**THUYẾT TRÌNH**

Lời nói đầu tiên, cho phép em xin gửi lời chào đến thầy cũng như các bạn và anh chị có mặt trong buổi báo cao nhóm ngày hôm nay.

Nhóm bao gồm 4 thành viên:

1.

2.

3.

4.

Với đề tài: ***“CÁC PHƯƠNG PHÁP TÌM KIẾM ĐƯỜNG ĐI TRONG MÊ CUNG VÀ ỨNG DỤNG”.***

**Tại sao chọn đề tài?** 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.

**1.1 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.

**1.7 Ứng dụng**

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.

**XÂY DỰNG MA TRẬN:** Do ma trận đề bài của thầy đưa ra kích thước 21x31 khá lớn và mất nhiều thời gian do đó nhóm chúng em quyết định xây dựng lại ma trận mới có kích thước 5x5, và các bước thực hiện là tương tự

**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).

**HÀM HEURISTIC**

F(n)= G(n)+H(n)

* **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.

**--------------------------------NÓI TRONG 3-5 PHÚT-------------------------------------**

**Trong bài báo cao sẽ đưa ra 2 phương pháp tìm kiếm chính:**

1. **Tìm kiếm mù (Bao gồm BFS,DFS).**
2. **Tìm kiếm dựa trên kinh nghiệm (Bao gồm A\*, Leo đồi).**

**TÌM KIẾM 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.

N(Trạng thái) 🡪 k trạng thái con (k1, k2, … ,kn) xét xem trong nk trạng thái đó đã có chưa (nếu có thì loại), **sử dụng hàng đợi** để lưu trữ

Các trạng thái sinh:

**[0][0] -> [0][1] và [0][0] -> [0][2] và [0][1] -> [0][3] và [0][2] -> [0][4] và [0][3]**

**-> [1][4] và [0][3] -> [2][4] và [0][4] -> [1][4] và [2][3] -> [2][4] và [2][2] và [3][3]**

**[2][2] -> [2][1] và [2][3]**

**[3][3] -> [4][3] và [2][3]**

**[2][1] -> [2][2] và [3][1]**

**[4][3] -> [3][3] và [4][4] và [4][2]**

**[3][1] -> [4][1] và [2][1]**

**[4][4] 🡪 Đích**

**TÌM KIẾM 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).

N(Trạng thái) 🡪 k trạng thái con (k1, k2, … ,kn) xét xem trong nk trạng thái đó đã có chưa (nếu có thì loại), **sử dụng ngăn xếp** để lưu trữ.

**[0][0] -> [0][1] và [0][0] -> [0][2] và [0][1] -> [0][3] và [0][2] -> [0][4] và [0][3]**

**-> [1][4] và [0][3] -> [2][4] và [0][4] -> [1][4] và [2][3] -> [2][4] và [2][2] và [3][3]**

**[3][3] -> [4][3] và [2][3]**

**[4][3] -> [3][3] và [4][4] và [4][2]**

**[4][4] 🡪 Đích**

**SO SÁNH BFS VÀ DFS**

1. **Giống nhau:**

* **Dựa trên tìm kiếm mù**
* **Sử dụng cây tìm kiếm**
* **Đặt nút gốc làm nút bắt đầu và phát triển bằng cách sinh trạng thái bao phủ cây.**

1. **Khác nhau**

|  |  |  |
| --- | --- | --- |
|  | **BFS** | **DFS** |
| **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 đuocẹ kết quả. |
| **Cấu trúc dữ liệu được sử dụng để lưu trữ** | Cấu trúc hàng đợi | Cấu trúc ngăn xếp |
| **Cấu trúc của cây được xây dựng** | Rộng và ngắn | Sâu và dài |

**--------------------------------NÓI TRONG 7-10 PHÚT----------------------------------**

**TÌM KIẾM HÁU ĂN**

**TÌM KIẾM A\***

**Vẽ tay nhe—làm biến đánh máy—**

**TIM KIẾM LEO ĐỒI**

* Là thuật toán tìm kiếm cục bộ.
* Hướng tới trạng thái tốt hơn trạng thái hiện tại.
* Các dạng Leo đồi:

+ Leo đồi đơn giản.

+ Leo đồi dốc nhất.

+ Leo đồi ngẫu nhiên.

**--------------------------------NÓI TRONG 7-10 PHÚT-----------------------------------**

**TỔNG 20-25 PHÚT**