khoảng cách ngắn nhất của các ô hợp lệ từ trạng thái n đến Goal.

# PHƯƠNG PHÁP TÌM KIẾM MÙ (UNINFORMED BLIND SEARCH)

## TÌM KIẾM THEO DUYỆT CHIỀU RỘNG (BREADTH FIRST SEARCH - BFS)

### 3.1.1 Khái niệm BFS:

**Breadth First Search (BFS)**: Thuật toán Tìm kiếm theo chiều rộng (BFS) được sử dụng để tìm kiếm cấu trúc dữ liệu đồ thị cho một nút đáp ứng một bộ tiêu chí. Nó bắt đầu từ gốc của biểu đồ và truy cập tất cả các nút ở mức độ sâu hiện tại trước khi chuyển sang các nút ở mức độ sâu tiếp theo. Trong phương pháp này, phần nhấn mạnh nằm trên các đỉnh của đồ thị, một đỉnh được chọn lúc đầu sau đó được truy cập và đánh dấu. Các đỉnh liền kề với đỉnh được truy cập sau đó được truy cập và lưu trữ trong hàng đợi một cách tuần tự. Tương tự, các đỉnh được lưu trữ sau đó được xử lý từng cái một và các đỉnh liền kề của chúng được truy cập. Một nút được khám phá đầy đủ trước khi truy cập bất kỳ nút nào khác trong biểu đồ, nói cách khác, nó đi qua các nút chưa được khám phá nông nhất trước tiên.

### 3.1.2 Giải thuật BFS:

**STEP 01:**

|  |  |
| --- | --- |
| Open =[Matrix[0][0]]; | Close =[ ]; |

* Xét Matrix[0][0]:🡺 Chưa phải là trạng thái đích.
* Đưa Matrix[0][0] vào Close; Close =[Matrix[0][0]];
* Xét các con của trạng thái Matrix[0][0]: Matrix[0][1];
* Các con của Matrix[0][0] không nằm trong Open, Close.
* Đưa các con của Matrix[0][0] vào Open: Open =[Matrix[0][1]].

**STEP 02:**

|  |  |
| --- | --- |
| Open =[Matrix[0][1]]; | Close =[ Matrix[0][0]]; |

* Xét Matrix[0][1]:🡺 Chưa phải là trạng thái đích.
* Đưa Matrix[0][1] vào Close: Close =[ Matrix[0][0];Matrix[0][1]];
* Xét các con của trạng thái Matrix[0][1]: Matrix[0][0], Matrix[0][2];
* Loại bỏ các con của Matrix[0][1] đã có tồn tại trong Open, Close: Matrix[0][0].
* Thêm các con còn lại của Matrix[0][1] vào open: Open =[Matrix[0][2]];

**STEP 03:**

|  |  |
| --- | --- |
| Open =[Matrix[0][2]]; | Close =[ Matrix[0][0];Matrix[0][1]]; |

* Xét Matrix[0][2]:🡺 Chưa phải là trạng thái đích.
* Đưa Matrix[0][2] vào Close: Close =[ Matrix[0][0];Matrix[0][1]; Matrix[0][2]];
* Xét các con của trạng thái Matrix[0][2]: Matrix[0][1], Matrix[0][3];
* Loại bỏ các con của Matrix[0][2] đã có tồn tại trong Open, Close: Matrix[0][1].
* Thêm các con còn lại của Matrix[0][2] vào open: Open =[Matrix[0][3]];

**STEP 04:**

|  |  |
| --- | --- |
| Open =[Matrix[0][3]]; | Close =[ Matrix[0][0];Matrix[0][1]; Matrix[0][2]]; |

* Xét Matrix[0][3]:🡺 Chưa phải là trạng thái đích.
* Đưa Matrix[0][3] vào Close: Close =[ Matrix[0][0];Matrix[0][1]; Matrix[0][2]; Matrix[0][3]];
* Xét các con của trạng thái Matrix[0][3]: Matrix[0][2], Matrix[0][4];
* Loại bỏ các con của Matrix[0][3] đã có tồn tại trong Open, Close: Matrix[0][2].
* Thêm các con còn lại của Matrix[0][3] vào open: Open =[Matrix[0][4]];

**STEP 05:**

|  |  |
| --- | --- |
| Open =[Matrix[0][4]]; | Close =[ Matrix[0][0];Matrix[0][1]; Matrix[0][2]; Matrix[0][3]]; |

* Xét Matrix[0][4]:🡺 Chưa phải là trạng thái đích.
* Đưa Matrix[0][4] vào Close: Close=[Matrix[0][0];Matrix[0][1]; Matrix[0][2]; Matrix[0][3]; Matrix[0][4]];
* Xét các con của trạng thái Matrix[0][4]: Matrix[0][3], Matrix[1][4];
* Loại bỏ các con của Matrix[0][4] đã có tồn tại trong Open, Close: Matrix[0][3].
* Thêm các con còn lại của Matrix[0][4] vào open: Open =[Matrix[1][4]];

**STEP 06:**

|  |  |
| --- | --- |
| Open =[Matrix[1][4]]; | Close =[ Matrix[0][0];Matrix[0][1]; Matrix[0][2]; Matrix[0][3]; Matrix[0][4]]; |

* Xét Matrix[1][4]:🡺 Chưa phải là trạng thái đích.
* Đưa Matrix[1][4] vào Close: Close =[ Matrix[0][0];Matrix[0][1]; Matrix[0][2]; Matrix[0][3]; Matrix[0][4]; Matrix[1][4]]];
* Xét các con của trạng thái Matrix[1][4]: Matrix[0][4], Matrix[2][4];
* Loại bỏ các con của Matrix[1][4] đã có tồn tại trong Open, Close: Matrix[0][4].
* Thêm các con còn lại của Matrix[1][4] vào open: Open =[Matrix[2][4]];

**STEP 07:**

|  |  |
| --- | --- |
| Open =[Matrix[2][4]]; | Close =  [Matrix[0][0];Matrix[0][1]; Matrix[0][2]; Matrix[0][3]; Matrix[0][4]; Matrix[1][4]]; |

* Xét Matrix[2][4]:🡺 Chưa phải là trạng thái đích.
* Đưa Matrix[2][4] vào Close: Close =[ Matrix[0][0];Matrix[0][1]; Matrix[0][2]; Matrix[0][3]; Matrix[0][4]; Matrix[1][4]]; Matrix[2][4]];];
* Xét các con của trạng thái Matrix[2][4]: Matrix[1][4], Matrix[2][3];
* Loại bỏ các con của Matrix[2][4] đã có tồn tại trong Open, Close: Matrix[1][4].
* Thêm các con còn lại của Matrix[2][4] vào open: Open =[Matrix[2][3]];

**STEP 08:**

|  |  |
| --- | --- |
| Open =[Matrix[2][3]]; | Close =[ Matrix[0][0];Matrix[0][1]; Matrix[0][2]; Matrix[0][3]; Matrix[0][4]; Matrix[1][4]; Matrix[2][4]]; |

* Xét Matrix[2][3]:🡺 Chưa phải là trạng thái đích.
* Đưa Matrix[2][3] vào Close: Close =[ Matrix[0][0];Matrix[0][1]; Matrix[0][2]; Matrix[0][3]; Matrix[0][4]; Matrix[2][4], Matrix[2][3]];
* Xét các con của trạng thái Matrix[2][3]: Matrix[2][4], Matrix[2][2]; Matrix[3][3];
* Loại bỏ các con của Matrix[2][3] đã có tồn tại trong Open, Close: Matrix[2][4].
* Thêm các con còn lại của Matrix[2][3] vào open: Open = [Matrix[2][2]; Matrix[3][3]];

**STEP 09:**

|  |  |
| --- | --- |
| Open = [Matrix[2][2]; Matrix[3][3]]; | Close =[ Matrix[0][0];Matrix[0][1]; Matrix[0][2]; Matrix[0][3]; Matrix[0][4]; Matrix[1][4]; Matrix[2][4]; Matrix[2][3]]; |

* Xét Matrix[2][2]:🡺 Chưa phải là trạng thái đích.
* Đưa Matrix[2][2] vào Close: Close =[ Matrix[0][0];Matrix[0][1]; Matrix[0][2]; Matrix[0][3]; Matrix[0][4]; Matrix[2][4]; Matrix[2][3]; Matrix[2][2]];
* Xét các con của trạng thái Matrix[2][2]: Matrix[2][3], Matrix[2][1];
* Loại bỏ các con của Matrix[2][2] đã có tồn tại trong Open, Close: Matrix[2][3].
* Thêm các con còn lại của Matrix[2][2] vào open: Open = [Matrix[3][3]; Matrix[2][1]];

**STEP 10:**

|  |  |
| --- | --- |
| Open = [Matrix[3][3]; Matrix[2][1]]; | Close =[ Matrix[0][0];Matrix[0][1]; Matrix[0][2]; Matrix[0][3]; Matrix[0][4]; Matrix[1][4]; Matrix[2][4]; Matrix[2][3]; Matrix[2][2]]; |

* Xét Matrix[3][3]:🡺 Chưa phải là trạng thái đích.
* Đưa Matrix[3][3] vào Close: Close =[ Matrix[0][0];Matrix[0][1]; Matrix[0][2]; Matrix[0][3]; Matrix[0][4]; Matrix[2][4]; Matrix[2][3]; Matrix[2][2]; Matrix[3][3]];
* Xét các con của trạng thái Matrix[3][3]: Matrix[4][3], Matrix[2][3];
* Loại bỏ các con của Matrix[3][3] đã có tồn tại trong Open, Close: Matrix[2][3].
* Thêm các con còn lại của Matrix[2][4] vào open: Open = [Matrix[2][1]; Matrix[4][3]];

**STEP 11:**

|  |  |
| --- | --- |
| Open = [Matrix[2][1]; Matrix[4][3];]; | Close =[ Matrix[0][0];Matrix[0][1]; Matrix[0][2]; Matrix[0][3]; Matrix[0][4]; Matrix[1][4]; Matrix[2][4]; Matrix[2][3]; Matrix[2][2]; Matrix[3][3]]; |

* Xét Matrix[2][1]:🡺 Chưa phải là trạng thái đích.
* Đưa Matrix[2][1] vào Close: Close =[ Matrix[0][0];Matrix[0][1]; Matrix[0][2]; Matrix[0][3]; Matrix[0][4]; Matrix[2][4]; Matrix[2][3]; Matrix[2][2]; Matrix[3][3]; Matrix[2][1]];
* Xét các con của trạng thái Matrix[2][1]: Matrix[2][2], Matrix[3][1];
* Loại bỏ các con của Matrix[2][1] đã có tồn tại trong Open, Close: Matrix[2][2].
* Thêm các con còn lại của Matrix[2][4] vào open: Open = [Matrix[4][3]; Matrix[3][1]];

**STEP 12:**

|  |  |
| --- | --- |
| Open = [Matrix[4][3]; Matrix[3][1]]; | Close =[ Matrix[0][0];Matrix[0][1]; Matrix[0][2]; Matrix[0][3]; Matrix[0][4]; Matrix[1][4]; Matrix[2][4]; Matrix[2][3]; Matrix[2][2]; Matrix[3][3]; Matrix[2][1]]; |

* Xét Matrix[4][3]:🡺 Chưa phải là trạng thái đích.
* Đưa Matrix[4][3] vào Close: Close =[ Matrix[0][0];Matrix[0][1]; Matrix[0][2]; Matrix[0][3]; Matrix[0][4]; Matrix[2][4]; Matrix[2][3]; Matrix[2][2]; Matrix[3][3]; Matrix[2][1]; Matrix[4][3]];
* Xét các con của trạng thái Matrix[4][3]: Matrix[3][3], Matrix[4][4], Matrix[4][2] ;
* Loại bỏ các con của Matrix[2][1] đã có tồn tại trong Open, Close: Matrix[3][3].

Thêm các con còn lại của Matrix[2][4] vào open: Open = [Matrix[3][1]; Matrix[4][4]; Matrix[4][2]];

**STEP 13:**

|  |  |
| --- | --- |
| Open = [Matrix[3][1]; Matrix[4][4]; Matrix[4][2]]; | Close =[ Matrix[0][0];Matrix[0][1]; Matrix[0][2]; Matrix[0][3]; Matrix[0][4]; Matrix[1][4]; Matrix[2][4]; Matrix[2][3]; Matrix[2][2]; Matrix[3][3]; Matrix[2][1]; Matrix[4][3]]; |

* Xét Matrix[3][1]:🡺 Chưa phải là trạng thái đích.
* Đưa Matrix[3][1] vào Close: Close =[ Matrix[0][0];Matrix[0][1]; Matrix[0][2]; Matrix[0][3]; Matrix[0][4]; Matrix[2][4]; Matrix[2][3]; Matrix[2][2]; Matrix[3][3]; Matrix[2][1]; Matrix[4][3]; Matrix[3][1]];
* Xét các con của trạng thái Matrix[3][1]: Matrix[4][1], Matrix[2][1].
* Loại bỏ các con của Matrix[2][1] đã có tồn tại trong Open, Close: Matrix[2][1].

Thêm các con còn lại của Matrix[2][4] vào open: Open = [Matrix[4][4]; Matrix[4][2]; Matrix[4][1]];

**STEP 14:**

|  |  |
| --- | --- |
| Open = [Matrix[4][4]; Matrix[4][2]; Matrix[4][1]]; | Close =[ Matrix[0][0];Matrix[0][1]; Matrix[0][2]; Matrix[0][3]; Matrix[0][4]; Matrix[1][4]; Matrix[2][4]; Matrix[2][3]; Matrix[2][2]; Matrix[3][3]; Matrix[2][1]; Matrix[4][3]; Matrix[3][1]]; |

* Xét Matrix[4][4]🡺 Trạng thái đích🡺Ngừng giải thuật.

Các bước của thuật toán tìm kiếm theo duyệt chiều rộng (BFS):

|  |  |  |  |
| --- | --- | --- | --- |
| 1. Matrix[0][0] | 4. Matrix[0][3] | 7. Matrix[2][4] | 10. Matrix[4][3] |
| 2. Matrix[0][1] | 5. Matrix[0][4] | 8. Matrix[2][3] | 11. Matrix[4][4] |
| 3. Matrix[0][2] | 6. Matrix[1][4] | 9. Matrix[3][3] |  |

## TÌM KIẾM THEO DUYỆT CHIỀU SÂU (DEPTH FIRST SEARCH-DFS)

### Khái niệm DFS:

**Phương pháp** **tìm kiếm sâu (DFS):** sử dụng ngăn xếp để lưu trữ các đỉnh đã truy cập. DFS là phương pháp dựa trên cạnh và hoạt động theo kiểu đệ quy trong đó các đỉnh được khám phá dọc theo một đường dẫn (cạnh). Việc thăm dò một nút bị đình chỉ ngay khi tìm thấy một nút chưa được khám phá khác và các nút chưa được khám phá sâu nhất được duyệt qua trước hết. DFS di chuyển / truy cập mỗi đỉnh chính xác một lần và mỗi cạnh được kiểm tra chính xác hai lần.

### Giải thuật DFS:

**STEP 01:**

|  |  |
| --- | --- |
| Open =[Matrix[0][0]]; | Close =[ ]; |

* Xét Matrix[0][0]:🡺 Chưa phải là trạng thái đích.
* Đưa Matrix[0][0] vào Close: Close =[Matrix[0][0]];
* Xét các con của trạng thái Matrix[0][0]: Matrix[0][1];
* Các con của Matrix[0][0] không nằm trong Open, Close.
* Đưa các con của Matrix[0][0] vào Open: Open =[Matrix[0][1]].

**STEP 02:**

|  |  |
| --- | --- |
| Open =[Matrix[0][1]]; | Close =[ Matrix[0][0]]; |

* Xét Matrix[0][1]:🡺 Chưa phải là trạng thái đích.
* Đưa Matrix[0][1] vào Close: Close =[ Matrix[0][0];Matrix[0][1]];
* Xét các con của trạng thái Matrix[0][1]: Matrix[0][0], Matrix[0][2];
* Loại bỏ các con của Matrix[0][1] đã có tồn tại trong Open, Close: Matrix[0][0].
* Thêm các con còn lại của Matrix[0][1] vào open: Open =[Matrix[0][2]];

**STEP 03:**

|  |  |
| --- | --- |
| Open =[Matrix[0][2]]; | Close =[ Matrix[0][0];Matrix[0][1]]; |

* Xét Matrix[0][2]:🡺 Chưa phải là trạng thái đích.
* Đưa Matrix[0][2] vào Close: Close =[ Matrix[0][0];Matrix[0][1]; Matrix[0][2]];
* Xét các con của trạng thái Matrix[0][2]: Matrix[0][1], Matrix[0][3];
* Loại bỏ các con của Matrix[0][2] đã có tồn tại trong Open, Close: Matrix[0][1].
* Thêm các con còn lại của Matrix[0][2] vào open: Open =[Matrix[0][3]];

**STEP 04:**

|  |  |
| --- | --- |
| Open =[Matrix[0][3]]; | Close =[ Matrix[0][0];Matrix[0][1]; Matrix[0][2]]; |

* Xét Matrix[0][3]:🡺 Chưa phải là trạng thái đích.
* Đưa Matrix[0][3] vào Close: Close =[ Matrix[0][0];Matrix[0][1]; Matrix[0][2]; Matrix[0][3]];
* Xét các con của trạng thái Matrix[0][3]: Matrix[0][2], Matrix[0][4];
* Loại bỏ các con của Matrix[0][3] đã có tồn tại trong Open, Close: Matrix[0][2].
* Thêm các con còn lại của Matrix[0][3] vào open: Open =[Matrix[0][4]];

**STEP 05:**

|  |  |
| --- | --- |
| Open =[Matrix[0][4]]; | Close =[ Matrix[0][0];Matrix[0][1]; Matrix[0][2]; Matrix[0][3]]; |

* Xét Matrix[0][4]:🡺 Chưa phải là trạng thái đích.
* Đưa Matrix[0][4] vào Close: Close =[ Matrix[0][0];Matrix[0][1]; Matrix[0][2]; Matrix[0][3]; Matrix[0][4]];
* Xét các con của trạng thái Matrix[0][4]: Matrix[0][3], Matrix[1][4];
* Loại bỏ các con của Matrix[0][4] đã có tồn tại trong Open, Close: Matrix[0][3].
* Thêm các con còn lại của Matrix[0][4] vào open: Open =[Matrix[1][4]];

**STEP 06:**

|  |  |
| --- | --- |
| Open =[Matrix[1][4]]; | Close =[ Matrix[0][0];Matrix[0][1]; Matrix[0][2]; Matrix[0][3]; Matrix[0][4]]; |

* Xét Matrix[1][4]:🡺 Chưa phải là trạng thái đích.
* Đưa Matrix[1][4] vào Close: Close =[Matrix[0][0];Matrix[0][1]; Matrix[0][2]; Matrix[0][3]; Matrix[0][4]; Matrix[1][4]]];
* Xét các con của trạng thái Matrix[1][4]: Matrix[0][4], Matrix[2][4];
* Loại bỏ các con của Matrix[1][4] đã có tồn tại trong Open, Close: Matrix[0][4].
* Thêm các con còn lại của Matrix[1][4] vào open: Open =[Matrix[2][4]];

**STEP 07:**

|  |  |
| --- | --- |
| Open =[Matrix[2][4]]; | Close =  [Matrix[0][0];Matrix[0][1]; Matrix[0][2]; Matrix[0][3]; Matrix[0][4]; Matrix[1][4]]; |

* Xét Matrix[2][4]:🡺 Chưa phải là trạng thái đích.
* Đưa Matrix[2][4] vào Close: Close=[Matrix[0][0];Matrix[0][1]; Matrix[0][2]; Matrix[0][3]; Matrix[0][4]; Matrix[1][4]]; Matrix[2][4]];];
* Xét các con của trạng thái Matrix[2][4]: Matrix[1][4], Matrix[2][3];
* Loại bỏ các con của Matrix[2][4] đã có tồn tại trong Open, Close: Matrix[1][4].
* Thêm các con còn lại của Matrix[2][4] vào open: Open =[Matrix[2][3]];

**STEP 08:**

|  |  |
| --- | --- |
| Open =[Matrix[2][3]]; | Close =[ Matrix[0][0];Matrix[0][1]; Matrix[0][2]; Matrix[0][3]; Matrix[0][4]; Matrix[1][4]; Matrix[2][4]]; |

* Xét Matrix[2][3]:🡺 Chưa phải là trạng thái đích.
* Đưa Matrix[2][3] vào Close: Close =[ Matrix[0][0];Matrix[0][1]; Matrix[0][2]; Matrix[0][3]; Matrix[0][4]; Matrix[2][4], Matrix[2][3]];
* Xét các con của trạng thái Matrix[2][3]: Matrix[2][4], Matrix[2][2]; Matrix[3][3];
* Loại bỏ các con của Matrix[2][3] đã có tồn tại trong Open, Close: Matrix[2][4].
* Thêm các con còn lại của Matrix[2][3] vào open: Open = [Matrix[2][3]; Matrix[3][3];

**STEP 09:**

|  |  |
| --- | --- |
| Open = [Matrix[2][2]; Matrix[3][3]; | Close =[ Matrix[0][0];Matrix[0][1]; Matrix[0][2]; Matrix[0][3]; Matrix[0][4]; Matrix[2][4], Matrix[2][3]]; |

* Xét Matrix[3][3]:🡺 Chưa phải là trạng thái đích.
* Đưa Matrix[3][3] vào Close: Close =[ Matrix[0][0];Matrix[0][1]; Matrix[0][2]; Matrix[0][3]; Matrix[0][4]; Matrix[2][4], Matrix[2][3]; Matrix[3][3]];
* Xét các con của trạng thái Matrix[3][3]: Matrix[2][3], Matrix[4][3]
* Loại bỏ các con của Matrix[3][3] đã có tồn tại trong Open, Close: Matrix[2][3].
* Thêm các con còn lại của Matrix[2][3] vào open: Open = [Matrix[2][2]; Matrix[4][3]]

**STEP 10:**

|  |  |
| --- | --- |
| Open = [Matrix[2][2]; Matrix[4][3]]; | Close =[ Matrix[0][0];Matrix[0][1]; Matrix[0][2]; Matrix[0][3]; Matrix[0][4]; Matrix[2][4], Matrix[2][3]; Matrix[3][3]]; |

* Xét Matrix[4][3]:🡺 Chưa phải là trạng thái đích.
* Đưa Matrix[4][3] vào Close: Close =[ Matrix[0][0];Matrix[0][1]; Matrix[0][2]; Matrix[0][3]; Matrix[0][4]; Matrix[2][4], Matrix[2][3]; Matrix[3][3]; Matrix[4][3]];
* Xét các con của trạng thái Matrix[4][3]: Matrix[3][3], Matrix[4][4], Matrix[4][2]
* Loại bỏ các con của Matrix[3][3] đã có tồn tại trong Open, Close: Matrix[3][3].
* Thêm các con còn lại của Matrix[2][3] vào open: Open = [Matrix[2][2]; Matrix[4][2]; Matrix[4][4]].

**STEP 11:**

|  |  |
| --- | --- |
| Open = [Matrix[2][2]; Matrix[4][2]; Matrix[4][4]]. | Close =[ Matrix[0][0];Matrix[0][1]; Matrix[0][2]; Matrix[0][3]; Matrix[0][4]; Matrix[2][4], Matrix[2][3]; Matrix[3][3]; Matrix[4][3]]; |

* Xét Matrix[4][4]:🡺 Trạng thái đích 🡺 Ngừng giải thuật.

Các bước của thuật toán tìm kiếm theo duyệt chiều sâu (DFS):

|  |  |  |  |
| --- | --- | --- | --- |
| 1. Matrix[0][0] | 4. Matrix[0][3] | 7. Matrix[2][4] | 10. Matrix[4][3] |
| 2. Matrix[0][1] | 5. Matrix[0][4] | 8. Matrix[2][3] | 11. Matrix[4][4] |
| 3. Matrix[0][2] | 6. Matrix[1][4] | 9. Matrix[3][3] | s |

## SO SÁNH BFS VÀ DFS

1. **Giống nhau:**

Cả hai thuật toán đều dựa trên tìm kiếm mù, tìm kiếm bằng cách xếp chồng một cây lên biểu đồ (cây tìm kiếm). DFS và BFS đặt gốc của nó thành nút bắt đầu và phát triển nó bằng cách thêm các phần kế thừa của các trạng thái hiện tại của cây. Theo cách đó, DFS và BFS bao phủ toàn bộ biểu đồ cho đến khi chúng tìm thấy nút mục tiêu hoặc đi hết biểu đồ.

1. **Khác nhau:**

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **TIÊU CHÍ** | **BREADTH FIRST SEARCH (BFS)** | **DEPTH FIRST SEARCH DFS)** |
| 1 | Kết quả bài toán | Luôn tìm ra kết quả của bài toán | Kết quả còn phụ thuộc vào các duyệt của giải thuật, một số bài toán không tìm thấy được kết quả. |
| 2 | Cấu trúc dữ liệu được sử dụng để lưu trữ | Cấu trúc hàng đợi | Cấu trúc ngăn xếp |
| 3 | Cấu trúc của cây được xây dựng | Rộng và ngắn | Sâu và dài |

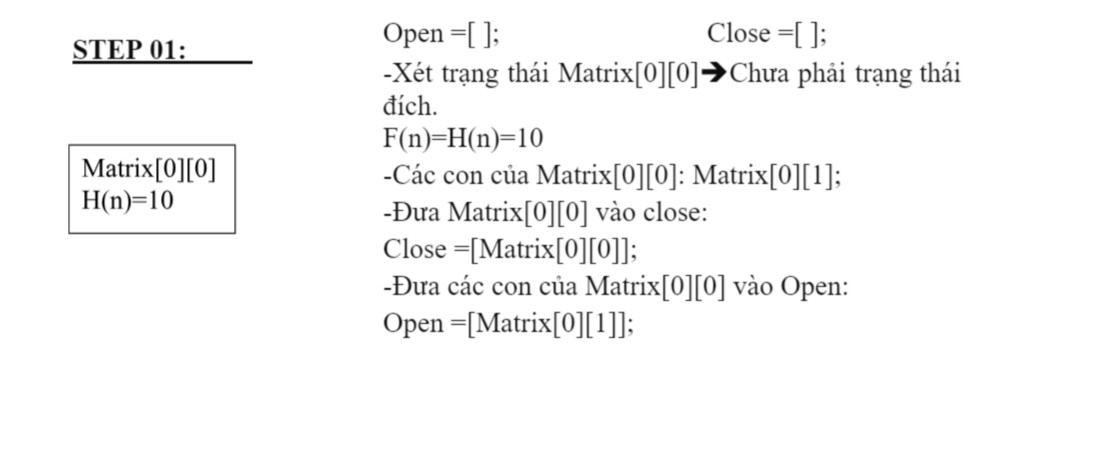
# TÌM KIẾM DỰA TRÊN KINH NGHIỆM (INFORMED/HEURISTIC SEARCH)

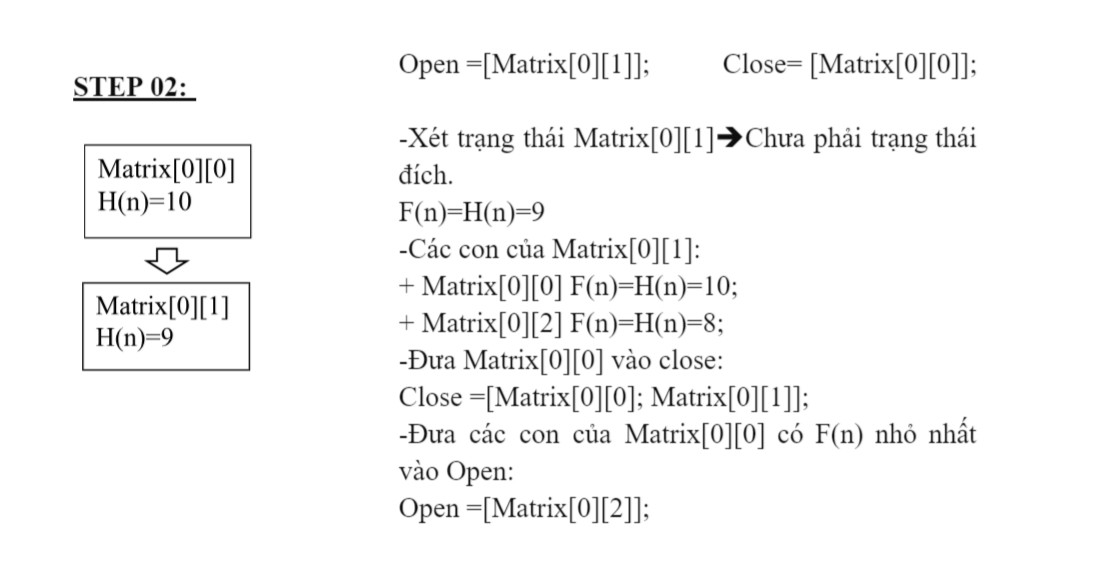
## TÌM KIẾM BẰNG PHƯƠNG PHÁP CỦA GIẢI THUẬT HÁU ĂN (GREEDY BEST-FIRST SEARCH)

Tìm kiếm háu ăn chỉ dùng hàm Heuristic H(n) để lượng giá các nút

|  |
| --- |
| **F(n)=H(n)** |

Xét các trạng thái có F(n) thấp nhất tiếp theo.





## 

dd

dd

ddd

dddd

dddd

## TÌM KIẾM BẰNG PHƯƠNG PHÁP CỦA GIẢI THUẬT A\*

Giải thuật A\* là một trường hợp đặc biệt Best first search (việc cập nhật lại đường đi dựa trên giá trị G(n) thay vì dựa trên giá trị **F(n)** tổng quát).

|  |
| --- |
| **F(n)=G(n)+H(n).** |

**H(n)** phụ thuộc và trạng thái n nên **F(n)** chỉ thay đổi khi **G(n)** thay dổi hay cách nói khác khi ta tìm được một đường đi mới đến n tốt hơn đường đi cũ=> cập nhật lại **G(n)** khi đường đi mới tốt hơn

Một trạng thái n tùy ý sẽ gồm 4 yếu tố **[G(n), H(n), F(n), Cha(n)]**. Trong đó **Cha(n)** là nút Cha của nút đang xét n.

### Tổng quát về giải thuật A\*

1. **Khái niệm:**

* A\* là giải thuật tìm kiếm trong đồ thị, tìm đường đi từ một nút hiện tại đến nút đích có sử dụng hàm để ước lượng khoảng cách hay còn gọi là hàm Heuristic.
* Từ trạng thái hiện tại, A\* xây dựng tất cả các đường đi có thể đi dùng hàm ước lược khoảng cách (hàm Heuristic) để đánh giá đường đi tốt nhất có thể đi.
* Tùy theo mỗi yêu cầu khác nhau mà hàm Heuristic sẽ được đánh giá khác nhau.
* Mục tiêu chung của giải thuật A\* là chi phí tốt nhất và số bước duyệt là ít nhất.
* Giải thuật A\* là một dạng cải tiến từ giải thuật tìm kiếm háu ăn (Greedy Best-first search) với ý tưởng là tránh việc xét (phát triển) các nhánh tìm kiếm đã xác định (cho đến thời điểm hiện tại) là có chi phí cao

1. **Mô tả giải thuật:**

* ***Open***: tập các trạng thái đã được sinh ra nhưng chưa được xét đến.
* ***Close***: tập các trạng thái đã được xét đến.
* ***Cost(p, q)***:là khoảng cách giữa p, q.
* Mỗi trạng thái p tùy ý gồm 4 yếu tố:
  + ***G(p)***: khoảng cách từ trạng thái đầu đến trạng thái hiện tại p.
  + ***H(p)***: giá trị được lượng giá từ trạng thái hiện tại đến trạng thái đích.
  + ***F(p) = G(p) + H(p)***
  + ***Cha(p)***: nút cha của nút đang xét p
* ***Bước 1:***
  + **Open:** = {p}
  + **Close:** = {}
* ***Bước 2:*** while (Open !={})
  + Chọn trạng thái (đỉnh) tốt nhất p trong **Open** (xóa p khỏi **Open**).
  + Nếu p là trạng thái kết thúc thì thoát.
  + Ngược lại: chuyển p qua Close và tạo ra các trạng thái kế tiếp q sau p.
    - Nếu q đã có trong Open
      * Nếu g(q) > g(p) + Cost(p, q)
        + g(q) = g(p) + Cost(p, q)
        + f(q) = g(q) + h(q)
        + cha(q) = p (đỉnh cha của q là p)
    - Nếu q chưa có trong Open
      * g(q) = g(p) + cost(p, q)
      * f(q) = g(q) + h(q)
      * cha(q) = p
      * Thêm q vào Open
    - Nếu q có trong Close
      * Nếu g(q) > g(p) + Cost(p, q)

Bỏ q khỏi Close

Thêm q vào Open

* + Sắp xếp các trạng thái có trong Open để trạng thái tốt nhất (Theo thứ tự **tăng dần của F(n)**).

### Giải thuật của thuật toán A\*

dddd

dddd

dddd

dddd

dddd

ddddd

ddddd

dddddd

ddddd

dddddd

dddddd

dddddd

d

d

dddddd

dddddd

dddddd

dddddd

dddddd

dddddd

dddddd

ddddddd

ddddd

dddddd

dddddd

## TÌM KIẾM BẰNG PHƯƠNG PHÁP CỦA GIẢI THUẬT LEO ĐỒI (HILL CLIMBING)

## Vài nét về giải thuật leo đồi (Hill Climbing)

* Là thuật toán tìm kiếm cục bộ.
* Ý tưởng: Tìm kiếm trạng thái đích bằng cách hướng tới trạng thái tốt hơn trạng thái hiện tại (leo lên đỉnh của một ngọn đồi).

## Đặc điểm của giải thuật leo đồi (Hill Climbing)

* Trạng thái con tốt nhất sẽ được chọn cho bước tiếp theo.
* Không lưu giữ bất kỳ thông tin về các nút cha và anh em.
* Quá trình tìm kiếm kết thúc khi:
  + - * Gặp trạng thái đích.
      * Trạng thái kế tiếp “xấu” hơn trạng thái hiện tại.
* Sử dụng hàm đánh giá Heuristic để xác định trạng thái nào là tốt nhất.

## Các loại của giải thuật leo đồi

1. **Leo đồi đơn giản (Simple Hill Climbing)**

* Kiểm tra từng nút lân cận và chọn nút lân cận đầu tiên để tối ưu hóa chi phí hiện tại làm nút tiếp theo.
* Mô tả giải thuật:
* Đánh giá trạng thái ban đầu:
  + Nếu là đích: dừng lại và trả về thành công.
  + Ngược lại: đặt trạng thái ban đầu là trạng thái hiện tại.
* Lặp lại cho đến khi tìm thấy trạng thái giải pháp hoặc không có toán tử mới nào có thể áp dụng cho trạng thái hiện tại.
  + Chọn smột trạng thái chưa được áp dụng cho trạng thái hiện tại và áp dụng nó để tạo ra trạng thái mới.
  + Với trạng thái mới:
    - Nếu trạng thái hiện tại là trạng thái đích thì dừng lại và trả về thành công.
    - Nếu nó tốt hơn trạng thái hiện tại thì hãy biến nó thành trạng thái hiện tại và tiếp tục.
    - Nếu nó không tốt hơn trạng thái hiện tại thì hãy tiếp tục vòng lặp cho đến khi tìm thấy giải pháp.
* Thoát khỏi hàm

1. **Leo đồi dốc nhất (Steepest\_Ascent Hill Climbing)**

* Đầu tiên, kiểm tra tất cả các nút lân cận và sau đó chọn nút gần nhất với trạng thái giải pháp cho nút tiếp theo.
* Mô tả giải thuật:
* Đánh giá trạng thái ban đầu:
  + Nếu là đích: dừng lại và trả về thành công.
  + Ngược lại: đặt trạng thái ban đầu là trạng thái hiện tại.
* Lặp lại cho đến khi tìm thấy trạng thái giải pháp hoặc không có toán tử mới nào có thể áp dụng cho trạng thái hiện tại.
  + Chọn một trạng thái chưa được áp dụng cho trạng thái hiện tại.
  + Khởi tạo 'trạng thái tốt nhất' mới bằng trạng thái hiện tại và áp dụng nó để tạo ra trạng thái mới.
  + Với trạng thái mới:
    - Nếu trạng thái hiện tại là trạng thái đích thì dừng lại và trả về thành công.
    - Nếu nó tốt hơn trạng thái tốt nhất thì hãy đặt nó thành trạng thái tốt nhất, nếu không thì tiếp tục vòng lặp với một trạng thái mới khác.
  + Tạo trạng thái tốt nhất làm trạng thái hiện tại và lặp lại bước thứ hai.
* Thoát khỏi hàm

1. **Leo đồi ngẫu nhiên (Stochastic Hill Climbing)**

* Không kiểm tra tất cả các nút lân cận trước khi quyết định chọn nút nào mà chỉ chọn ngẫu nhiên một nút lân cận và quyết định (dựa trên mức độ cải thiện của nút lân cận đó) xem nên di chuyển đến nút lân cận đó hay kiểm tra nút khác.
* Mô tả giải thuật:
* Đánh giá trạng thái ban đầu:
  + Nếu là đích: dừng lại và trả về thành công.
  + Ngược lại: đặt trạng thái ban đầu là trạng thái hiện tại.
* Lặp lại cho đến khi tìm thấy trạng thái giải pháp hoặc không có toán tử mới nào có thể áp dụng cho trạng thái hiện tại.
  + Chọn một trạng thái chưa được áp dụng cho trạng thái hiện tại.
  + Áp dụng hàm đánh giá kế tiếp cho trạng thái hiện tại và tạo ra tất cả các trạng thái lân cận.
  + Trong số các trạng thái lân cận được tạo tốt hơn trạng thái hiện tại, hãy chọn ngẫu nhiên một trạng thái (hoặc dựa trên một số hàm xác suất).
  + Nếu trạng thái được chọn là trạng thái đích thì trả về thành công, nếu không thì đặt trạng thái đó thành trạng thái hiện tại và lặp lại bước 2.
* Thoát khỏi hàm

## Ảnh có chứa biểu đồ, hàng, Sơ đồ, văn bản Mô tả được tạo tự độngSơ đồ không gian trạng thái của giải thuật leo đồi (Hill Climbing)

* **Objective function** (trục y): có thể là hàm mục tiêu hoặc hàm chi phí
* **State space** (trục x): không gian trạng thái
* **Global maximum** (Cực đại toàn cục): trạng thái tốt nhất có thể của không gian trạng thái và có giá trị cao nhất của hàm mục tiêu.
* **Local maximum** (Cực đại cục bộ): trạng thái hiện tại tốt hơn các trạng thái lân cận của nó, nhưng cũng có một trạng thái khác cao hơn nó.

# CẤU TRÚC DỮ LIỆU VÀI GIẢI THUẬT

1. **Chương trình được viết trên C và sử dụng các cấu trúc dữ liệu sau:**

* Các thư viện:

#include<stdlib.h>

#include<stdio.h>s

#include<conio.h>

#include<graphics.h>

* Các biến nguyên : inx,iny,outx,outy;
* Các mảng số nguyên : int x[2500], y[2500], a[50][50], c[50][50], dx[5]={0,0,1,0,-1},dy[5]={0,-1,0,1,0};
* Mảng 2 chiều(Kiểu ký tự) : e[50][50];
* Mảng sâu ký tự : Namefile[50][50];
* Biến phụ : m,n,i,j,d,s;
* Tên tệp : FILE \*f,\*f1;
* **Input:**  :
* Dòng thứ nhất : Số hàng, số cột ma trận.
* Dòng thứ hai : Tọa độ xuất phát (hàng mấy, cột mấy),tọa độ kết thúc(hàng mấy, cột mấy).
* Dòng tiếp theo : Ma trận (gồm 0 và 1).
* Filelưu trữ ma trận.
* **Output**: :
* Tổng các bước của giải thuật.
* File chứa các bước cụ thể cụ thể của đường đi tên là OUTPUT.TXT.
* **Các hàm con cần thiết** :
* Void NHAP();
* Void BUOCDITHOAMAN();
* Void DFS();
* Void CHUANBI();
* Void TIM();
* Vòi XULY();
* Void PRINT();

1. **Chương trình cài đặt trên C:**

***#include<stdlib.h>***

***#include<stdio.h>***

***#include<conio.h>***

***#include<graphics.h>***

***int*** inx,iny,outx,outy;

***int*** x[2500],y[2500],a[50][50],c[50][50],

dx[5]={0,0,1,0,-1},

dy[5]={0,-1,0,1,0};

***char*** e[50][50], Namefile[50][50];

***int*** m,n,i,j,d,s;

FILE \*f,\*f1;

***void*** NHAP()

{

f=fopen("Matrix.TXT","r");

printf("\t Mo file thanh cong: \n\n");

fscanf(f,"%d %d",&m,&n);

printf("\t\t So Hang : %d \n",m);

printf("\t\t So Cot : %d \n",n);

fscanf(f,"%d %d %d %d",&inx,&iny,&outx,&outy);

printf("\t\t START : A[%2d][%2d] \n",inx-1,iny-1);

printf("\t\t GOAL DESCRIPTION: A[%2d][%2d] \n",outx-1,outy-1);

printf("\n\t An Enter de tiep tuc.......roi nhan di nhin gi nua\n");

printf("\n\t <<<<<<<<<<\n");

getch();

printf("\t HINH ANH CUA ME CUNG \n");

printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

for (i=0;i<m;++i)

{

printf("\t\t\t");

printf("\n");

for (j=0;j<n;++j)

{

fscanf(f,"%d",&a[i][j]);

printf(" %d",a[i][j]);

if(a[i][j]==1) system("color 4");

else system("color 1");

}

}

printf("\n");

printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

printf("\n\t An Enter de tiep tuc:.......\n\n");

getch();

fclose(f);

}

***int*** BUOCDITHOAMAN(int i,int j){

if ((i<0)||(i>=m)||(j<0)||(j>=n)) return 0;

if ((a[i][j]!=0)||(c[i][j]!=0)) return 0;

return 1;

}

***void*** DFS(int i,int j)

{

if ((i==outx-1)&&(j==outy-1)) d=1;

else

{

int k;

for (k=1;k<=4;++k)

if (BUOCDITHOAMAN(i+dx[k],j+dy[k]))

{

c[i+dx[k]][j+dy[k]]=k;

DFS(i+dx[k],j+dy[k]);

}

}

}

***void*** CHUANBI()

{

for (i=0;i<m;++i)

for (j=0;j<n;++j) c[i][j]=0;

c[inx-1][iny-1]=-1;

d=0;

}

***void*** TIM()

{

s=0;

x[0]=outx-1;

y[0]=outy-1;

i=outx-1;

j=outy-1;

***int*** k;

while (c[i][j]!=-1)

{

k=c[i][j];

i=i-dx[k];

j=j-dy[k];

s++;

x[s]=i;

y[s]=j;

}

}

***void*** XULY()

{

CHUANBI();

DFS(inx-1,iny-1);

if (d==1) TIM();

e[2\*(inx-1)][2\*(iny-1)]='1';

e[2\*(outx-1)][2\*(outy-1)]='2';

}

***void*** PRINT()

{

FILE \*f;

f=fopen("OUTPUT.TXT","w");

if (d != 1)

{

fprintf(f,"KET LUAN: KHONG CO DUONG DI");

printf("KET LUAN:\t KHONG CO DUONG DI\n\n");

}

if (d==1)

{

fprintf(f,"CAC TOA DO CAC DIEM TREN DUONG DI THOAT KHOI ME CUNG \n\n");

for (i=s;i>=0;i--)

fprintf(f,"\t%2d.a[%2d][%2d]-->\t||",i,x[i],y[i]);

printf("K.E.T Q.U.A C.U.A B.A.N L.A: \n");

printf("DO DAI DUONG DI TRONG ME CUNG LA: %d\n",s);

printf("Day cac toa do duong di da duoc luu ra file OUTPT.txt.....\n\n");

fprintf(f,"\n\n DO DAI DUONG DI TRONG ME CUNG LA : %d\n",s);

}

fclose(f);

}

**int main** ()

{

NHAP();

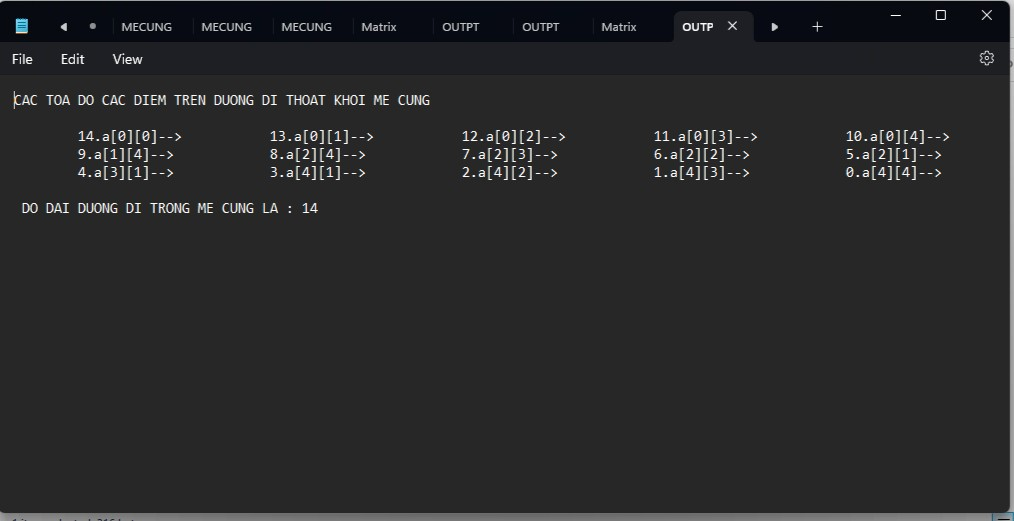
XULY();

PRINT();

system("color 6");

return 0;

}

**KẾT QUẢ:**

Kết quả trên giao diện DEV\_C:

Kết quả trên file OUTPUT.TXT:

# ỨNG DỤNG GIẢI THUẬT VÀO VIDEO GAME

## ỨNG DỤNG GIẢI THUẬT VÀO GAME GIẢI ĐỐ

Nhận thấy việc tối ưu khi có thể cập nhật lại đường đi khi sử dụng giải thuật tìm kiếm astar rất phù hợp với thể loại game giải đố hoặc đối kháng, do các tình huống trong game thay đổi thường xuyên, yêu cầu giải thuật phải đáp ứng theo, do đó, lựa chọn giải thuật A\* là lựa chọn tốt nhất.

## GIẢI THUẬT ĐƯỢC ỨNG DỤNG NHƯ THẾ NÀO?

Giải thuật được áp dụng cho nhân vật tìm đến vị trí đích (Goal), mục tiêu là tìm ra đường đi ngắn nhất, đồng thời tránh né được các chướng ngại vật khác

Tình huống của trò chơi: A là người chơi, B là con vật ăn trộm lương thực trong nông trại của A, A có nhiệm vụ phải bắt được B, không được để cho B thoát, và B có nhiệm vụ là trở về hang (thoát khỏi nông trại) mà không bị bắt. Lúc này, giải thuật tìm kiếm Astar được áp dụng vào B. Yêu cầu của B là phải tìm được đường đi ngắn nhất đến được nơi thoát (Goal) và đường đi đó phải tránh được các vật cản (Obstacles), cũng như là tránh khỏi vùng mà người chơi có thể bắt được (Player zone).

## NGUYÊN LÝ HOẠT ĐỘNG CỦA GẢM DỰA TRÊN GIẢI THUẬT

1. **Chơi theo lượt**: Sau khi người chơi A hành động xong thì sẽ đến con vật B, và ngược lại. Trò chơi kết thúc khi người chơi A bắt được B, hoặc B đến được Goal mà không bị người chơi bắt.

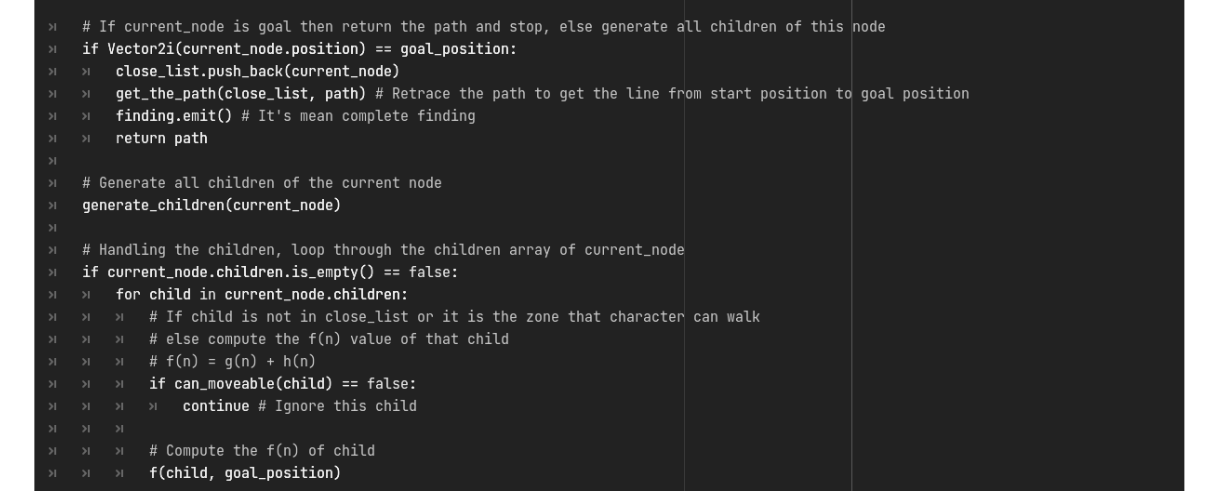
* **Trong lượt người chơi:** A có thể làm được những hành động sau: 1. Di chuyển, 2. Ghi chú.
* **Trong lượt của con vật:** B có thể: 1. Tìm kiếm đường đi ngắn nhất, 2. Di chuyển

1. **Giải thích:**

* **Di chuyển**: mỗi lần di chuyển được 1 ô (step), cũng tương ứng là 1 node
* **Ghi chú:** dùng bản đồ mô phỏng để vẽ ra đường đi mà bạn dự đoán con vật sẽ đi đến Goal.
* **Tìm kiếm đường đi ngắn nhất**: áp dụng giải thuật để chọn ra được đường sẽ đi, mỗi lần đến lượt, nếu người chơi di chuyển trên đường đi đã chọn trước đó, B cập nhật lại đường đi ngắn khác để tránh A. Trường hợp B không tìm được Goal thì sẽ chọn ngẫu nhiên 1 ô xung quanh để đi hoặc không làm gì (đứng in tại chỗ).
* **Mã nguồn:** (mã nguồn dưới đây được trích ra từ file code trong game của bạn Chiến, là thành viên nhóm, dưới đây là đoạn code áp dụng giải thuật tìm kiếm Astar, sử dụng ngôn ngữ lập trình GDScript, ngôn ngữ được thiết kế riêng dành cho công cụ thiết kế game Godot Engine).

1. **Input :** vị trí bắt đầu và vị trí đích đến.
2. **Output :** mảng gồm các node (là đường đi ngắn nhất).

## SẢN PHẨM GAME



# NGUỒN TÀI LIỆU THAM KHẢO

* Nguồn: https://www.GeeksforGeeks.com
  + Các loại của giải thuật leo đồi
* Nguồn: <https://www.baeldung.com/cs/dfs-vs-bfs>
  + Sự giống nhau và khác nhau giữa DFS và BFS