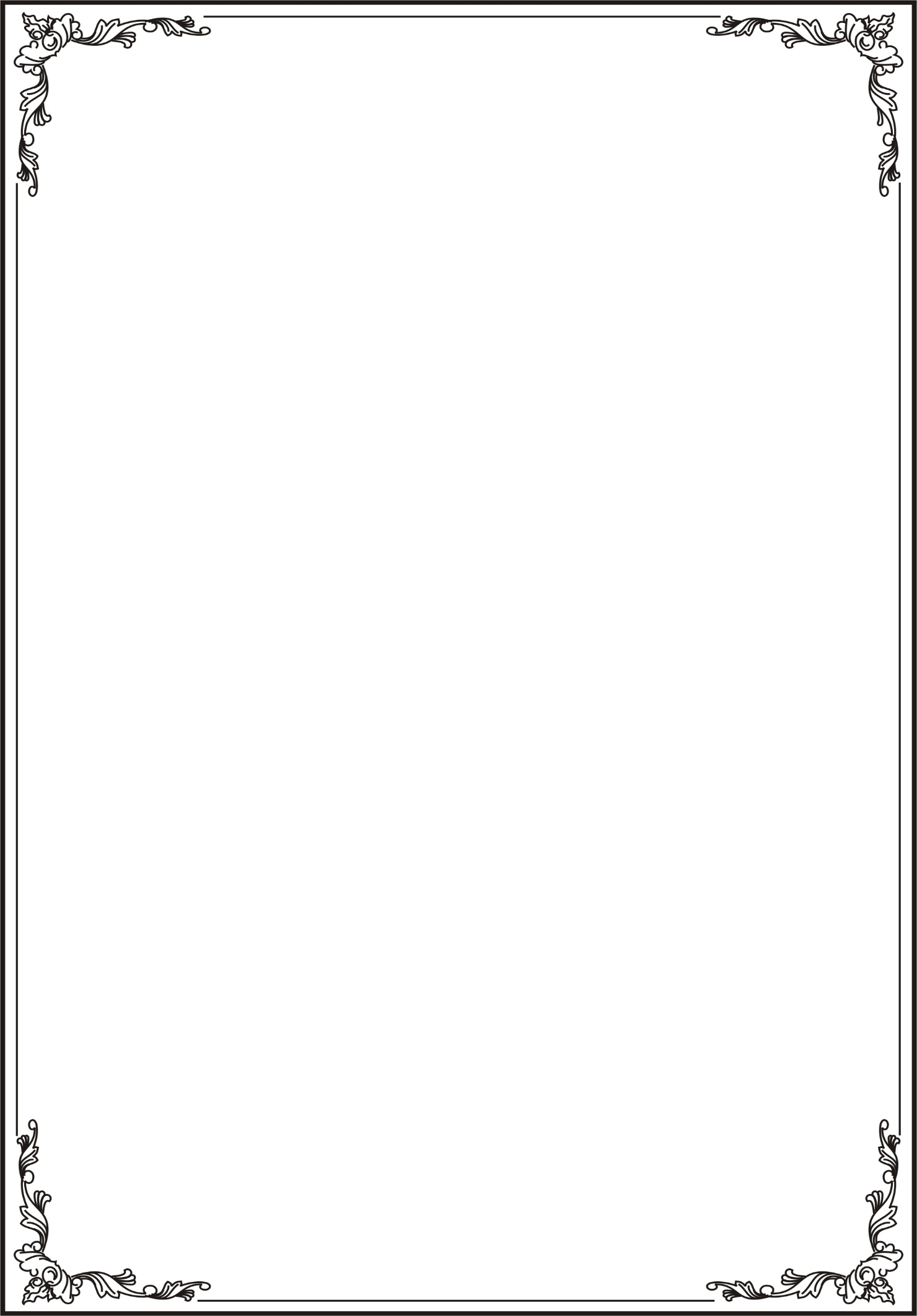
**­­ĐẠI HỌC QUỐC GIA HÀ NỘI**

**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN**

**KHOA TOÁN - CƠ - TIN HỌC**

--🙢🕮🙠--



BÁO CÁO

**LẬP TRÌNH NÂNG CAO**

**Đề tài:**

**Sử dụng thuật toán A\* giải bài toán n-puzzle**

**Giảng viên : PGS.TS Lê Trọng Vĩnh**

**Thành viên trong nhóm : Đặng Hữu Tấn - 20001972**

**Nguyễn Thị Phương Ngân - 20001953**

**Nguyễn Trần Đức Thắng - 20001975**

Hà Nội - 2022

**MỤC LỤC**

[**Lời nói đầu 3**](#_Toc121259404)

[**I. Bài toán N-puzzle 4**](#_Toc121259405)

[**II. Thuật toán A\* 5**](#_Toc121259406)

[**1. Diễn giải thuật toán 5**](#_Toc121259407)

[**2. Tối ưu không gian tìm kiếm 5**](#_Toc121259408)

[**3. Bài toán không có lời giải 6**](#_Toc121259409)

[**III. Cài đặt thuật toán A\* 6**](#_Toc121259410)

[**IV. Trò chơi 10**](#_Toc121259411)

[**V. Kết luận 14**](#_Toc121259412)

[**Tài liệu tham khảo 15**](#_Toc121259413)

Lời nói đầu

Trong ngành khoa học máy tính, một giải thuật tìm kiếm là một thuật toán lấy đầu vào là một bài toán và trả về kết quả là một lời giải cho bài toán đó, thường là sau khi cân nhắc giữa một loạt các lời giải có thể. Tập hợp tất cả các lời giải có thể cho bài toán được gọi là không gian tìm kiếm. Thuật toán tìm kiếm có thông tin sử dụng hàm đánh giá heuristic giúp ta giảm đáng kể thời gian cần thiết cho việc tìm kiếm lời giải.

Trong phạm vi bài báo cáo, chúng em xin trình bày về đề tài "**Sử dụng thuật toán A giải bài toán n-puzzle**". Trong đó thuật toán A sử dụng hàm heuristic là tính khoảng cách Manhattan. Nhóm chúng em đã sử dụng ngôn ngữ Python và các thư viện hỗ trợ, thuật toán A\* vào việc xây dựng trò chơi n-puzzle với kích thước 3\*3.

Trong quá trình thực hiện đề tài không tránh khỏi những sai sót, nhóm chúng em rất mong nhận được sự đánh giá, góp ý của thầy để hoàn thiện thêm về vấn đề này.

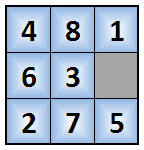
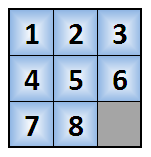
**Chúng em xin chân thành cảm ơn!**

1. Bài toán N-puzzle

Bài toán (hay game) n-puzzle có lẽ rất quen thuộc với chúng ta. Nó được biết đến với nhiều phiên bản và tên gọi khác nhau như 8-puzzle, 15-puzzle, Gem puzzle, Boss puzzle, Game of Fifteen, Mystic Square,... Bài toán n-puzzle là vấn đề cổ điển cho mô hình liên quan tới trí tuệ nhân tạo. Bài toán đặt ra là phải tìm đường đi từ trạng thái hiện tại tới trạng thái đích.

Game n-puzzle là một chương trình trò chơi giải quyết bài toán này. Phần mềm được viết bằng ngôn ngữ Python. Người dùng sử dụng chuột để chơi sắp xếp các con số, hoặc có thể sử dụng chức năng tìm lời giải nhờ thuật toán A\*

Yêu cầu xây dựng một bảng ô vuông gồm n hàng, n cột. Bảng gồm 1 ô trống và n2 – 1 ô chứa các số trong phạm vi [1, n2 - 1]. Xuất phát từ một cách xếp bất kì, di chuyển ô trống lên trên, xuống dưới, sang phải, sang trái để đưa các ô về trạng thái đích. Dưới đây là ví dụ về 8-puzzle:



Trạng thái đầu Trạng thái đích

Các bước di chuyển của ô trống:

|  |  |  |
| --- | --- | --- |
| 1 | 3 | 2 |
| 5 |  | 4 |
| 7 | 6 | 8 |

|  |  |  |
| --- | --- | --- |
| 1 | 3 | 2 |
| 5 | 6 | 4 |
| 7 |  | 8 |

|  |  |  |
| --- | --- | --- |
| 1 |  | 2 |
| 5 | 3 | 4 |
| 7 | 6 | 8 |

|  |  |  |
| --- | --- | --- |
| 1 | 3 | 2 |
| 5 | 4 |  |
| 7 | 6 | 8 |

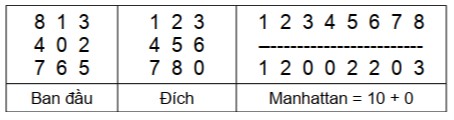
|  |  |  |
| --- | --- | --- |
| 1 | 3 | 2 |
|  | 5 | 4 |
| 7 | 6 | 8 |

1. Thuật toán A\*
2. **Diễn giải thuật toán**

A\* là một giải thuật tìm kiếm thường được sử dụng trong các vấn đề liên quan đến đồ thị và tìm đường đi, được sử dụng rộng rãi trong ngành trí tuệ nhân tạo. Nó được chọn không chỉ vì tính hiệu quả mà còn vì rất dễ dàng để hiểu và cài đặt. Ta gọi mỗi *nút tìm kiếm* trong bài toán gồm ma trận, số bước đã dịch chuyển để đến ma trận này và nút tìm kiếm trước đó.

Trước tiên, ta thêm nút tìm kiếm đầu tiên (gồm ma trận ban đầu, 0 bước dịch chuyển, nút tìm kiếm trước đó là rỗng) vào một hàng đợi ưu tiên Sau đó ta xóa nút tìm kiếm có độ ưu tiên thấp nhất trong hàng đợi, và thêm vào hàng đợi tất cả các nút tìm kiếm hàng xóm của nó – tức là thêm các nút tìm kiếm có thể đạt tới từ nút hiện tại vói một bước chuyển. Lặp lại thủ tục này cho tới khi nút tìm kiếm được xóa tương thích với nút đích.

Ta sử dụng *hàm ưu tiên* cho mỗi nút tìm kiếm là hàm Manhattan, xác định như sau: hàm Manhattan của một nút tìm kiếm là tổng các khoảng cách Manhattan (tổng các khoảng cách ngang và dọc) từ các ô tới các vị trí đích của nó, cộng với số bước dịch chuyển đã thực hiện để tới nút tìm kiếm hiện tại. Ví dụ: hàm ưu tiên Manhattan của nút tìm kiếm đầu tiên cho giá trị 10:

****

Ta có quan sát sau: để giải bài toán từ một nút tìm kiếm cho trước, tổng số lần dịch chuyển ta cần thực hiện (gồm cả những lần đã dịch chuyển) nhỏ nhất chính là độ ưu tiên của nút. Chú ý rằng ta không xét ô trắng khi tính độ ưu tiên. Do đó, khi nút đích được lấy ra khỏi hàng đợi thì ta sẽ tìm được dãy dịch chuyển từ nút ban đầu tới nút đích và đây cũng chính là dãy có số lần dịch chuyển ít nhất.

1. **Tối ưu không gian tìm kiếm**

Thuật toán tìm kiếm A\* nêu trên có nhược điểm là các nút tìm kiếm ứng với ma trận giống nhau được đưa vào hàng đợi nhiều lần. Để giảm việc tìm kiếm trên nhiều nút vô ích, khi xét các nút hàng xóm, ta khong thêm nút hàng xóm nào có ma trận giống với ma trận của nút tìm kiếm trước đó. VD:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 8 1 3  4 0 2  7 6 5 | 8 1 3  4 2 0  7 6 5 | 8 1 0  4 2 3  7 6 5 | 8 1 3  4 0 2  7 6 5 | 8 1 3  4 2 5  7 6 0 |
| Trước đó | Nút tìm kiếm | Hàng xóm | Không xét | Hàng xóm |

1. **Bài toán không có lời giải**

Không phải ma trận ban đầu nào cũng cho lời giải. Ví dụ:

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 8 | 7 | 0 |

Để phát hiện ma trận ban đầu có giải được hay không, ta sử dụng quan sát sau. Tập các ma trận được chia làm 2 lớp tương ứng có giải được hay không:

1. Tập các ma trận giải được
2. Tập các ma trận không giải được nếu ta sửa đổi ma trận ban đầu bằng cách hoán đổi hai ô trống bất kì trong cùng một hàng,

Từ đó, để phát hiện ma trận có giải được hay không, ta chạy thuật toán A\* đồng thời trên hai ma trận – một với ma trận ban đầu và một với ma trận đã được sửa bằng cách hoán đổi hai ô khác trống trên một hàng bất kì. Chỉ có một và chỉ một ma trận dẫn tới ma trận đích. Nếu ma trận sửa đổi dẫn tới ma trận đích thì ma trận ban đầu là không giải được.

1. Cài đặt thuật toán A\*

***Quy ước:***

* FRINGE : là hàng đợi ưu tiên chứa các node con sinh ra mà chưa được xét đến.
* CLOSED : tập các trạng thái đã xét, các trạng thái con được sinh ra sẽ được kiểm tra,
* nếu đã tồn tại trong CLOSED thì không thêm vào FRINGE nữa.
* RESULT: tập trạng thái từ trạng thái hiện tại cho đến đích.
* CHILD: tập node con của một node bất kỳ.

***Cài đặt***

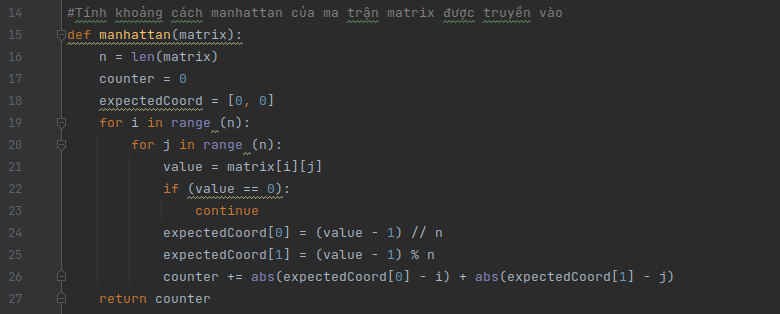
* Bước 1
* RESULT = {}
* startNode.f = startNode.h = đánh giá heuristic từ startNode đến goalNode
* startNode.g = 0
* totalNodes = approvedNodes = 0
* FRINGE = {startNode}
* Bước 2

While(FRINGE != {})

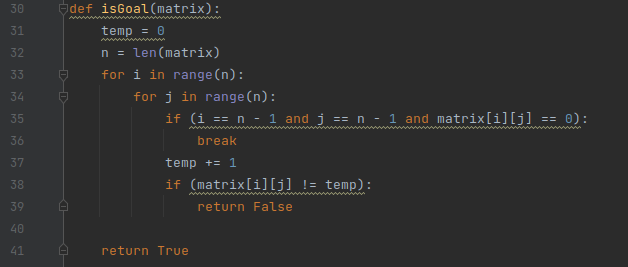
* Nếu quá thời gian tìm kiếm hoặc có yêu cầu dừng
* FRINGE.clear()
* CHILD.clear()
* CLOSED.clear()
* Tìm node có đánh giá f(n) nhỏ nhất trong FRINGE và gán cho currentNode
* Nếu currentNode là goalNode
* totalNodes = approvedNodes + FRINGE.size()
* Thêm kết quả và RESULT
* Tính thời gian tìm kiếm
* FRINGE.clear()
* CHILD.clear()
* CLOSED.clear()
* Dừng thuật toán
* Thiết lập các node con của currentNode vào CHILD
* Loại những node có trạng thái đã xét ra khỏi CHILD
* Đưa các node con vào FRINGE
* Đặt child của các node con là currentNode
* Tính chi phí từ đầu đến node con = currentNode + 1
* Tính hàm ước lượng của các node con đến goalNode
* Tính hàm f = g + h
* CHILD.clear()
* approvedNodes++

Rất dễ thấy là mỗi trạng thái của bảng số là một hoán vị của m x m phần tử (với m là cạnh), như vậy **không gian trạng thái** của nó là (m x m)!, với 8-puzzle là 9! = 362880 (m = 3) và 15-puzzle là 16! = 20922789888000 (m =4). Khi m tăng lên 1 đơn vị thì không gian trạng thái tăng lên rất nhanh, điều này khiến cho việc giải quyết các phiên bản m > 3 ít khi được áp dụng.

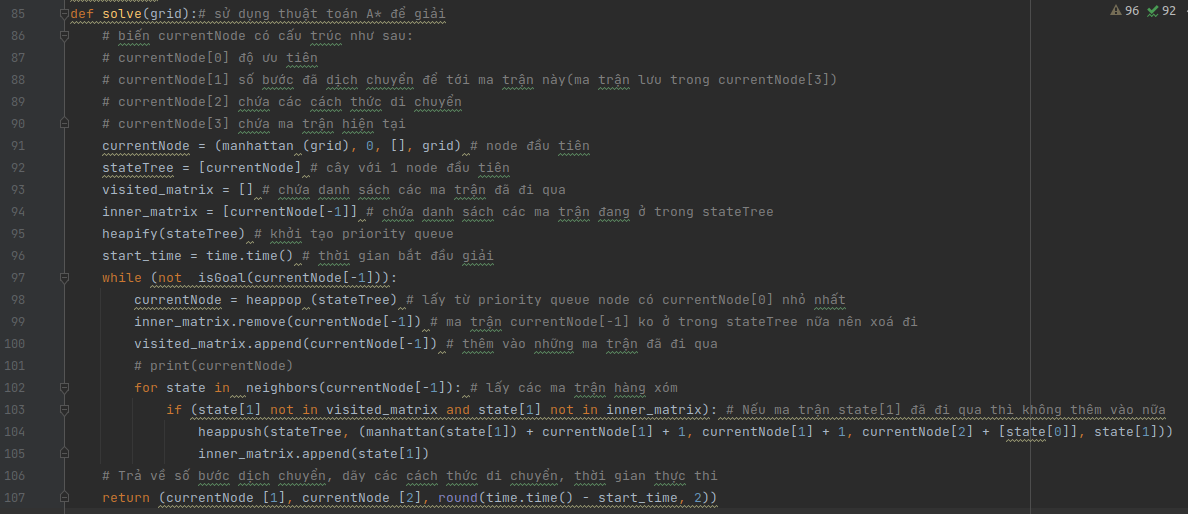
Để áp dụng thuật toán A\* giải bài toán này, cần một hàm heuristic **h** để ước lượng giá trị của mỗi trạng thái của bảng số. Có một số cách có thể đã biết tới như tính dựa vào khoảng cách sai lệch của các ô số với vị trí đúng, hoặc đơn giản là đếm xem có bao nhiêu ô sai vị trí,… Ở đây chọn theo cách thứ nhất, tức là tính tổng số ô sai lệch của các ô số so với vị trí đúng của nó (Manhattan**)**.



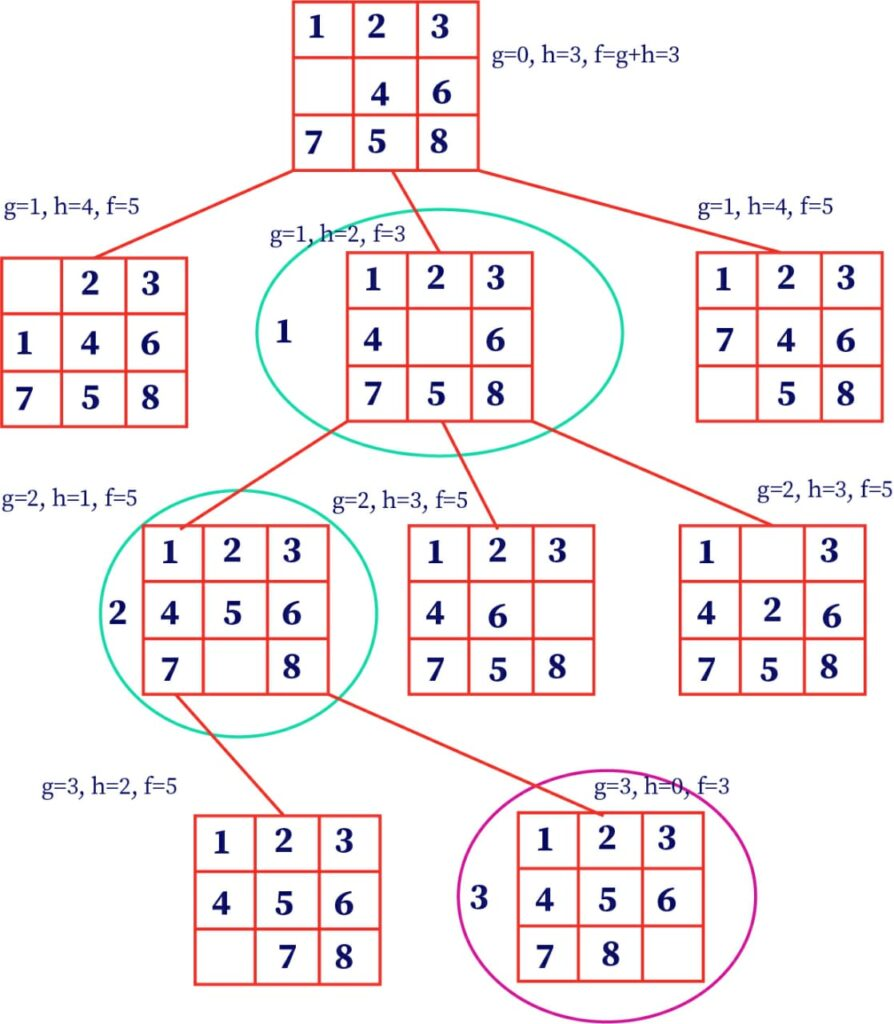
Hàm isGoal để kiểm tra ma trận matrix được truyền vào đã là ma trận đích chưa:



Hàm solve sử dụng thuật toán A\* để giải:



Minh họa A\*:



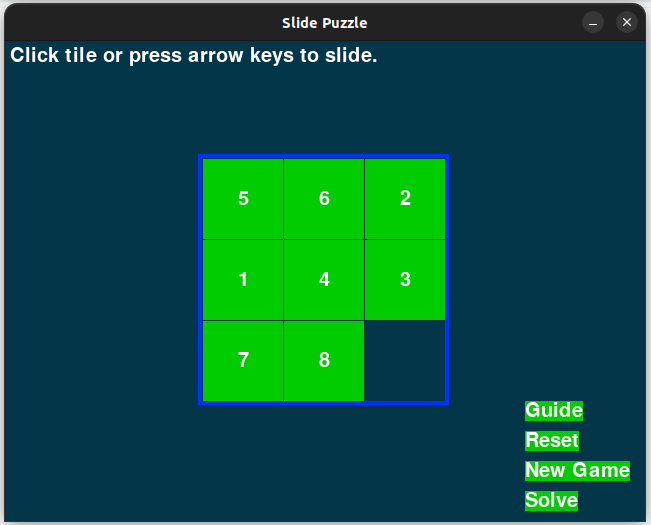
Ngoài ra, chương trình còn có một số hàm khác như guide() để cài đặt giao diện hiển thị kết quả…

1. Trò chơi

Trò chơi 8-puzzle được cài đặt trong chương trình **PuzzleGame.py**. Chương trình có sử dụng các thư viện trong Python như pygame, tkinker,... và chương trình cài đặt thuật toán A\* là **a\_star.py**.

Trò chơi 8-puzzle là một bảng gồm lưới 3x3 với tám ô (được đánh số từ 1 đến 8 từ trái sang phải) và một ô trống. Các ô xếp bắt đầu ở các vị trí ngẫu nhiên và người chơi phải trượt các ô xếp xung quanh cho đến khi các ô xếp trở lại thứ tự ban đầu.

Giao diện trò chơi khi khởi chạy chương trình PuzzleGame.py là:

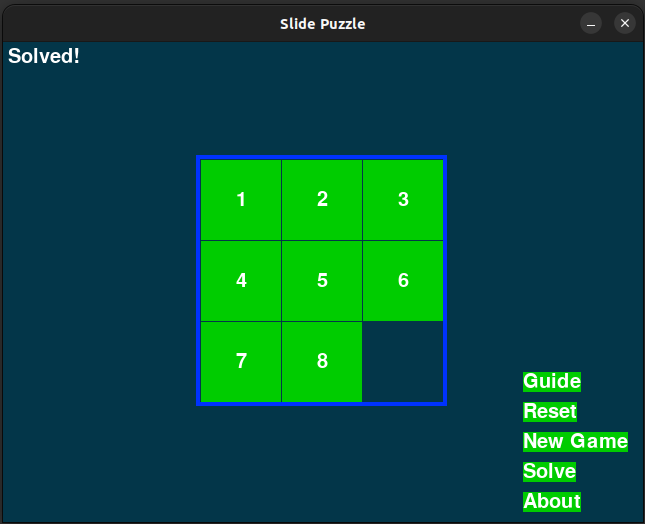


Để chơi trò chơi người dùng có thể sử dụng các phím mũi tên trên bàn phím hoặc click chuột vào các ô để di chuyển ô trống.

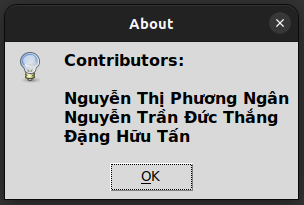
Chức năng của các nút bấm:

* Guide: Hiển thị hướng dẫn trò chơi
* Reset: Cho phép người dùng chơi lại từ đầu
* New Game: Tạo trò chơi mới
* Solve: Giải trò chơi tự động
* About: Thông tin sinh viên phát triển trò chơi

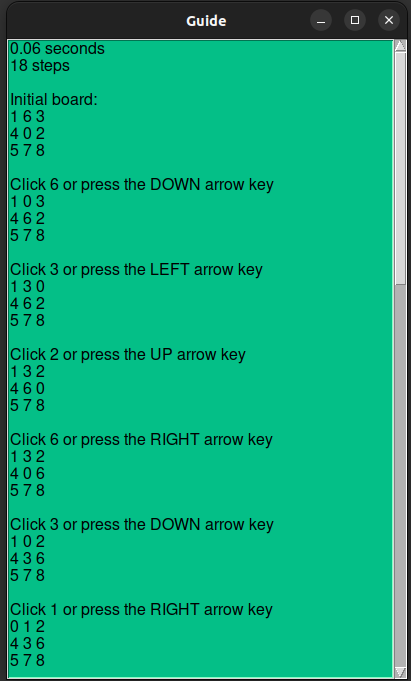
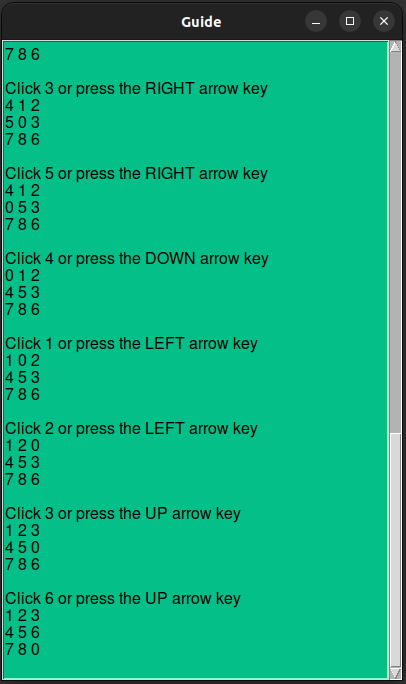
Giao diện đích khi trò chơi đã được giải xong:

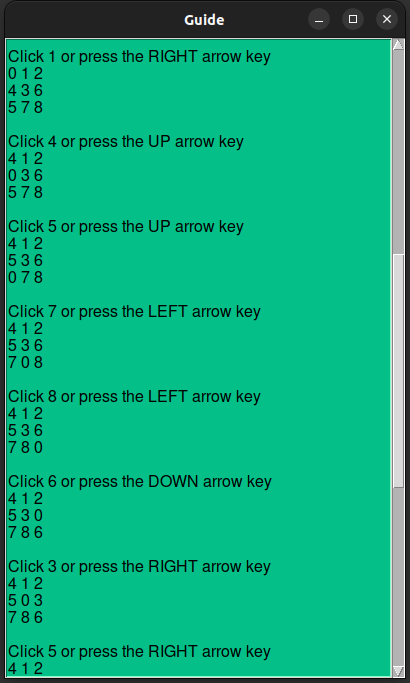


Giao diện About:



Giao diện hướng dẫn trò chơi Guide:





1. Kết luận

Bài báo cáo trên đã làm rõ về bài toán n-puzzle, cách áp dụng thuật toán A\* với sự hỗ trợ của ngôn ngữ Python và các thư viện hỗ trợ để giải bài toán. Chương trình có thể tạm chấp nhận với mức 8-puzzle, tuy nhiên nếu muốn giải được bài toán với n tương đối lớn thì ta cần tìm một giải pháp khác.

Tài liệu tham khảo

[1] Mathew, K., & Tabassum, M. (2014). Experimental comparison of uninformed and heuristic AI algorithms for N Puzzle and 8 Queen puzzle solution. *Int. J. Dig. Inf. Wireless Commun.(IJDIWC)*, *4*(1), 143-154.

[2] Gupta, S., & Bairagi, S. Evaluating Search Algorithms for Solving n-Puzzle.

[3] Kunkle, D. R. (2001). Solving the 8 puzzle in a minimum number of moves: An application of the a\* algorithm. *Introduction to Artificial Intelligence*.

[4] Morazán, M.T. (2022). N-Puzzle Version 6. In: Animated Program Design. Texts in Computer Science. Springer, Cham. https://doi.org/10.1007/978-3-031-04317-8\_12

[5] https://faramira.com/solving-8-puzzle-problem-using-a-star-search/

[6] https://blog.goodaudience.com/solving-8-puzzle-using-a-algorithm-7b509c331288?gi=acc43e0d33d8

[7] https://en.wikipedia.org/wiki/A\*\_search\_algorithm

[8] https://aptech.vn/kien-thuc-tin-hoc/n-puzzle-tim-hieu-ve-cach-giai-bai-toan.html