TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI

TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

BÁO CÁO BÀI TẬP LỚN

HỌC PHẦN TRÍ TUỆ NHÂN TẠO

ĐỀ TÀI: TÌM HIỂU THUẬT TOÁN TÌM KIẾM HEURISTIC VÀ ỨNG DỤNG VÀO BÀI TOÁN TRÒ CHƠI 8 SỐ

GVHD: Ths. Mai Thanh Hồng

Lớp: 20242IT6094003 Khoá: K18

Mã SV: 2023600614 Họ và tên: Đỗ Đức Anh

Mã SV: 2023602802 Họ và tên: Nguyễn Lan Anh

Mã SV: 2023600547 Họ và tên: Nguyễn Thị Thùy Dương

Mã SV: 2023600787 Họ và tên: Nguyễn Lâm Kha

Hà Nội, tháng 6 năm 2025

LỜI NÓI ĐẦU

Trong thời đại công nghệ phát triển mạnh mẽ như hiện nay, các thuật toán tìm kiếm đóng vai trò quan trọng trong việc giải quyết nhiều bài toán phức tạp, đặc biệt là trong lĩnh vực trí tuệ nhân tạo và xử lý thông tin. Trong đó, thuật toán tìm kiếm Heuristic nổi bật nhờ khả năng tối ưu hóa quá trình tìm kiếm, giúp giảm thiểu thời gian và tài nguyên tính toán so với các phương pháp truyền thống.

Bài toán Trò chơi 8 số là một trong những bài toán kinh điển thường được sử dụng để minh họa cho các thuật toán tìm kiếm. Bài toán yêu cầu người chơi sắp xếp các ô số trong một ma trận 3x3 về trạng thái đích bằng cách di chuyển các ô trống, với số bước di chuyển tối thiểu. Đây không chỉ là một trò chơi trí tuệ thú vị mà còn là một bài toán thách thức đối với các thuật toán tìm kiếm, đặc biệt khi kích thước bàn cờ tăng lên.

Báo cáo này tập trung tìm hiểu về thuật toán tìm kiếm Heuristic, bao gồm các phương pháp phổ biến như A*, Greedy Best-First Search, và ứng dụng chúng vào việc giải quyết bài toán Trò chơi 8 số.

Mục tiêu của báo cáo là cung cấp cái nhìn tổng quan về tìm kiếm Heuristic, đồng thời làm rõ cách áp dụng các thuật toán này vào bài toán thực tế. Kết quả nghiên cứu không chỉ có ý nghĩa lý thuyết mà còn mở ra hướng ứng dụng trong nhiều lĩnh vực khác như robot di chuyển, giải quyết vấn đề tối ưu hóa và trò chơi trí tuệ.

Chúng em hy vọng rằng báo cáo này sẽ là tài liệu hữu ích cho những ai quan tâm đến lĩnh vực trí tuệ nhân tạo, thuật toán tối ưu và ứng dụng của chúng trong thực tiễn. Trong quá trình thực hiện đề tài thì không thể tránh khỏi những sai sót, nhóm chúng em rất mong nhận được sự đánh giá và góp ý của cô để hoàn thiện cho đề tài này.

MỤC LỤC

LỜI NÓI ĐẦU 2
DANH SÁCH HÌNH ẢNH 5
CHƯƠNG 1: TÌM HIỂU VỀ KHÔNG GIAN TRẠNG THÁI, TÌM KIẾM HEURISTIC VÀ BÀI TOÁN TRÒ CHƠI 8 SỐ 6
1.1. Không gian trạng thái 6
1.1.1 Một số khái niệm 6
1.1.1.1 Mô tả trạng thái 6
1.1.1.2. Toán tử 7
1.1.2. Cấu trúc chung của bài toán tìm kiếm 7
1.1.3. Tìm kiếm lời giải trong không gian trạng thái 9
1.2. Các thuật toán tìm kiếm Heuristic 10
1.2.1. Tổng quan về giải thuật tìm kiếm Heuristic 10
1.2.1.1. Khái niệm 10
1.2.1.2. Vai trò 10
1.2.1.3. Ưu điểm 10
1.2.1.4. Phương pháp xây dựng 11
1.2.2. Tìm kiếm tối ưu 11
1.2.3. Thuật giải AKT 13
1.2.4. Thuật giải AT 15
1.2.5. Thuật giải A* 17
CHƯƠNG 2: THỰC HIỆN BÀI TOÁN TRÒ CHƠI 8 SỐ 18
2.1. Bài toán trò chơi 8 số 18
2.1.1. Giới thiệu bài toán 18
2.1.2. Không gian trạng thái của bài toán 19
2.1.3. Lý do chọn thuật toán A* để code bài toán trò chơi 8 số 20
2.2. Phân tích các thành phần để giải quyết bài toán 21
2.2.1 Giải thuật sử dụng 21
2.2.2. Cài đặt chi tiết 21

KẾT LUẬN 32
TÀI LIỆU THAM KHẢO 33

DANH SÁCH HÌNH ẢNH

Hình 1.1 Tìm đường đi từ A đến B 6
Hình 1.2 Đồ thị có trọng số 8
Hình 1.3 Một phần đồ thị biểu diễn bài toán trò chơi 8 số 8
Hình 1.4 Đồ thị tìm kiếm BeFS 12
Hình 2.1 Ví dụ cho trạng thái đích của bài toán 18
Hình 2.2 Trạng thái ban đầu 19
Hình 2.3 Trạng thái kết thúc 19
Hình 2.4 Ví dụ minh họa cho không gian trạng thái của bài toán 8 số 20
Hình 2.5 Code lớp State 22
Hình 2.6 Code lớp State (tiếp) 23
Hình 2.7 Code lớp Operator 24
Hình 2.8 Code lớp Operator (tiếp) 25
Hình 2.9 Code các hàm hỗ trợ 26
Hình 2.10. Code hàm Run 28
Hình 2.11 Code hàm nhập trạng thái ban đầu, trạng thái đích và chạy toàn bộ code 30
Hình 2.4 Kết quả sau khi chạy code 31

CHƯƠNG 1: TÌM HIỂU VỀ KHÔNG GIAN TRẠNG THÁI VÀ THUẬT TOÁN TÌM KIẾM HEURISTIC

1.1. Không gian trạng thái

1.1.1 Một số khái niệm

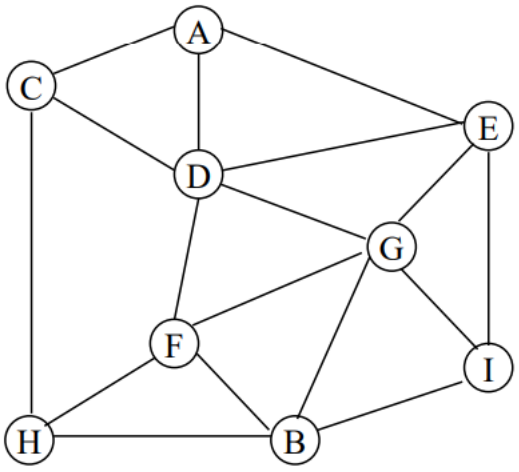
1.1.1.1 Mô tả trạng thái

Giải bài toán trong không gian trạng thái, trước hết phải xác định dạng mô tả trạng thái bài toán sao cho bài toán trở nên đơn giản hơn, phù hợp bản chất vật lý của bài toán (Có thể sử dụng các xâu ký hiệu, véctơ, mảng hai chiều, cây, danh sách,…).

Mỗi *trạng thái* chính là mỗi *hình trạng* của bài toán, các tình trạng ban đầu và tình trạng cuối của bài toán gọi là trạng thái đầu và trạng thái cuối.

Ví dụ 1.1: Bài toán khách du lịch

Một khách du lịch có trong tay bản đồ mạng lưới giao thông nối các thành phố trong một vùng lãnh thổ (hình 1.1). Du khách đang ở thành phố A và anh ta muốn tìm đường đi tới thành phố B.



Hình 1.1 Tìm đường đi từ A đến B

Trong bài toán này, các thành phố có trong bản đồ là các trạng thái. Thành phố A là trạng thái đầu còn B là trạng thái kết thúc. Với bài toán này ta có thể sử dụng cách biểu diễn của đồ thị (ma trận kề, ma trận trọng số, danh sách cạnh, danh sách kề).

1.1.1.2. Toán tử

Toán tử là các phép biến đổi từ trạng thái này sang trạng thái khác.

Có hai cách dùng để biểu diễn các toán tử:

- Biểu diễn như một hàm xác định trên tập các trạng thái và nhận giá trị cũng trong tập này.

- Biểu diễn dưới dạng các quy tắc sản xuất S →A có nghĩa là nếu có trạng thái S thì có thể đưa đến trạng thái A.

Trong ví dụ 1.1, mỗi toán tử là hành động đi từ thành phố này tới các thành phố khác.

1.1.2. Cấu trúc chung của bài toán tìm kiếm

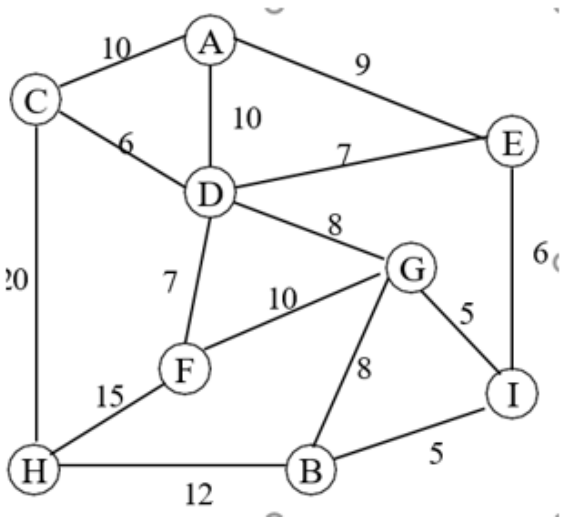
Từ các ví dụ ta có thể thấy nhiều bài toán đều có dạng "tìm đường đi trong đồ thị", hay nói một cách hình thức hơn là *"xuất phát từ một đỉnh của một đồ thị, tìm đường đi hiệu quả nhất đến một đỉnh nào đó"*. Từ đây ta có bài toán phát biểu trong không gian trạng thái.

Bài toán S: Cho trước hai trạng thái T0 và TG hãy xây dựng chuỗi trạng thái T0, T1, T2, ..., Tn-1, Tn = TG sao cho:

$\sum_{i=1}^{n} \cos t(T_{i-1}, T_i)$ thỏa mãn một điều kiện cho trước (thường là nhỏ nhất).

Trong đó, Ti thuộc tập hợp S (gọi là không gian trạng thái – state space) bao gồm tất cả các trạng thái có thể có của bài toán, và cost(Ti-1, Ti) là chi phí để biến đổi từ trạng thái Ti-1 sang trạng thái Ti. Dĩ nhiên, từ một trạng thái Ti-1 ta có nhiều cách để biến đổi sang trạng thái Ti. Khi nói đến một biến đổi cụ thể từ Ti-1 sang Ti ta sẽ dùng thuật ngữ hướng đi (với ngụ ý nói về sự lựa chọn).

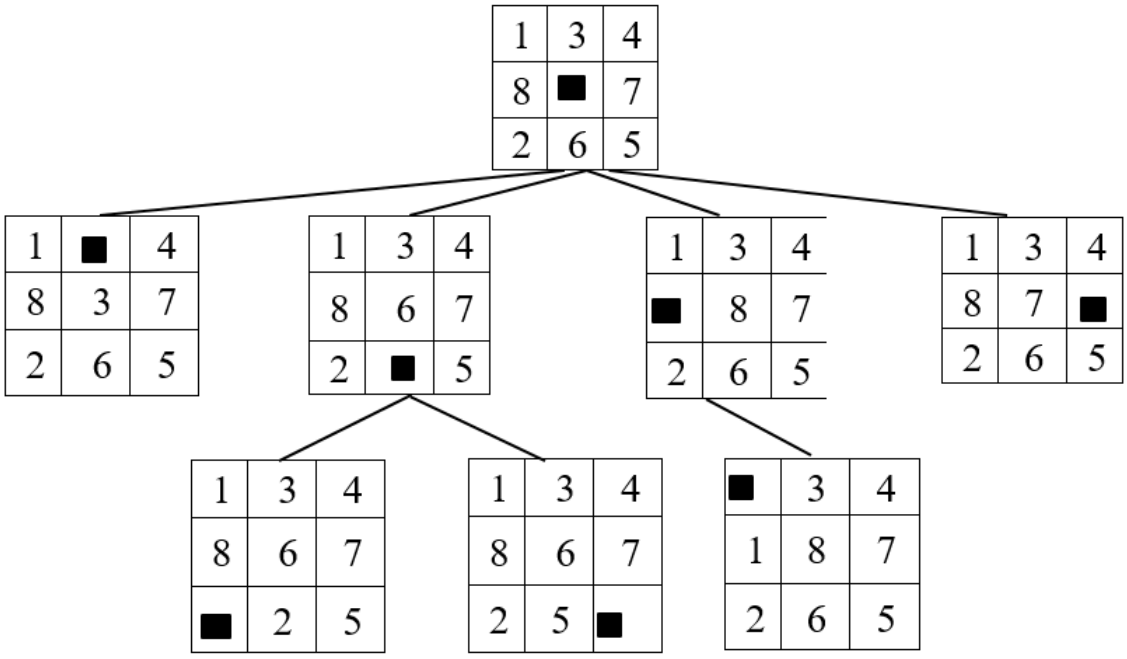
Mô hình chung của các bài toán phải giải quyết bằng phương pháp tìm kiếm lời giải. Không gian tìm kiếm là một tập hợp trạng thái - tập các nút của đồ thị. Chi phí cần thiết để chuyển từ trạng thái Ti-1 này sang trạng thái Ti được biểu diễn dưới dạng các con số nằm trên cung nối giữa hai nút.



Hình 1.2 Đồ thị có trọng số

Chẳng hạn, với đồ thị trong hình 1.2 ta có thể phát biểu bài toán: Hãy tìm đường đi ngắn nhất từ đỉnh A đến đỉnh B.

Đa số các bài toán thuộc dạng mà chúng ta đang mô tả đều có thể được biểu diễn dưới dạng đồ thị. Trong đó, mỗi trạng thái là một đỉnh của đồ thị. Tập hợp S bao gồm tất cả các trạng thái chính là tập hợp bao gồm tất cả các đỉnh của đồ thị. Việc biến đổi từ trạng thái Ti-1 sang trạng thái Ti là việc đi từ đỉnh đại diện cho Ti-1 sang đỉnh đại diện cho Ti theo cung nối giữa hai đỉnh này.



Hình 1.3 Một phần đồ thị biểu diễn bài toán trò chơi 8 số

Bài toán G: Cho đỉnh đầu So và tập các đỉnh đích Goal. Hãy tìm đường đi p (tối ưu) nào đó từ đỉnh So đến đỉnh nào đó thuộc tập Goal.

Từ những phân tích ở trên ta có thể thấy được sự tương đương giữa không gian trạng thái và đồ thị được tổng hợp ở bảng sau:

Không gian trạng thái	**Đồ thị**
- Trạng thái đầu	- Đỉnh đầu
- Trạng thái đích	- Đỉnh đích
- Toán tử dịch chuyển	- Cung
- Dãy các trạng thái	- Đường đi
- Bài toán S	- Bài toán G

1.1.3. Tìm kiếm lời giải trong không gian trạng thái

Quá trình tìm kiếm lời giải của bài toán được biểu diễn trong không gian trạng thái được xem như quá trình dò tìm trên đồ thị, xuất phát từ trạng thái ban đầu, thông qua các toán tử chuyển trạng thái, lần lượt đến các trạng thái tiếp theo cho đến khi gặp được trạng thái đích hoặc không còn trạng thái nào có thể tiếp tục được nữa.

Khi áp dụng các phương pháp tìm kiếm trong không gian trạng thái, người ta thường quan tâm đến các vấn đề sau:

- Kỹ thuật tìm kiếm lời giải
- Phương pháp luận của việc tìm kiếm
- Chiến lược tìm kiếm

Tuy nhiên, không phải các phương pháp này đều có thể áp dụng để giải quyết cho tất cả các bài toán phức tạp mà chỉ cho từng lớp bài toán.

Việc chọn chiến lược tìm kiếm cho bài toán cụ thể phụ thuộc nhiều vào các đặc trưng của bài toán.

Trong phần này, chúng ta sẽ nghiên cứu chiến lược tìm kiếm:

- Các kỹ thuật tìm kiếm *kinh nghiệm* (tìm kiếm heuristic): với kỹ thuật tìm kiếm này, phải dựa vào kinh nghiệm và sự hiểu biết của chúng ta về vấn đề cần giải quyết để xây dựng nên hàm đánh giá nhằm tìm ra các đỉnh tiềm năng dẫn đến lời giải. Trong số các trạng thái chờ phát triển, ta chọn trạng thái được đánh giá là tốt nhất để phát triển. Do đó tốc độ tìm kiếm sẽ nhanh hơn.

1.2. Các thuật toán tìm kiếm Heuristic

1.2.1. Tổng quan về giải thuật tìm kiếm Heuristic

1.2.1.1. Khái niệm

Trong bài toán tìm kiếm không gian trạng thái, Heuristic là các quy tắc hoặc phương pháp được sử dụng để chọn lựa nhánh có khả năng cao nhất dẫn đến một giải pháp chấp nhận được. Heuristic không phải là một giải pháp chính xác mà chỉ là phỏng đoán dựa trên thông tin có sẵn về các bước tiếp theo trong quá trình giải quyết vấn đề.

- Heuristic có thể được coi là tri thức rút ra từ kinh nghiệm hoặc trực giác của con người trong việc đưa ra các quyết định trong quá trình tìm kiếm.
- Heuristic có thể đúng hoặc sai,bởi vì chúng chỉ là các ước lượng về cách thức tiếp cận bài toán dựa trên những yếu tố chưa chắc chắn hoặc thiếu thông tin đầy đủ.
- Heuristic có khả năng dự đoán cách hành xử của không gian trạng thái ở các giai đoạn gần. Tuy nhiên, khi tìm kiếm ở các giai đoạn xa hơn, Heuristic ít khi cho kết quả chính xác vì thiếu thông tin hoặc vì không gian trạng thái quá phức tạp.

1.2.1.2. Vai trò

Heuristic được sử dụng trong các trường hợp:

- Vấn đề không có giải pháp chính xác vì thiếu thông tin hoặc mệnh đề không rõ ràng.
- Vấn đề có giải pháp chính xác nhưng chi phí tính toán quá cao để tìm ra

1.2.1.3. Ưu điểm

- Tìm được giải pháp tốt, mặc dù không đảm bảo là tối ưu.

- Nhanh chóng và chi phí thấp: Heuristic giúp giải bài toán nhanh hơn so với các giải thuật tối ưu, vì không cần duyệt qua toàn bộ không gian tìm kiếm.
- Dễ hiểu và gần gũi với cách suy nghĩ con người: Heuristic dựa trên kinh nghiệm và trực giác của con người.

1.2.1.4. Phương pháp xây dựng

Thuật toán Heuristic bao gồm Hàm đánh giá Heuristic và Thuật toán tìm kiếm. Một số nguyên lý cơ bản để xây dựng thuật toán Heuristic:

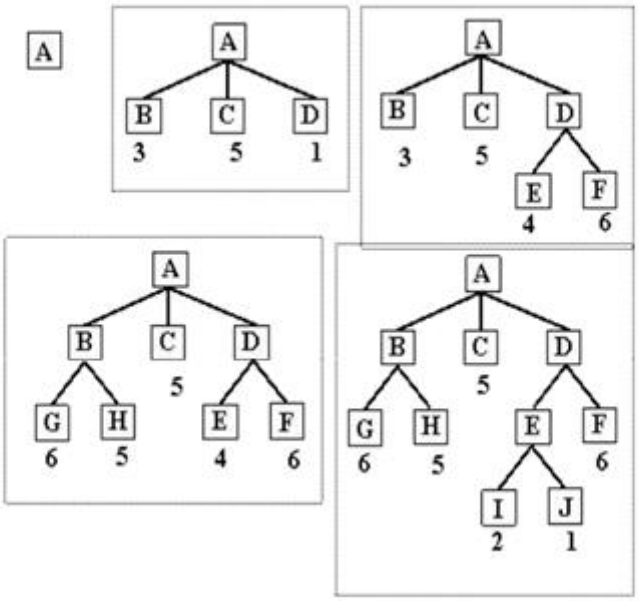
1. Nguyên lý vét cạn thông minh: Giới hạn không gian tìm kiếm hoặc sử dụng dò tìm đặc biệt dựa vào đặc thù bài toán để nhanh chóng tìm ra mục tiêu.
2. Nguyên lý tham lam (Greedy): Lựa chọn hành động tối ưu cục bộ ở mỗi bước mà không quan tâm đến tổng thể, giúp tìm giải pháp nhanh nhưng không luôn tối ưu.
3. Nguyên lý thứ tự: Thực hiện hành động theo cấu trúc thứ tự hợp lý của không gian tìm kiếm để nhanh chóng đạt được kết quả tốt.

1.2.2. Tìm kiếm tối ưu

Tìm kiếm tối ưu (Best-First Search - BeFS) kết hợp ưu điểm của tìm kiếm theo chiều sâu và chiều rộng. Phương pháp này cho phép ta đi theo một con đường duy nhất tại mỗi thời điểm, nhưng vẫn xem xét các hướng khác. Nếu con đường đang đi không hiệu quả, ta sẽ chuyển sang một con đường khác có triển vọng hơn. BeFS chọn trạng thái có khả năng cao nhất từ các trạng thái đã xét, khác với tìm kiếm leo đồi chỉ chọn trạng thái tốt nhất từ các trạng thái kế tiếp. Nhờ vậy, BeFS giúp ta tránh bị mắc kẹt trong các nhánh không hiệu quả, và luôn ưu tiên các hướng tìm kiếm khả thi nhất.

Để làm rõ tư tưởng này ta xét đồ thị (hình 1.4). Khởi đầu, chỉ có một nút (trạng thái) A nên nó sẽ được mở rộng tạo ra 3 nút mới B,C và D. Các con số dưới nút là giá trị cho biết độ tốt của nút. Con số càng nhỏ, nút càng tốt. Do D là nút có khả năng nhất nên nó sẽ được mở rộng tiếp sau nút A và sinh ra 2 nút kế tiếp là E và F. Đến đây, ta lại thấy nút B có vẻ có khả năng nhất (trong các nút B,C,E,F) nên ta sẽ chọn mở rộng nút B và tạo ra 2 nút G và H. Nhưng lại một lần nữa, hai

nút G, H này được đánh giá ít khả năng hơn E, vì thế sự chú ý lại trở về E, E được mở rộng và các nút được sinh ra từ E là I và J. Ở bước kế tiếp, J sẽ được mở rộng vì nó có khả năng nhất. Quá trình này tiếp tục cho đến khi tìm thấy một lời giải.



Hình 1.4 Đồ thị tìm kiếm BeFS

Để cài đặt các thuật giải theo kiểu tìm kiếm BFS, thường cần dùng 2 tập:

- OPEN : tập chứa các trạng thái đã được sinh ra nhưng chưa được xét đến (vì ta đã chọn một trạng thái khác). Thực ra, OPEN là một loại hàng đợi ưu tiên (priority queue) mà trong đó, phần tử có độ ưu tiên cao nhất là phần tử tốt nhất. Người ta thường cài đặt hàng đợi ưu tiên bằng Heap.
- CLOSE : tập chứa các trạng thái đã được xét đến. Chúng ta cần lưu trữ những trạng thái này trong bộ nhớ để đề phòng trường hợp khi một trạng thái mới được tạo ra lại trùng với một trạng thái mà ta đã xét đến trước đó. Trong trường hợp không gian tìm kiếm có dạng cây thì không cần dùng tập này.

Thuật giải tìm kiếm tối ưu

1. Đặt OPEN chứa trạng thái khởi đầu T_o.
2. Cho đến khi tìm được trạng thái đích hoặc không còn nút nào trong OPEN, thực hiện:

a. Chọn trạng thái tốt nhất (Tmax) trong OPN (và xóa Tmax khỏi OPEN)

b. Nếu Tmax là trạng thái kết thúc thì thoát.

c. Ngược lại, tạo ra các trạng thái kế tiếp Tk có thể có từ trạng thái Tmax.

Đối với mỗi trạng thái kế tiếp Tk thực hiện:

+ Tính f(Tk)

+ Thêm Tk vào OPEN

BeFS khá đơn giản. Tuy vậy, trên thực tế, cũng như BFS và DFS, hiếm khi ta dùng BeFS một cách trực tiếp. Thông thường, ta thường dùng các phiên bản của BeFS là A^T , A^{KT} và A^* .

Thông tin về quá khứ và tương lai:

Thông thường, trong các phương án tìm kiếm theo kiểu BeFS, chi phí f của một trạng thái được tính dựa theo hai giá trị mà ta gọi là là g và h. Trong đó h, như đã biết,đó là một ước lượng về chi phí từ trạng thái hiện hành cho đến trạng thái đích (thông tin tương lai), còn g là chiều dài quãng đường đã đi t trạng thái ban đầu cho đến trạng thái hiện tại (thông tin quá khứ). Khi đó hàm ước lượng tng chi pho f(n) được tính theo công thức:

$$f(n) = g(n) + h(n)$$

1.2.3. Thuật giải AKT

Thuật giải A^T trong quá trình tìm đường đi chỉ xét đến các đỉnh và giá của chúng. Nghĩa là việc tìm đỉnh triển vọng chỉ phụ thuộc hàm g(n) (thông tin quá khứ). Tuy nhiên thuật giải này không còn phù hợp khi gặp phải những bài toán phức tạp (do phức tạp cấp hàm mũ) do ta phải tháo một lượng nút lớn. Để khắc phục nhược điểm này, người ta sử dụng thêm các thông tin bổ sung xuất phát từ bản thân bài toán để tìm ra các đỉnh có triển vọng, tức là đường đi tối ưu sẽ tập trung xung quanh đường đi tốt nhất nếu sử dụng các thông tin đặc tả về bài toán (thông tin quá tương lai).

Theo thuật giải này, chi phí của đỉnh được xác định:

$$f(n) = g(n) + h(n)$$

Đỉnh n được chọn nếu $f(n) \rightarrow \min$.

Việc xác định hàm ước lượng h(n) được thực hiện dựa theo:
i) Chọn toán tử xây dựng cung sao cho có thể loại bớt các đỉnh không liên quan và tìm ra các đỉnh có triển vọng.
ii) Sử dụng thêm các thông tin bổ sung nhằm xây dựng tập OPEN và cách lấy các đỉnh trong tập OPEN.
Để làm được việc này người ta phải đưa ra độ đo, tiêu chuẩn để tìm ra các đỉnh có triển vọng. Các hàm sử dụng các kỹ thuật này gọi là hàm đánh giá. Sau đây là một số phương pháp xây dựng hàm đánh giá:
i) Dựa vào xác suất của đỉnh trên đường đi tối ưu.
ii) Dựa vào khoảng cách, xác suất khác của trạng thái đang xét với trạng thái đích hoặc các chỉ tiêu cụ thể của trạng thái đích.
Thuật giải A^{KT}
Vào:
-Đồ thị G = (V, E) trong đó V là tập đỉnh, E là tập cung.
- f: V → $\mathbb{R}^+$ (f(n): hàm ước lượng)
- Đỉnh đầu T_0 và tập các đỉnh đích
Ra:
- Đường đi p: T_0 → T_G ∈ Goal
Phương pháp: Sử dụng 2 danh sách CLOSE và OPEN

```
void AKT()
{
        OPEN = {T0}, g(T0) = 0
        Tính h(T0), f(T0) = g(T0) + h(T0)
        while OPEN ≠ ∅ do
        {
        n ← getNew(OPEN)        // lấy đỉnh n sao cho f(n) → min
        if(n = T_G) then return True
        else
        {
        for each m ∈ A(n) do
```

```
        {
            g(m) = g(n) + cost(m,n)
            Tính h(m), f(m) = g(m) + h(m)
            OPEN = OPEN ∪ {m}
        }
        }
        }
        return False;
}
            {
                for each m   A(n) do
                {
                    g(m) = g(n) + cost(m,n)
                    Tính h(m), f(m) = g(m) + h(m)
                    OPEN = OPEN     {m}
                }
            }
            return False }
```

1.2.4. Thuật giải AT

Thuật giải A^T là một phương pháp tìm kiếm theo kiểu BeFS với chi phí của đỉnh là giá trị hàm g (tổng chiều dài thực sự của đường đi từ đỉnh bắt đầu đến đỉnh hiện tại).

Cho đồ thị G = (V, E) với V: tập đỉnh; E: Tập cung. Với mỗi một cung người ta gắn thêm một đại lượng được gọi là giá của cung.

$C : E \rightarrow R_+$

$e \longmapsto C(e)$

Khi đó đường đi p = n1, n2, …nk có giá được tính theo công thức:

$C(p) = \sum_{i=1}^{k-1} C(n_i, n_{i+1})$

Vấn đề đặt ra là tìm đường đi p từ T_0 đến đỉnh $T_G \in$ Goal sao cho c(p) $\rightarrow$min

Vào: - Đồ thị G = (V, E)

C: E $\rightarrow R^+$

e $\longmapsto$ C(e)

- Đỉnh đầu T_0 và Goal chứa tập các đỉnh đích

Ra: Đường đi p: $T_0 \rightarrow T_G \in$ Goal sao cho:

$C(p) = g(n_k) = \min \{g(n)/n \in \text{Goal}\}$.

Phương pháp : Sử dụng hai danh sách CLOSE và OPEN

```
void AT()
{
    OPEN = {T0}, g(T0) = 0, CLOSE = Ø
    while OPEN ≠ Ø do
    {
        n ← getNew(OPEN)      // lấy đỉnh n sao cho g(n) → min
        if (n = TG) then return True
        else
        {
            for each m ∈ A(n) do
            if (m ∉ OPEN) and (m ∉ CLOSE) then
            {
                g(m) = g(n) + cost(m, n)
                OPEN = OPEN ∪ {m}
            }
            else g(m) = min{g(m), gnew(m)}
            CLOSE = CLOSE ∪ {n}
        }
        return False;
    }
```

}

1.2.5. Thuật giải A*

A* là một phiên bản đặc biệt của KT áp dụng cho trường hợp đồ thị. Thuật giải A* có sử dụng tập hợp CLOSE để lưu trữ những trường hợp đã được xét đến. A* mở rộng AKT bằng cách bổ sung cách giải quyết trường hợp khi mở một nút mà nút này đã có sẵn trong OPEN hoặc CLOSE. Khi xét đến một trạng thái Ti, bên cạnh việc lưu trữ 3 giá trị cơ bản g, h, f để phản ánh chi phí của trạng thái đó, A* còn lưu trữ thêm hai thông số sau:

i) Trạng thái cha của trạng thái Ti (ký hiệu là Cha(Ti) : cho biết trạngthái dẫn đến trạng thái Ti. Trong trường hợp có nhiều trạng thái dẫn đến Ti thì chọn cha(Ti)sao cho chi phí đi từ trạng thái khởi đầu đến Ti là thấp nhất, nghĩa là : g(Ti) = g(Tcha) + cost(Tcha, Ti) là thấp nhất

ii) Danh sách các trạng thái kế tiếp của Ti : danh sách này lưu trữ các trạng thái kế tiếp Tk của Ti sao cho chi phí đến Tk thông qua Ti t trạng thái ban đầu là thấp nhất.

CHƯƠNG 2: THỰC HIỆN BÀI TOÁN TRÒ CHƠI 8 SỐ

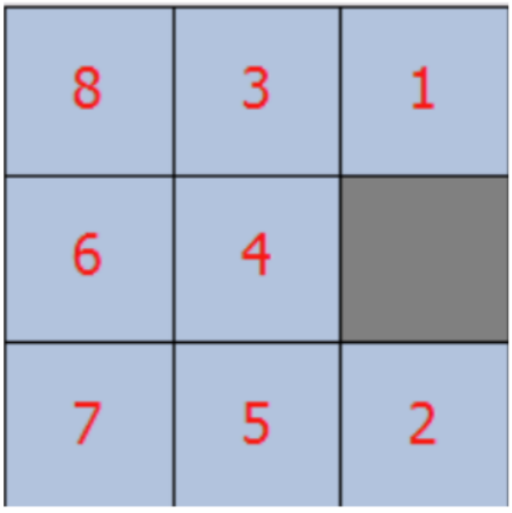
2.1. Bài toán trò chơi 8 số

2.1.1. Giới thiệu bài toán

Trò chơi 8 số (8-puzzle) là một trò chơi cổ điển trong lĩnh vực trí tuệ nhân tạo (AI). Trò chơi này được sử dụng phổ biến để minh họa cho các thuật toán tìm kiếm và phương pháp giải quyết vấn đề trong AI. Việc áp dụng các thuật toán Heuristic giúp cải thiện hiệu suất và tốc độ tìm kiếm lời giải.

Mục tiêu của bài toán là sắp xếp lại các ô số từ trạng thái ban đầu sao cho chúng về đúng vị trí trong trạng thái đích, thông qua việc di chuyển ô trống (thường ký hiệu là 0).

Bài toán có nhiều phiên bản khác nhau dựa theo số ô, như 8-puzzle, 15-puzzle,… ở mức độ đơn giản nhất, chúng em xem xét dạng bài toán 8-puzzle. Bài toán gồm một bảng ô vuông kích thước 3x3, có tám ô được đánh số từ 1 tới 8 và một ô trống. Trạng thái ban đầu, các ô được sắp xếp một cách ngẫu nhiên, nhiệm vụ của người chơi là tìm cách đưa chúng về đúng thứ tự theo như yêu cầu. Ví dụ đưa về giống như hình dưới:



Hình 2.1 Ví dụ cho trạng thái đích của bài toán

Trong quá trình giải bài toàn, tại mỗi bước, ta giả định chỉ có ô trống là di chuyển, như vậy, tối đa ô trống có thể có 4 khả năng di chuyển (lên trên, xuống dưới, sang trái, sang phải).

2.1.2. Không gian trạng thái của bài toán

Trong bảng ô vuông 3 hàng, 3 cột , mỗi ô chứa một số nằm trong phạm vi từ 1 đến 8 sao cho không có 2 ô có cùng giá trị, có một ô trong bảng bị trống (không chứa giá trị nào cả. Xuất phát từ một sắp xếp nào đó các số trong bảng, hãy dịch chuyển ô trống sang phải, sang trái, lên trên hoặc xuống dưới (nếu có thể được) để đưa về bảng ban đầu về bảng quy ước trước. Chẳng hạn hình 1 dưới đây là bảng xuất phát và hình 2 là bảng mà ta phải thực hiện các bước di chuyển ô trống để đạt được.

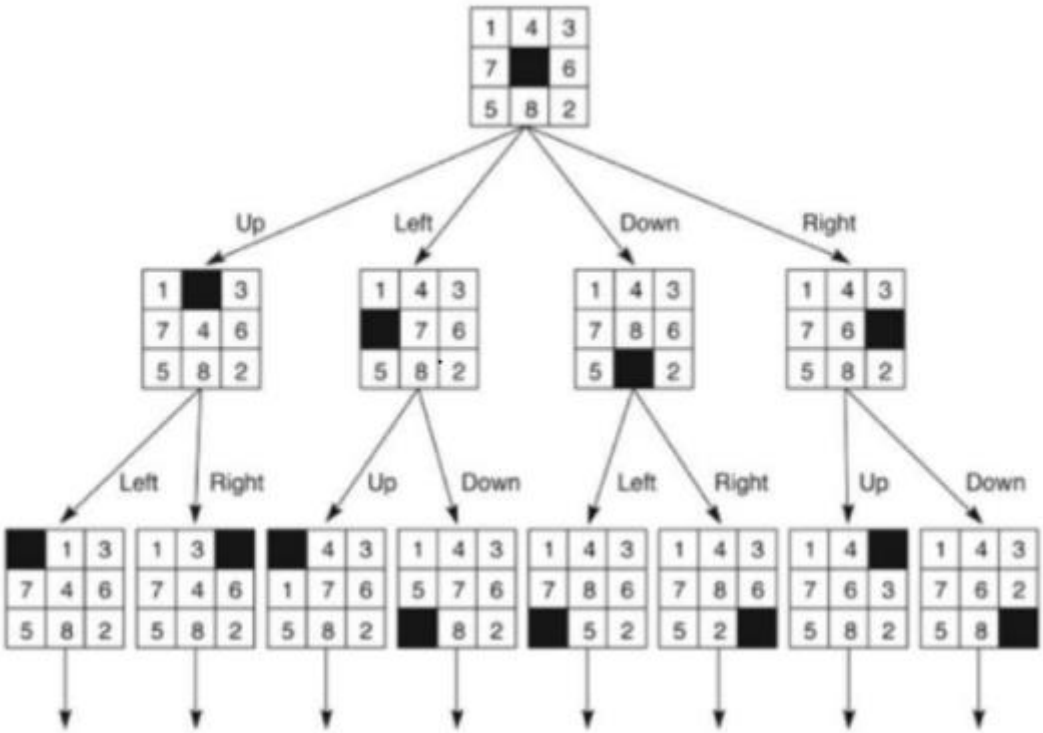
1	4	3
7		6
5	8	2

Hình 2.2 Trạng thái ban đầu

1	2	3
8		4
7	6	5

Hình 2.3 Trạng thái kết thúc

Ta có không gian trạng thái của bài toán được miêu tả như sau:



Hình 2.4 Ví dụ minh họa cho không gian trạng thái của bài toán 8 số

Như vậy, chúng ta nhận thấy rằng, nếu không sử dụng giải thuật một cách hợp lý, tập không gian tìm kiếm sẽ vô cùng lớn. Điều này khiến cho việc đưa khối tiến đến trạng thái đích là vô cùng khó khăn.

2.1.3. Lý do chọn thuật toán A* để code bài toán trò chơi 8 số

A* là thuật toán đặc biệt phù hợp để giải bài toán 8 số nhờ vào nhiều ưu điểm vượt trội. A* đảm bảo tìm được lời giải ngắn nhất nếu sử dụng hàm đánh giá phù hợp, như ước lượng số bước di chuyển còn lại giữa các ô. Thuật toán cũng sẽ luôn tìm ra lời giải nếu có, miễn là hàm đánh giá không đánh giá quá cao so với chi phí thực tế. Không giống như các thuật toán tìm kiếm mù như BFS hay DFS, A* tận dụng hàm f(n)=g(n)+h(n) để chỉ mở rộng những trạng thái tốt nhất, nhờ đó giúp tiết kiệm đáng kể thời gian và bộ nhớ.

Ngoài ra, A* sử dụng cấu trúc OPEN và CLOSED để tránh duyệt lại các trạng thái, giúp tăng tốc độ xử lý và tránh vòng lặp. Đây là điểm mạnh vượt trội khi so với các thuật toán không kiểm soát trạng thái trùng lặp. A* cũng được xem là một phiên bản mở rộng của thuật toán AKT, kết hợp cả chi phí thực tế và ước lượng heuristic, đồng thời có khả năng cập nhật các trạng thái đã có trong OPEN

hoặc CLOSED khi tìm thấy đường đi tốt hơn. Tóm lại, A* là lựa chọn hợp lý và hiệu quả nhất để giải bài toán 8 số.

2.2. Phân tích các thành phần để giải quyết bài toán

2.2.1 Giải thuật sử dụng

A* là giải thuật tìm kiếm trong đồ thị, 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.

Từ trạng thái hiện tại A* xây dựng tất cả các đường đi có thể đi dùng hàm ước lượng khoảng cách (hàm Heuristic) để đánh giá đường đi tốt nhất có thể đi. Tùy theo mỗi dạng bài khác nhau mà hàm Heuristic sẽ được đánh giá khác nhau. A* luôn tìm được đường đi ngắn nhất nếu tồn tại đường đi như thế.

A* lưu giữ một tập các đường đi qua đồ thị, từ đỉnh bắt đầu đến đỉnh kết thúc, tập các đỉnh có thể đi tiếp được lưu trong tập Open.

Thứ tự ưu tiên cho một đường đi được quyết định bởi hàm Heuristic được đánh giá $f(x) = g(x) + h(x)$

- $g(x)$ là chi chi phí của đường đi từ điểm xuất phát cho đến thời điểm hiện tại.
- $h(x)$ là hàm ước lượng chi phí từ đỉnh hiện tại đến đỉnh đích $f(x)$ thường có giá trị càng thấp thì độ ưu tiên càng cao

2.2.2. Cài đặt chi tiết

a) Cài đặt lớp State

```
import copy
class State:
    def __init__(self, data = None, par = None,
                 g = 0, h = 0, op = None):
        self.data = data
        self.par = par
        self.g = g
        self.h = h
        self.op = op
    def clone(self):
        sn = copy.deepcopy(self)
        return sn
    def Print(self):
        sz = 3
        for i in range(sz):
            for j in range(sz):
                print(self.data[i*sz + j], end = ' ')
            print()
        print()
```

Hình 2.5 Code lớp State

```
    def Key(self):
        if self.data == None:
            return None
        res = ''
        for x in self.data:
            res += (str)(x)
        return res
    def __lt__(self, other):
        if other == None:
            return False
        return self.g + self.h < other.g + other.h
    def __eq__(self, other):
        if other == None:
            return False
        return self.Key() == other.Key()
```

Hình 2.6 Code lớp State (tiếp)

Lớp State (Class State) này định nghĩa cấu trúc của một trạng thái trong bài toán 8 số, trong đó:

- Khởi tạo hàm __init__ với:
  + data: Một list biểu diễn trạng thái hiện tại của bài toán (ví dụ: [1, 2, 3, 8, 0, 4, 7, 6, 5]), 0 biểu thị ô trống.
  + par: Tham chiếu đến trạng thái cha (trạng thái dẫn đến trạng thái hiện tại). Dùng để dựng lại đường đi sau này.
  + g: Chi phí thực tế từ trạng thái bắt đầu đến trạng thái hiện tại.
  + h: Giá trị heuristic (ước lượng chi phí từ trạng thái hiện tại đến trạng thái đích).
  + op: Toán tử (Operator) đã được áp dụng để đạt được trạng thái này.
- Khởi tạo hàm clone(self) để tạo một bản sao sâu (deep copy) của trạng thái hiện tại tránh việc thay đổi trạng thái gốc khi tạo các trạng thái con.

- Khởi tạo hàm Print(self) in ra trạng thái của bảng số 3x3 một cách dễ đọc, mỗi hàng được lấy từ danh sách data.

- Khởi tạo hàm Key(self) để chuyển trạng thái thành chuỗi giúp dễ dàng kiểm tra trùng lặp khi tìm kiếm.

- Khởi tạo hàm __lt__ định nghĩa toán tử "nhỏ hơn" (<) để so sánh các trạng thái. Trạng thái có g + h (tổng chi phí + heuristic) nhỏ hơn sẽ có ưu tiên cao hơn.

- Khởi tạo hàm __eq__ để kiểm tra hai trạng thái có giống nhau không (dựa vào chuỗi Key).

b) Cài đặt lớp Operator

```
class Operator:
    def __init__(self, i):
        self.i = i

    def checkStateNull(self, s):
        return s.data is None

    def findPos(self, s):
        sz = 3
        for i in range(sz):
            for j in range(sz):
                if s.data[i * sz + j] == 0:
                    return i, j
        return None

    def swap(self, s, x, y, i):
        sz = 3
        sn = s.clone()
        x_new, y_new = x, y
        if i == 0: x_new += 1
        if i == 1: x_new -= 1
        if i == 2: y_new += 1
        if i == 3: y_new -= 1
        sn.data[x * sz + y], sn.data[x_new * sz + y_new] = sn.data[x_new * sz + y_new], 0
        return sn
```

Hình 2.7 Code lớp Operator

```
    def Down(self, s):
        if self.checkStateNull(s): return None
        x, y = self.findPos(s)
        if x == 2: return None
        return self.swap(s, x, y, self.i)

    def Up(self, s):
        if self.checkStateNull(s): return None
        x, y = self.findPos(s)
        if x == 0: return None
        return self.swap(s, x, y, self.i)

    def Right(self, s):
        if self.checkStateNull(s): return None
        x, y = self.findPos(s)
        if y == 2: return None
        return self.swap(s, x, y, self.i)

    def Left(self, s):
        if self.checkStateNull(s): return None
        x, y = self.findPos(s)
        if y == 0: return None
        return self.swap(s, x, y, self.i)

    def Move(self, s):
        if self.i == 0: return self.Down(s)
        if self.i == 1: return self.Up(s)
        if self.i == 2: return self.Right(s)
        if self.i == 3: return self.Left(s)
        return None
```

Hình 2.8 Code lớp Operator (tiếp)

Lớp Operator (class Operator) này định nghĩa các hành động có thể thực hiện trên bàn cờ 8-puzzle (di chuyển ô trống), trong đó

- Khởi tạo hàm __init__(self, i): Biến i là chỉ số quy định hướng di chuyển của ô trống

(0: xuống, 1: lên, 2: phải, 3: trái).

- Khởi tạo hàm checkStateNull(self, s): kiểm tra xem trạng thái s có hợp lệ không. Nếu s.data == None, trả về True.

- Khởi tạo hàm findPos(self, s) duyệt qua mảng data để tìm vị trí của số 0 (ô trống) , trả về tọa độ (x, y).
- Khởi tạo hàm swap(self, s, x, y, i) : hàm hoán đổi ô trống với các bước như sau:
  + Tạo một bản sao trạng thái s để tránh sửa trực tiếp trạng thái gốc.
  + Xác định vị trí mới của ô trống dựa trên hướng di chuyển.
  + Thực hiện hoán đổi vị trí ô trống với ô cần di chuyển.
- Khởi tạo hàm Up(self, s) : Nếu trạng thái hợp lệ và ô trống có thể di chuyển lên trên, thì thực hiện hoán đổi.
- Khởi tạo hàm Down(self, s), Left(self, s), Right(self, s): Các phương thức Down, Left, Right hoạt động tương tự, kiểm tra xem ô trống có thể di chuyển theo hướng yêu cầu hay không trước khi hoán đổi.
- Khởi tạo hàm Move(self, s): Dựa vào giá trị i, quyết định thao tác di chuyển.

c) Cài đặt các hàm hỗ trợ

```
def checkPriority(Open, tmp):
  if tmp == None:
    return False
  return (tmp in Open.queue)
def equal(O, G):
  if O == None:
    return False
  return O.Key() == G.Key()
def Path(O):
  if O.par != None:
    Path(O.par)
    print(O.op.i)
  O.Print()
def Hx(S, G):
  sz = 3
  res = 0
  for i in range(sz):
    for j in range(sz):
      if S.data[i*sz + j] != G.data[i*sz + j]:
        res += 1
  return res
```

Hình 2.9 Code các hàm hỗ trợ

- Khởi tạo hàm checkPriority(Open, tmp): Kiểm tra trạng thái có trong hàng đợi với

    + Kiểm tra xem trạng thái tmp có nằm trong hàng đợi Open hay không.
    + Open.queue là danh sách các trạng thái đang được xem xét bởi thuật toán A*.

- Khởi tạo hàm equal(O, G): So sánh hai trạng thái O và G:

    + Kiểm tra xem trạng thái O có giống trạng thái mục tiêu G không.
    + Sử dụng phương thức Key() của lớp State để so sánh.

- Khởi tạo hàm Path(O) để truy vết đường đi sau khi thực hiện thành công bài toán:

    + Đệ quy để in ra các bước đi từ trạng thái ban đầu đến trạng thái O.
    + Nếu O có trạng thái cha (par), tiếp tục truy vết về trạng thái gốc.
    + Sau cùng, in trạng thái O.

- Khởi tạo hàm Hx(S, G): đây là hàm heuristic (ước lượng khoảng cách đến trạng thái mục tiêu).

    + Đếm số lượng ô sai vị trí giữa trạng thái hiện tại S và trạng thái mục tiêu G.

d) Cài đặt hàm Run (thực hiện thuật toán A*)

```
from queue import PriorityQueue
def RUN():
  Open = PriorityQueue()
  Closed = PriorityQueue()
  S.g = 0
  S.h = Hx(S, G)
  Open.put(S)
  while True:
    if Open.empty() == True:
      print('Tim kiem that bai')
      return
    O = Open.get()
    Closed.put(O)
    if equal(O, G) == True:
      print('Tim kiem thanh cong')
      Path(O)
      return
    for i in range(4):
      op = Operator(i)
      child = op.Move(O)
      if child == None:
        continue
      ok1 = checkPriority(Open, child)
      ok2 = checkPriority(Closed, child)
      if ok1 == False and ok2 == False:
        child.par = O
        child.op = op
        child.g = O.g + 1
        child.h = Hx(child, G)
        Open.put(child)
```

Hình 2.10 Code hàm Run

* Khởi tạo hàm RUN() để triển khai thuật toán tìm kiếm A* để giải bài toán 8 số. Đây là một thuật toán tìm kiếm đồ thị dùng hàm đánh giá heuristic để tìm đường đi tối ưu từ trạng thái bắt đầu đến trạng thái đích.

- Khởi tạo hàng đợi ưu tiên
  + Open: Hàng đợi ưu tiên chứa các trạng thái đang chờ xử lý.
  + Closed: Hàng đợi ưu tiên chứa các trạng thái đã được xét duyệt.
- Thiết lập trạng thái ban đầu
  + S.g = 0: Chi phí từ trạng thái ban đầu đến S là 0.

+ S.h = Hx(S, G): Tính giá trị heuristic h (ước lượng khoảng cách đến trạng thái mục tiêu).
+ Open.put(S): Đưa trạng thái ban đầu vào hàng đợi.

- Khởi tạo vòng lặp tìm kiếm
  + Nếu Open trống, tức là không còn trạng thái nào để mở rộng → tìm kiếm thất bại.
  + Lấy trạng thái tốt nhất từ Open (trạng thái có g + h nhỏ nhất).
  + Đưa vào Closed để đánh dấu đã xét duyệt.
  + Nếu O là trạng thái mục tiêu G → tìm kiếm thành công.
  + In ra đường đi từ trạng thái ban đầu đến trạng thái mục tiêu (Path(O)).
- Sinh trạng thái con:
  + Sinh trạng thái mới bằng cách di chuyển ô trống theo 4 hướng (Up, Down, Left, Right).
  + Nếu trạng thái child không hợp lệ → bỏ qua.
  + Kiểm tra xem child đã có trong hàng đợi Open hoặc Closed chưa.
  + Nếu trạng thái con chưa được xét duyệt, tiếp tục xử lý.
  + Lưu cha (par) để có thể truy vết đường đi.
  + Lưu hành động (op) để biết bước di chuyển nào đã được thực hiện.
  + Cập nhật chi phí (g): cộng thêm 1 vì mỗi nước đi đều có cùng chi phí.
  + Tính toán h bằng hàm heuristic Hx(child, G).
  + Đưa trạng thái con vào hàng đợi Open để tiếp tục mở rộng.

e) Cài đặt hàm nhập trạng thái ban đầu, trạng thái đích và chạy toàn bộ code với trạng thái ban đầu nhập từ bàn phím

```
def init(custom_data):
    if not custom_data or len(custom_data) != 9:
        raise ValueError("Dữ liệu phải có đúng 9 phần tử.")

    G = State(data=[1, 2, 3, 4, 5, 6, 7, 0, 8])
    S = State(data=custom_data)

    return S, G
```

```
custom_start = [1, 2, 3, 7, 4, 6, 5, 0, 8]
S, G = init(custom_start)
RUN(S, G)
```

Hình 2.11 Code hàm nhập trạng thái ban đầu, trạng thái đích và chạy toàn bộ code

- Khởi tạo hàm init(num): khởi tạo trạng thái ban đầu (S) và trạng thái mục tiêu (G):

- Hàm nhận một danh sách custom_data, đại diện cho trạng thái ban đầu. Nếu danh sách rỗng (not custom_data) hoặc không có đúng 9 phần tử, nó sẽ báo lỗi.
- G là trạng thái mục tiêu mà chương trình cần đạt được, trạng thái này có thể thay đổi tùy theo yêu cầu.
- S được tạo từ dữ liệu đầu vào mà người dùng cung cấp. Điều này giúp chương trình có thể bắt đầu từ bất kỳ trạng thái nào, không bị cố định.

- Thực thi thuật toán A*:

+ custom_start: danh sách biểu diễn trạng thái ban đầu.
+ Khởi tạo trạng thái ban đầu S, trạng thái đích G với dữ liệu từ custom_start.
+ Chạy thuật toán A* để tìm đường đi từ S đến G*.

```
Tìm kiếm thành công
1 2 3
7 4 6
5 0 8

3
1 2 3
7 4 6
0 5 8

1
1 2 3
0 4 6
7 5 8

2
1 2 3
4 0 6
7 5 8

0
1 2 3
4 5 6
7 0 8
```

Hình 2.12 Kết quả sau khi chạy code

KẾT LUẬN

Báo cáo đã hoàn thành việc nghiên cứu về lý thuyết không gian trạng thái, các thuật toán tìm kiếm Heuristic, và ứng dụng chúng vào giải quyết bài toán trò chơi 8 số kinh điển.

Trong Chương 1, chúng em đã trình bày chi tiết về khái niệm không gian trạng thái, các thành phần cơ bản như mô tả trạng thái và toán tử, cùng với cấu trúc chung của bài toán tìm kiếm trong không gian này. Đặc biệt, báo cáo đã đi sâu phân tích tổng quan về các thuật toán tìm kiếm Heuristic, làm rõ khái niệm, vai trò, ưu điểm và phương pháp xây dựng hàm Heuristic. Ba thuật toán quan trọng là AKT, AT và A* đã được nghiên cứu kỹ lưỡng, cung cấp nền tảng lý thuyết vững chắc cho việc lựa chọn giải pháp.

Chương 2 tập trung vào việc áp dụng lý thuyết vào thực tiễn thông qua bài toán trò chơi 8 số. Chúng em đã giới thiệu bài toán, mô tả không gian trạng thái đặc trưng của nó và lý giải chi tiết lý do lựa chọn thuật toán A* để cài đặt. Thuật toán A* được chọn nhờ khả năng tìm kiếm tối ưu với chi phí thấp hơn so với các thuật toán tìm kiếm mù, và hiệu quả trong việc tìm đường đi ngắn nhất trong không gian trạng thái rộng lớn của bài toán 8 số. Phần cài đặt chi tiết đã được trình bày, bao gồm các thành phần cốt lõi và cách thức triển khai giải thuật A* để tìm lời giải cho bài toán.

Thông qua quá trình nghiên cứu và thực hiện, báo cáo đã đạt được những kết quả chính sau:

- Nắm vững lý thuyết: Hiểu rõ các khái niệm về không gian trạng thái và vai trò của tìm kiếm Heuristic trong trí tuệ nhân tạo.
- Phân tích thuật toán: Đánh giá được ưu nhược điểm và khả năng ứng dụng của các thuật toán tìm kiếm Heuristic, đặc biệt là A*.
- Ứng dụng thực tế: Xây dựng thành công chương trình giải bài toán trò chơi 8 số bằng thuật toán A*, minh chứng cho tính hiệu quả của phương pháp này trong các bài toán tìm kiếm đường đi tối ưu.

.

TÀI LIỆU THAM KHẢO

[1] Giáo trình Trí tuệ nhân tạo trường Đại học Công Nghiệp Hà Nội

[2] https://websitehcm.com/cac-thuat-toan-informed-search-algorithms/

[3] https://www.iostream.vn/article/thuat-giai-a-DVnHj

[4] https://voer.edu.vn/m/thuat-toan-tim-kiem-heuristic/90488a73

[5]https://cuuduongthancong.com/atc/1153/cac-phuong-phap-tim-kiem-heuristic