BỘ CÔNG THƯƠNG

TRƯỜNG ĐẠI HỌC ĐIỆN LỰC

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO CHUYÊN ĐỀ HỌC PHẦN

TRÍ TUỆ NHÂN TẠO

SỬ DỤNG THUẬT TOÁN A* VÀO TRÒ CHƠI 8 CON SỐ

Giảng viên hướng dẫn : **TS.NGUYỄN HÀ NAM**

Sinh viên thực hiện : **NGUYỄN THỊ HẠNH AN**

NGUYỄN THỊ THÙY LINH

Ngành : **CÔNG NGHỆ THÔNG TIN**

Chuyên ngành : **CÔNG NGHỆ PHẦN MỀM**

Lớp học phần : **D17CNPM3**

Khóa : **2022-2027**

PHIẾU CHẤM ĐIỂM

Sinh viên thực hiện:

Họ và tên	Nội dung thực hiện	Chữ ký	Ghi chú
NGUYỄN THỊ THÙY LINH			
NGUYỄN THỊ HẠNH AN			

Giảng viên chấm:

Họ Tên	Chữ ký	Ghi chú

MỤC LỤC

DANH MỤC HÌNH ẢNH

LỜI NÓI ĐẦU

Đề tài kết thúc học phần mà chúng em lựa chọn cho môn Nhập môn trí tuệ nhân tạo là " Áp dụng thuật giải A* vào trò chơi 8 con số". Để hoàn thành báo cáo cho đề tài này, trước hết thì chúng em cần hiểu rõ thuật giải A*, tiếp đến là hiểu rõ quy luật trò chơi, sau đó là áp dụng thuật giải vào giải quyết mục tiêu mà trò chơi yêu cầu và cuối cùng là sử dụng C# để lập trình lên trò chơi 8 con số. Trò chơi 8 con số hay còn gọi là puzzle 8 hẳn đã rất quen thuộc với nhiều người, để giành chiến thắng thì đòi hỏi người chơi phải sử dụng bộ óc linh hoạt của mình, còn về cảm nhận khó hay dễ thì còn phải tùy thuộc vào khả năng của mỗi người. Tuy nhiên, trong lĩnh vực AI thì những dạng trò chơi như thế này sẽ được xử lý rất đơn giản nhờ bàn tay và khối óc của người lập trình viên đã thiết lập vào sản phẩm trí tuệ nhân tạo. Các sản phẩm trí tuệ nhân tạo sẽ còn phổ biến hơn nữa mà thậm chí là thay thế con người nhiều vấn đề trong tương lai. So sánh về tốc độ làm việc thì có thể AI sẽ còn vượt trộ hơn hẳn con người. Thế nhưng để kiến tạo ra lĩnh vực AI còn phải trải qua rất nhiều năm lịch sử đi cùng là bao công sức của các nhà khoa học máy tính-những người tìm ra hoặc đưa các phương pháp vào để phân tích nhiều khía cạnh và xử lý vấn đề. Một trong những phương pháp tạo nền móng nhằm giải quyết các bài toán có thể kể đến như thuật giải A*. Đây là loại thuật giải tìm kiếm có thông tin mà chúng em đã được học trên lớp và nhân cơ hội này, nhóm em muốn sử dụng nó làm đề tài để được hiểu rõ hơn cũng như ứng dụng thực tế trong việc học lập trình.

Trước khi đến nội dung cụ thể, nhóm em mong muốn giành lời cảm ơn tới các thầy cô trong khoa công nghệ thông tin nói chung và thầy Nguyễn Hà Nam nói riêng, vì nếu không có sự hướng dẫn tận tình của thầy cô thì sinh viên khó mà hoàn thành được môn học lần này.

Chúng em xin chân thành cảm ơn !

CHƯƠNG 1: TỔNG QUAN VỀ TRÍ TUỆ NHÂN TẠO

1.1 Khái niệm về trí tuệ nhân tạo

Trí tuệ nhân tạo hay trí thông minh nhân tạo (Artificial intelligence – viết tắt là AI) 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.

Trí tuệ nhân tạo khác với việc lập trình logic trong các ngôn ngữ lập trình là ở việc ứng dụng các hệ thống học máy (machine learning) để mô phỏng trí tuệ của con người trong các xử lý mà con người làm tốt hơn máy tính.

Cụ thể, trí tuệ nhân tạo giúp máy tính có được những trí tuệ của con người như: biết suy nghĩ và lập luận để giải quyết vấn đề, biết giao tiếp do hiểu ngôn ngữ, tiếng nói, biết học và tự thích nghi, …

Tuy rằng trí thông minh nhân tạo có nghĩa rộng như là trí thông minh trong các tác phẩm khoa học viễn tưởng, nó là một trong những ngành trọng yếu của tin học. Trí thông minh nhân tạo liên quan đến cách cư xử, sự học hỏi và khả năng thích ứng thông minh của máy móc.

1.2 Lịch sử phát triển của trí tuệ nhân tạo

Nghiên cứu AI sớm vào những năm thập niên 60 đã khám phá các vấn đề mà công nghệ này có thể giải quyết. Vào những năm 1960, Bộ Quốc phòng Hoa Kỳ đã quan tâm đến loại công việc này và bắt đầu đào tạo máy tính để bắt chước lý luận cơ bản của con người. Ví dụ, Cơ quan Dự án Nghiên cứu Quốc phòng Tiên tiến (DARPA) đã hoàn thành các dự án lập bản đồ đường phố vào những năm 1970. Và DARPA đã sản xuất trợ lý cá nhân thông minh vào năm 2003…Trí tuệ nhân tạo là đột phá công nghệ mới nhất, là ngành khoa học đang định hình lại xã hội của chúng ta. Đồng thời trí tuệ nhân tạo có tác động sâu sắc đến các ngành công nghiệp máy móc và công ty cung cấp năng lượng.

Với sự phát triển nhanh chóng về kiến thức và tiến bộ trong trí tuệ nhân tạo đang thúc đẩy mạnh mẽ nhu cầu về các dịch vụ kỹ thuật số mớiđể giúp khai thác công

nghệ này với tiềm năng cao nhất. Cải thiện các sản phẩm đã có trên thị trường để tất cả chúng ta đưa vào sử dụng trong cuộc sống hàng ngày sẽ là cốt lõi cho tương lai của trí tuệ nhân tạo AI. Đa số trọng tâm của các nghiên cứu trí tuệ nhân tạo ban đầu được lấy từ cách tiếp cận bằng thực nghiệm của tâm lý học, và xem trọng cái gọi là "trí tuệ ngôn ngữ" - việc hiểu biết ngôn ngữ con người.

Các hướng nghiên cứu về trí thông minh nhân tạo khôngliên quan đến ngôn ngữ bao gồm ngành robotic và ngành thông minh tập thể (colective intelligence). Hai hướng tiếp cận này tập trung vào việc chủ động tác động lên môi trường hoặc việc đưa ra quyết định bằng đồng thuận về một vấn đề nào đó. Các hướng nghiên cứu này có nguồn gốc từ các mô hình thông minh xuất phát từ sinh học và chính trị học.

Lý thuyết trí tuệ nhân tạo còn được rút ra từ các nghiên cứu về động vật, đặc biệt là côn trùng, do có thể dễ dàng được mô phỏng con trùng bằng robot cũng như các động vật với nhận thức phức tạp hơn, ví dụ như loài khỉ, chúng có thể bắt chước con người trong nhiều trường hợp nhưng lại hạn chế về sự phát triển trong việc lập kế hoạch và nhận thức. Các nhà nghiên cứu về trí tuệ nhân tạo đưa ra kết luận rằng những loài động vật, có nhận thức đơn giản hơn con người, đáng ra có thể dễ dàng mô phỏng được. Tuy nhiên, đến giờ vẫn chưa có mô hình tính toán đủ tốt mô phỏng trí thông minh của động vật

Có những bài báo đưa ra khái niệm trí thông minh của máy (machine intelligence) như bài A Logical Calculus of theIdeas Immanent in Nervous Activity (Tính toán logic của các ý tưởng nội tại trong hoạt động thần kinh) (1943), do WarrenMcCulloch và Walter Pitts viết; On Computing Machinery andIntelligence (Về bộ máy tính toán và trí thông minh) (1950),được viết bởi Alan Turing; và Man-Computer Symbiosis viết bởi J.C.R.Licklider. Xem thêm phần điều khiển học (cybernetics) và Thử thách Turing. Với sự phát triển của các kỹ thuật thực hành dựa trên các nghiên cứu về trí tuệ nhân tạo, những người ủng hộ ngành trí tuệ nhân tạo đã cho rằng phe chống đối ngành này đã liên tục thay đổi lập trường của họ trong các vấn đề như máy chơi cờ hay nhận dạng tiếng nói, mà trước đây chúng đã từng được coi là thông minh, để phủ nhận các thành tựu của ngành trítuệ nhân tạo. Bởi vậy, Douglas Hofstadter, trong cuốn Gödel, Escher, Bach, đã chỉ ra rằng chính sự chuyển dịch đó đã

định nghĩa trí thông minh là bất cứ việc gì mà con người làm được còn máy móc thì không.

John von Neumann (trích dẫn trong E.T. Jaynes) đã thấy trước được điều này vào năm 1948 khi trả lời một lời bình luận tại một buổi diễn thuyết cho rằng may móc không thể suy nghĩ:"Bạn nhất quyết rằng có một điều gì đó mà máy móc không thể làm được. Nếu bạn nói cho tôi một cách chính xác đó là điều gì,thì tôi sẽ luôn luôn làm được một cái máy mà sẽ chỉ thực hiện được điều đó!". Von Neumann được cho là đã có ý nói đến luận đề Church-Turing khi khẳng định rằng bất kì một thủ tục có hiệu lực nào cũng có thể được mô phỏng bởi một máy tính (tổng quát) nào đó.

Vào năm 1969 McCarthy và Hayes đã bắt đầu thảo luậnvề bài toán khung (frame problem) với bài luận của họ, SomePhilosophical Problems from the Standpoint of ArtificialIntelligence (Một số vấn đề triết học từ điểm khởi đầu của trí tuệ nhân tạo).

Công việc ban đầu này đã mở đường cho tự động hóa và lýluận chính thức mà chúng ta thấy trong các máy tính ngày nay, bao gồm các hệ thống hỗ trợ quyết định và hệ thống tìm kiếm thông minh có thể được thiết kế để bổ sung và tăng cường khả năng của con người.

Ngày nay, hầu hết các thuật toán AI đang đạt đến trạng thái ngang bằng với con người, có nghĩa là chúng có thể thực hiện một nhiệm vụ với trí thông minh giống như con người. Chúng cũng hiện diện trong các tương tác hàng ngày trên Internet, từ hệ thống đề xuất đến các thuật toán xếp hạng tìm kiếm

1.3 Phân loại

Công nghệ AI thường được chia làm 4 loại chính:

Loại 1: Công nghệ AI phản ứng.

Công nghệ AI phản ứng có khả năng phân tích những động thái khả thi nhất của chính mình và của đối thủ, từ đó, đưa ra được giải pháp tối ưu nhất. Một ví dụ điển hình của công nghệ AI phản ứng là Deep Blue. Đây là một chương trình chơi cờ vua tự động, được tạo ra bởi IBM, với khả năng xác định các nước cờ đồng thời dự đoán

những bước đi tiếp theo của đối thủ. Thông qua đó, Deep Blue đưa ra những nước đi thích hợp nhất.

Loại 2: Công nghệ AI với bộ nhớ hạn chế

Đặc điểm của công nghệ AI với bộ nhớ hạn chế là khả năng sử dụng những kinh nghiệm trong quá khứ để đưa ra những quyết định trong tương lai. Công nghệ AI này thường kết hợp với cảm biến môi trường xung quanh nhằm mục đích dự đoán những trường hợp có thể xảy ra và đưa ra quyết định tốt nhất cho thiết bị.Ví dụ như đối với xe không người lái, nhiều cảm biến được trang bị xung quanh xe và ở đầu xe để tính toán khoảng cách với các xe phía trước, công nghệ AI sẽ dự đoán khả năng xảy ra va chạm, từ đó điều chỉnh tốc độ xe phù hợp để giữ an toàn cho xe.

Loại 3: Lý thuyết trí tuệ nhân tạo

Công nghệ AI này có thể học hỏi cũng như tự suy nghĩ, sau đó áp dụng những gì học được để thực hiện một việc cụ thể. Hiện nay, công nghệ AI này vẫn chưa trở thành một phương án khả thi.

Loại 4: Tự nhận thức

Công nghệ AI này có khả năng tự nhận thức về bản thân, có ý thức và hành xử như con người. Thậm chí, chúng còn có thể bộc lộ cảm xúc cũng như hiểu được những cảm xúc của con người. Đây được xem là bước phát triển cao nhất của công nghệ AI và đến thời điểm hiện tại, công nghệ này vẫn chưa khả thi.

1.4 Ứng dụng vai trò của trí tuệ nhân tạo

- **Trong ngành vận tải**

Trí tuệ nhân tạo được ứng dụng trên những phương tiện vận tải tự lái, điển hình là ô tô. Sự ứng dụng này góp phần mang lại lợi ích kinh tế cao hơn nhờ khả năng cắt giảm chi phí cũng như hạn chế những tai nạn nguy hiểm đến tính mạng. Vào năm 2016, Otto, hãng phát triển xe tự lái thuộc Uber đã vận chuyển thành công 50.000 lon bia Budweisers bằng xe tự lái trên quãng đường dài 193 km. Theo dự đoán của công ty tư vấn công nghệ thông tin Gartner, trong tương lai, những chiếc xe có thể kết nối với

nhau thông qua Wifi để đưa ra những lộ trình vận tải tốt nhất.

- **Trong sản xuất**

Trí tuệ nhân tạo được ứng dụng để xây dựng những quy trình sản xuất tối ưu hơn. Công nghệ AI có khả năng phân tích cao, làm cơ sở định hướng cho việc ra quyết định trong sản xuất.

- **Trong y tế**

Ứng dụng tiêu biểu của trí tuệ nhân tạo trong lĩnh vực y tế là máy bay thiết bị bay không người lái được sử dụng trong những trường hợp cứu hộ khẩn cấp. Thiết bị bay không người lái có tốc độ nhanh hơn xe chuyên dụng đến 40% và vô cùng thích hợp để sử dụng ở những nơi có địa hình hiểm trở.

- **Trong giáo dục**

Sự ra đời của trí tuệ nhân tạo giúp tạo ra những thay đổi lớn trong lĩnh vực giáo dục. Các hoạt động giáo dục như chấm điểm hay dạy kèm học sinh có thể được tự động hóa nhờ công nghệ AI. Nhiều trò chơi, phần mềm giáo dục ra đời đáp ứng nhu cầu cụ thể của từng học sinh, giúp học sinh cải thiện tình hình học tập theo tốc độ riêng của mình.

Trí tuệ nhân tạo còn có thể chỉ ra những vấn đề mà các khóa học cần phải cải thiện. Chẳng hạn như khi nhiều học sinh được phát hiện là gửi đáp án sai cho bài tập, hệ thống sẽ thông báo cho giáo viên đồng thời gửi thông điệp đến học sinh để chỉnh sửa đáp án phù hợp. Công nghệ AI còn có khả năng theo dõi sự tiến bộ của học sinh và thông báo đến giáo viên khi phát hiện ra vấn đề đối với kết quả học tập của học sinh.

Hơn nữa, sinh viên còn có thể học hỏi từ bất cứ nơi nào trên thế giới thông qua việc sử dụng những phần mềm có hỗ trợ AI. Công nghệ AI cũng cung cấp dữ liệu nhằm giúp sinh viên lựa chọn được những khóa học tốt nhất cho mình.

- **Trong truyền thông**

Đối với lĩnh vực truyền thông, sự phát triển của trí tuệ nhân tạo góp phần làm thay đổi cách thức tiếp cận đối với khách hàng mục tiêu. Nhờ những ưu điểm của công nghệ AI, các công ty có thể cung cấp quảng cáo vào đúng thời điểm, đúng khách hàng tiềm năng, dựa trên việc phân tích các đặc điểm về nhân khẩu học, thói quen hoạt động trực tuyến và những nội dung mà khách hàng thường xem trên quảng cáo.

- **Trong ngành dịch vụ**

Công nghệ AI giúp ngành dịch vụ hoạt động tối ưu hơn và góp phần mang đến những trải nghiệm mới mẻ hơn và tốt hơn cho khách hàng. Thông qua việc thu thập và phân tích dữ liệu, công nghệ AI có thể nắm bắt thông tin về hành vi sử dụng dịch vụ của khách hàng, từ đó mang lại những giải pháp phù hợp với nhu cầu của từng khách hàng.

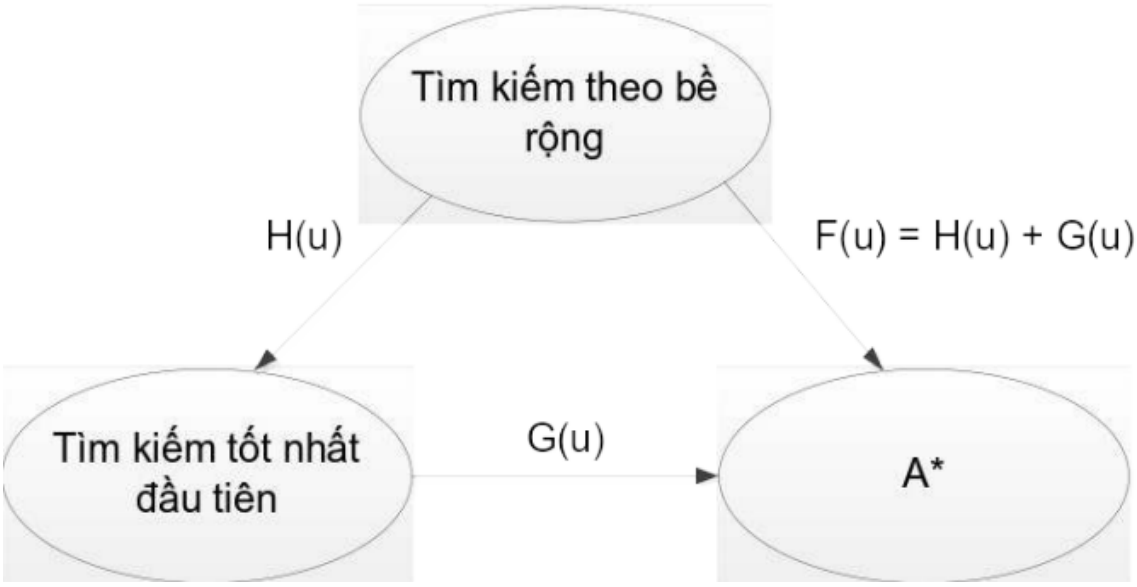
CHƯƠNG 2: TÌM HIỂU VỀ THUẬT TOÁN A*

2.1 Giới thiệu thuật toán

Trong khoa học máy tính, A* (đọc là *A sao*) là một thuật toán tìm kiếm trong đồ thị. Thuật toán này tìm một đường đi từ một nút khởi đầu tới một nút đích cho trước (hoặc tới một nút thỏa mãn một điều kiện đích). Thuật toán này sử dụng một "đánh giá heuristic" để xếp loại từng nút theo ước lượng về tuyến đường tốt nhất đi qua nút đó. Thuật toán này duyệt các nút theo thứ tự của đánh giá heuristic này. Do đó, thuật toán A* là một ví dụ của tìm kiếm theo lựa chọn tốt nhất (*best-first search*).

Thuật toán A* được mô tả vào năm 1968 bởi Peter Hart, Nils Nilsson, và Bertram Raphael. Trong báo cáo của họ, thuật toán được gọi là thuật toán A; khi sử dụng thuật toán này với đánh giá heuristic thích hợp sẽ thu được hoạt động tối ưu, do đó mà có A*.

Năm 1964, Nils Nilsson phát minh phương pháp tiếp cận dựa trên khám phá để tăng tốc độ của thuật toán Dijkstra. Thuật toán này được gọi là A1. Năm 1967, Bertram Raphael đã cải thiện đáng kể thuật toán này, nhưng không thể hiện tối ưu. Ông gọi thuật toán này là A2. Sau đó, trong năm 1968, Peter E. Hart đã giới thiệu một đối số chứng minh A2 là tối ưu sử dụng thuật toán này với một đánh giá heuristic thích hợp sẽ thu được hoạt động tối ưu. Chứng minh của ông về thuật toán cũng bao gồm một phần cho thấy rằng các thuật toán A2 mới là thuật toán tốt nhất có thể được đưa ra các điều kiện. Do đó ông đặt tên cho thuật toán mới này là A*(A sao, A-star).



*Hình 2.1 Tổng quan thuật toán A**

2.2 Mô tả thuật toán

Nếu u là một trạng thái hiện tại .

Ta xác định hàm đánh giá: f(u) = g(u) + h(u).

• g(u) là chi phí từ nút gốc u0 tới nút hiện tại u.

• h(u) là hàm đánh giá heuristic nhằm tínhchi phí nhỏ nhất có thể để đến đích từ u.

Hàm f(u) có giá trị càng thấp thì độ ưu tiên của u càng cao .

• f(u) tổng chi phí ước lượng của đường đi qua nút hiện tại u đến đích.

2.3 Cài đặt thuật toán

OPEN: bao gồm tập các trạng thái đang được chờ để xét. Trong tập OPEN ta sẽ ưu tiên cho nút có chi phí bé nhất so với các nút còn lại trong tập OPEN để phát triển.

CLOSE: tập chứa các trạng thái đã được xét đến. Nếu có một trạng thái khác đang set mà trùng với trạng thái trong tập CLOSE và có tổng chi phí đường đi tốt hơn thì trạng thái đó sẽ thay thế trạng thái cũ trong tập CLOSE.

Khi xét đến trạng thái u, trong OPEN bên cạnh việc lưu trữ 3 giá trị cơ bản h, g, f để so sánh độ ưu tiên của trạng thái đó, A* còn lưu trữ thêm hai thông số sau:

• Cha của u (ký hiệu Father(v)): cho biết đường đi dẫn đến trạng thái u.

• Danh sách các trạng thái kề với u: chứa các trạng thái kế tiếp v của u.

- Thuật toán.

BEGIN

1. Khởi tạo danh sách OPEN chỉ chứa trạng thái ban đầu, CLOSE = rỗng.

2. Lặp

2.1 If OPEN rỗng {thông báo tìm thất bại; stop};

2.2 Loại trạng thái u ở đầu danh sách OPEN

2.3 If u là trạng thái đích {thông báo đã tìm thành công; stop;}; 2.4 For mỗi trạng thái v kề u

{

g(v) = g(u) + k(u,v)

If v không thuộc OPEN hay CLOSE

{

f(v) g(v) + h(v);

Father (v) □ u

Đặt v vào danh sách OPEN;

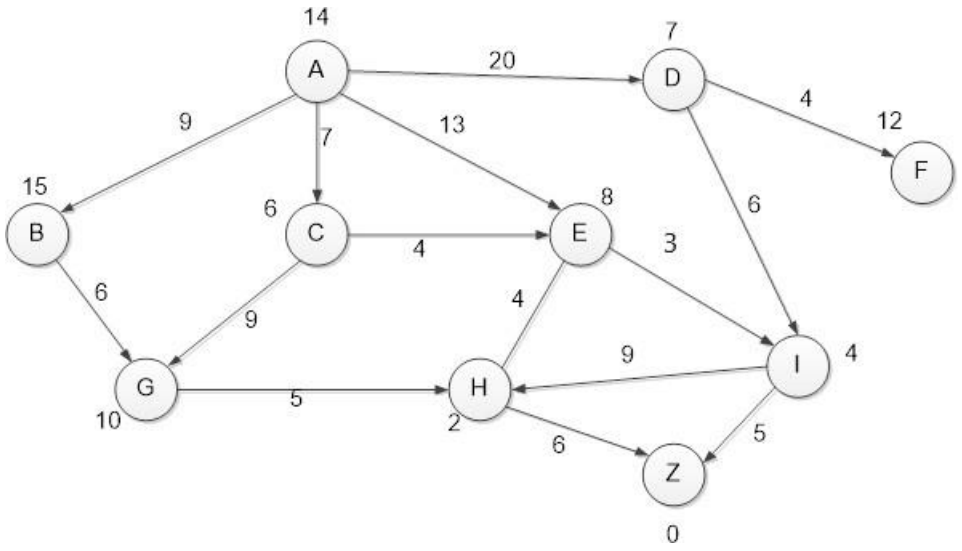
}

}

2.5 Sắp xếp OPEN theo thứ tự tăng dần của hàm đánh giá f sao cho trạng thái có giá trị của hàm f nhỏ nhất ở đầu danh sách.

END

2.4 Ví dụ minh họa tìm đường đi từ A -Z



Hình 2.2. Biểu đồ bào toán tìm đường đi ngắn nhất

Bảng 2.1. Bảng các bước tìm đường đi ngắn nhất

Bước lặp	u	v	OPEN	CLOSE	father
0			A		
1	A	B,C,D,E	C13,E21,B24,D27	A	
2	C	E,G	E21,B24,G26,D27	A,C	{A}
3	E	I	I18,E21,B24,G26,D27	A,C,E	{C}
4	I	H,Z	Z19,E21,B24,H25,G26,D27	A,C,E	{E}
5	Z			I	I

Vậy đường đi là ACEIZ.

2.5 Đánh giá thuật toán

Độ phưc tạp thời gian của A* phụ thuộc vào hàm đánh giá heuristic. Tùy từng bài toán mà có hàm đánh giá khách nhau.

Trong trường hợp xấu nhất, số nút được mở rộng theo hàm mũ của độ dài lời giải.

Độ phức tạp bộ nhớ A* còn rắc rối hơn độ phức tạp về thời thời gian. Trong trường hợp xấu nhất, A* phải ghi nhớ số lượng nút tăng theo hàm mũ.

2.6 Ưu điểm của A*

- Một thuật toán hiệu quả, tổng quát, linh động, tối ưu.

- Thuật toán chứa cả tìm kiếm theo chiều sâu, tìm kiếm theo bề rộng và những hàm đánh giá Heuristic khác.

-Tìm kiếm đến đích nhanh hơn với sự định hướng của hàm Heuristic.

2.7 Nhược điểm của A*

A* rất linh động nhưng vẫn gặp một khuyết điểm cơ bản đó là phải lưu lại các trạng thái đi qua nhằm so sánh với trạng thái hiên tại nên tốn khá nhiều bộ nhớ.

CHƯƠNG 3: ÁP DỤNG THUẬT GIẢI A* VÀO TRÒ CHƠI 8 CON SỐ

3.1 Mô tả bài toán

A* lưu giữ một tập các lời giải chưa hoàn chỉnh, nghĩa là các đường đi qua đồ thị, bắt đầu từ nút xuất phát. Tập lời giải này được lưu trong một hàng đợi ưu tiên (*priority queue*). Thứ tự ưu tiên gán cho một đường đi x được quyết định bởi hàm:

f(x) = g(x) + h(x)

Trong đó:

- g(x) là chi phí của đường đi cho đến thời điểm hiện tại, nghĩa là tổng trọng số của các cạnh đã đi qua.
- h(x) là hàm đánh giá heuristic về chi phí nhỏ nhất để đến đích từ x.

Ví dụ, nếu "chi phí" được tính là khoảng cách đã đi qua, khoảng cách đường chim bay giữa hai điểm trên một bản đồ là một đánh giá heuristic cho khoảng cách còn phải đi tiếp.

Hàm f(x) có giá trị càng thấp thì độ ưu tiên của x *càng cao (do đó có thể sử dụng một cấu trúc heap tối thiểu để cài đặt hàng đợi ưu tiên này) :*

function A*(điểm_xuất_phát,đích)

var đóng := *tập rỗng*

var q := tạo_hàng_đợi(tạo_đường_đi(điểm_xuất_phát))

while q *không phải tập rỗng*

var p := lấy_phần_tử_đầu_tiên(q)

var x := *nút cuối cùng của p*

if x **in** đóng

continue if

x = đích

return p

bổ sung x vào tập đóng

foreach y **in** các_đường_đi_tiếp_theo(p)

đưa_vào_hàng_đợi(q, y)

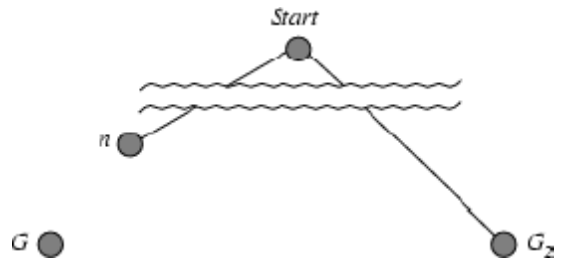
return failure

A*- search Các đặc điểm:

- Nếu không gian các trạng thái là hữu hạn và có giải pháp để tránh việc xét (lặp) lại các trạng thái, thì giải thuật A* là hoàn chỉnh (tìm được lời giải) – nhưng không đảm bảo là tối ưu.
- Nếu không gian các trạng thái là hữu hạn và không có giải pháp để tránh việc xét (lặp) lại các trạng thái, thì giải thuật A* là không hoàn chỉnh (không đảm bảo tìm được lời giải).
- Nếu không gian các trạng thái là vô hạn, thì giải thuật A* là không hoàn chỉnh (không đảm bảo tìm được lời giải).
- Một ước lượng (heuristic) *h(n)* được xem là chấp nhận được nếu đối với mọi nút *n*: $0 \leq h(n) \leq h^*(n)$, trong đó $h^*(n)$ là chi phí thật (thực tế) để đi từ nút *n* đến đích.
- Giải thuat A với hàm heuristic h(n) luôn luôn giá trị thực đi từ n đến goal.

Tính tối ưu của A* - Chứng minh:

Giả sử có một đích không tối ưu (suboptimal goal) *G2* được sinh ra và lưu trong cấu trúc *fringe*. Gọi *n* là một nút chưa xét trong cấu trúc *fringe* sao cho *n* nằm trênmột đường đi ngắn nhất đến một đích tối ưu (optimal goal) *G*.



Ta có: 1) f(G2) = g(G2) vì h(G2) = 0

Ta có: 2) g(G2) > g(G) vì G2 là đích không tối ưu Ta
có: 3) f(G) = g(G) vì h(G) = 0
Từ 1)+2)+3) suy ra: 4) f(G2) > f(G)

Ta có: 5) h(n) ≤ h*(n) vì h là ước lượng chấp nhận được Từ 5)
suy ra: 6) g(n) + h(n) ≤ g(n) + h*(n)
Ta có: 7) g(n) + h*(n) = f(G) vì n nằm trên đường đi tới G Từ
6)+7) suy ra: 8) f(n) ≤ f(G)
Từ 4)+8) suy ra: f(G2) > f(n). Tức là, giải thuật A* không bao giờ xét G2.

Đặc điểm:

Tính hoàn chỉnh?

- Có (trừ khi có rất nhiều các nút có chi phí f ≤ *f(G)*) Độ

phức tạp về thời gian?

- Bậc của hàm mũ – Số lượng các nút được xét là hàm mũ của độ dài đường đi của lời giải

Độ phức tạp về bộ nhớ?

- Lưu giữ tất cả các nút trong bộ nhớ

Tính tối ưu?

- Có

3.2 Sử dụng giải thuật A* vào bài toán 8 con số

- Như đã giới thiệu trong bài trước, có những trạng thái của bảng số không thể chuyển về trạng thái đích, ta gọi là cấu hình hợp lệ và không hợp lệ. Tỉ lệ giữa chúng là ½, điều này có thể nhận ra dễ dàng từ phương pháp tính xem bài toán có thể đưa về trạng thái đích hay không.

- 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). Bạn có

thể 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, bạn cần một hàm heuristic h để ước lượng giá trị của mỗi trạng thái của bảng số. Có một số cách bạn 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 tôi chọn theo cách thứ nhất, tức là tính tổng số ô sai lệch của các ô số so với vị trí đúng của nó. Đây là cách tính thường được sử dụng và nó có tên gọi là Manhattan.

a. Ý tưởng:

Thuật toán A*:

Gọi G là số bước đã di chuyển ô trống

H là hàm heuristic, ước tính số hao tổn để tới trạng thái đích, tính bằng tổng các quãng đường của các ô ở vị trí sai để về vị trí dùng $F_{(n)}=G_{(n)}+H_{(n)}$

Có hai danh sách Open và Close, Open chứa các trạng thái chưa xét, danh sách Close chứa các trạng thái đã xét.

Ban đầu ta thêm trạng thái khởi đầu vào Open, sau đó chọn trạng thái có f= g+ h nhỏ nhất, lúc này danh sách Open chứa duy nhất trạng thái khởi đầu nên ta lấy trạng thái khởi đầu khỏi Open và đưa vào danh sách Close các trạng thái đã xét. Từ trạng thái đang xét ta xác định được các trạng thái tiếp theo dựa vào các hướng di chuyển của ô trống. Đưa tất cả các trạng thái mới mà chưa có trong Close và Open vào danh sách Open. Ta tiếp tục chọn trạng thái có f = g+h nhỏ nhất khỏi Open như bước đầu tiên cho đến khi tìm ra trạng thái đích thì dừng lại. Từ trạng thái đích vừa tìm được đường đi từ trạng thái khởi đầu đến trạng thái đích.

b. Cách tính khoảng cách sai lệch của các ô số với vị trí đúng (hàm lượng giá h):

1	5	7
2	3	6
4		8

Trong bảng số 3×3 trên, để di chuyển ô số 5 vào đúng vị trí ta cần di chuyển nó 1 lần, để di chuyển ô số 7 về đúng vị trí ta cần cần 4 lần (qua 4 ô khác). Để có được kết quả này ta làm phép tính đơn giản: lấy tổng khoảng cách của dòng và cột giữa hai vị trí (ví dụ với ô số 7):

- Lấy tọa độ của ô số 7 ta có row1 = 0 và column1 = 2
- Lấy tọa độ của ô số 7 khi ở vị trí đúng, ta có row2 = 2 và column2 = 0
- Vậy khoảng cách Manhattan của hai ô này là:

| row1 - row2| + |column1 – column2| = |0 – 2| + |2 – 0| = 4 Theo đó, ta tính được **h = 0+1+4+2+2+0+1+1+1 = 12**

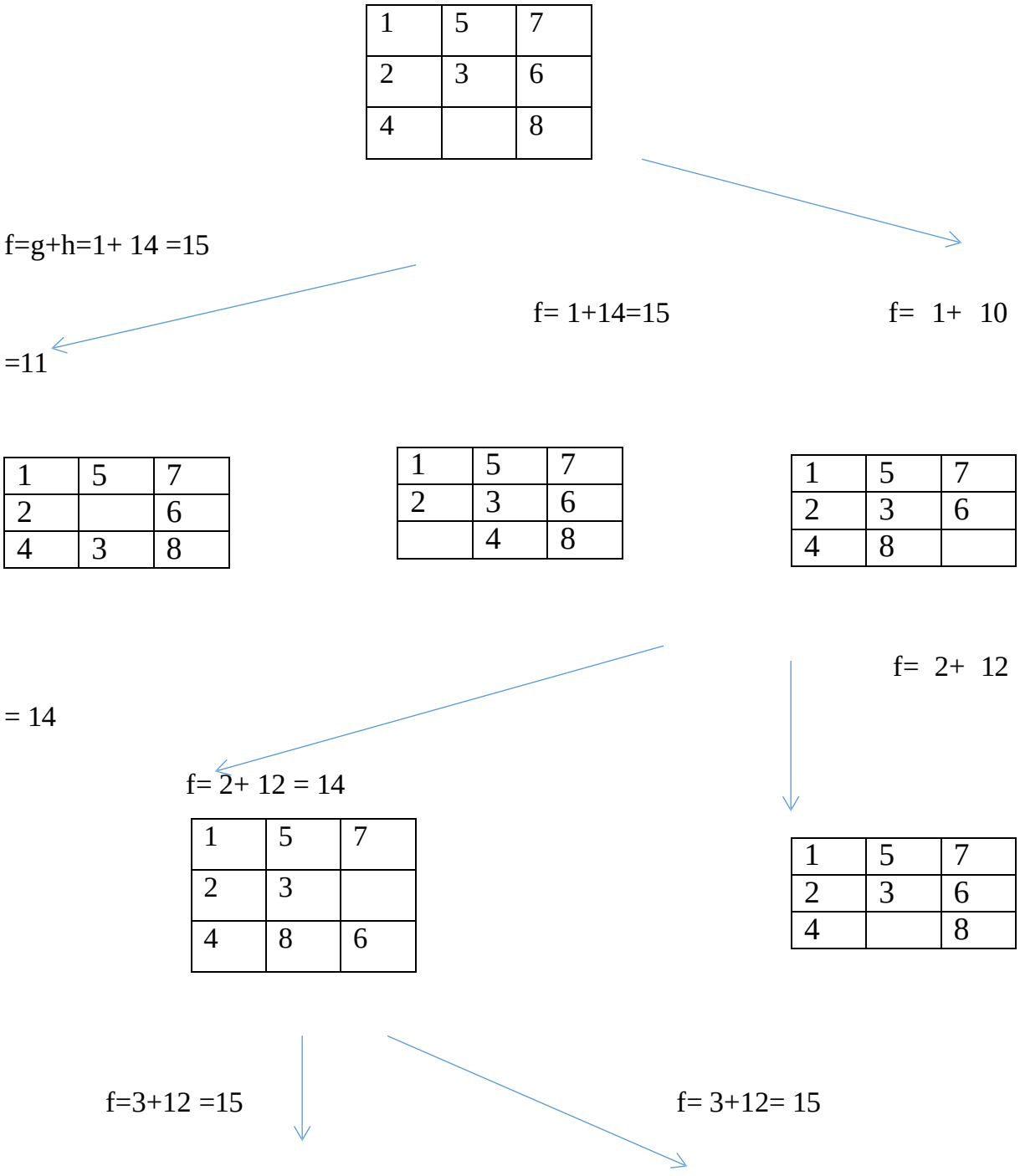
Như vậy ở trạng thái đích thì bảng số sẽ có giá trị thấp nhất là 0. Từ giá trị của một ô số ta tính vị trí dòng và cột của nó như sau:

Ví dụ ô số 7 có thứ tự trong bảng là 6 (tính từ 0 với m là cạnh) ta có row = 6 / 3 = 2, col = 6 % 3 = 0. Vậy tổng quát lại ta có:

RowIndex = Index / m ColIndex

= Index %

3.3 Biểu diễn không gian trạng thái



1	5	7
2		3
4	8	6

1	5	
2	3	7
4	8	6

3.4 Triển khai thuật toán và cài đặt thuật toán bằng C#

Trò chơi 8 con số ở mức độ khó vừa phải nên là trò chơi rất thú vị. Một giải pháp điển hình gồm 20 bước, mặc dù con số này biến đổi phụ thuộc vào trang thái ban đầu. Hệ số rẽ nhánh khoảng bằng 3 (kho ô trống ở giữa, có bốn khả năng di chuyển; khi nó ở góc có hai khả năng di chuyển; và khi nó ở trên các cạnh, có ba khả năng đi). Để giải quyết bài toán này ta cần tìm mọt hàm Heuristic tốt. Ta có hai hàm ước lượng:

H1 = số lượng các số sai vị trí

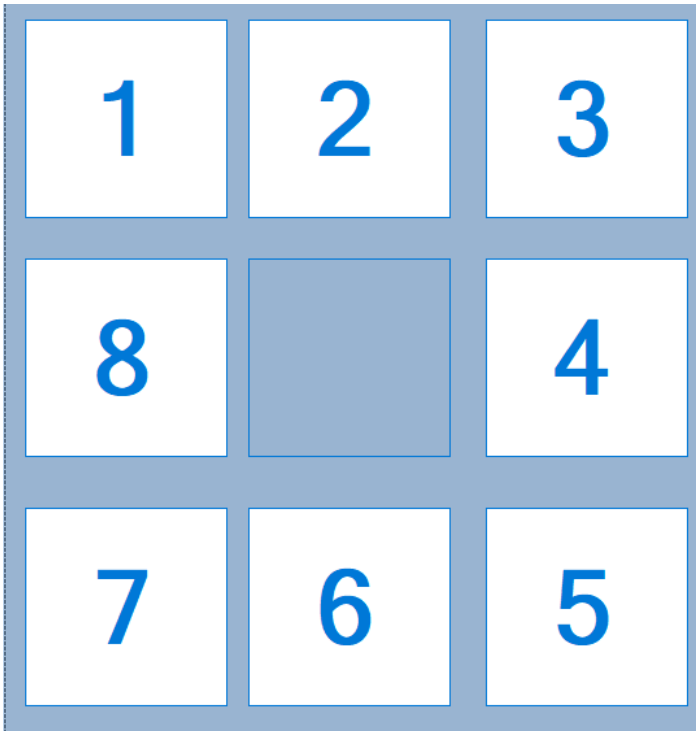
H2 = tổng số khoảng cách của các con số so với vị trí mục tiêu, là tổng khoảng cách theo chiều ngang và theo chiều dọc.

Bài toán tám cạnh khi được giải quyết bằng thuật toán A* sẽ thực hiện theo bước sau.

- Từ trạng thái ban đầu ta xác định được trạng thái đích.
- Goi G là số bước đã di chuyển ô trống**.** H là hàm heuristic, ước tính số hao tổn để tới trạng thái đích, tính bằng tổng các quãng đường của các ô vị trí sai để về tới vị trí đúng: F= G + H.
- Có hai danh sách Open và Close, Open chứa các trạng thái chưa xét, danh sách Close chứa các trạng thái đã xét
- Ban đầu ta thêm trạng thái khởi đầu vào Open, sau đó chọn trạng thái có f = g + h nhỏ nhất, lúc này danh sách Open duy nhất chứa trạng thái ban đầu nên ta lấy trạng thái khởi đầu khỏi Open, và đưa vào danh sách Close các trạng thái đã xét. Từ trạng thái đang xét t xác định được các trạng thái tiếp theo, dựa vào các hướng di chuyển của ô trống. Đưa tất cả các trạng thái mới mà chưa có trong Close và Open vào danh sách Open. Ta tiếp tục chọn trạng thái có f = g + h nhỏ nhất khỏi Open như bước đầu tiên cho đến khi tìm ra trạng thái đích thì dừng lại. Từ trạng thái đích vừa tìm được đi ngược lại danh sách ta sẽ tìm được

đường đi từ trạng thái khởi đầu đến trạng thái đích.

Ví dụ cho hình sau:



Hình 3.1 Hình minh họa

Các bước giải bài toán sau:

Đầu tiên ta xác định trạng thái tiếp theo của bài toán trên:

Có ba trường hợp xảy ra:

Đối với trường hợp 1 ta có: g = 1, h = 7, f = g + h = 8

Đối với trường hợp 2 ta có: g = 1, h = 8, f = g + h = 9

Đối với trường hợp 3 ta có: g = 1, h = 6, f = g + h = 7

So sánh các f với nhau ta thấy f của trường hợp 1 nhỏ nhất nên trạng thái tiếp theo là trạng thái 1.

Từ 1 ta có ba trạng thái:

Đối với trường hợp 1.1 ta có: g = 2, h = 6, f = g + h = 8

Đối với trường hợp 1.2 ta có: g = 2, h = 6, f = g + h = 8

Đối với trường hợp 1.3 ta có: g = 2, h = 8, f = g + h = 10

Cài đặt thuật toán:

Ta tiến hành cài đặt thuật toán trên ngôn ngữ Visual C# Ta
xây dựng các class sau:

- Class Node: Xác định thuộc tính mỗi trạng thái của bài toán. Trong lớp này xây dựng hàm HeuristicFunction() để xác định độ ước lượng h.
- Class Puzzles: Có các thành phần là các danh sách Open và Close kiểu LinkedNode<Node>: ngoài ra lớp này còn có các phương thức để thực hiện thuật toán A*:

  - Hàm GetSmallListNode(): để lấy trạng thái nhỏ nhất khỏi danh sách Open.
  - Hàm Open.Remove(SmallestNode): để xóa trạng thái nhỏ nhất vừa được chọn khỏi Open.
  - Hàm Close.AddFirst(SmallestNode): để thêm trạng thái nhỏ nhất đã được xét vào Close.
  - Hàm AddNodeToOpen(): thêm các trạng thái chưa được xét vào Open.

Ta dùng vòng lặp while(Open!=null) để duyệt danh sách Open và chọn ra trạng thái nhỏ nhất. Danh sách Open được duyệt đến khi tìm được trạng thái đích.

Ngoài ra chương trình còn có class giao diện dùng để xây dựng giao diện trực quan cho bài toán.

a. Hàm khởi tạo Node

```
public class Node
{
```

```
    public int[,] MaTran;// ma trận 8 số
    public int SoManhSai;//số mảnh sai vị trí của ma trận
    public int ChiSo;// chỉ số của node
    public int Cha;// cha của node, để truy vét kết quả
    public int fn;// chi phí đi đến node đó
}
```

b. Hàm đánh giá

Tùy vào từng bài toán, mỗi bài toán có hàm đánh giá khác nhau. Trong bài toán ghép tranh thì hàm đánh giá sẽ dựa vào số mảnh sai so với vị trí đích cộng với chi phí đã đi tới nút đó.

```
int soMiengSaiViTri(int[,] MaTran)
{
    int dung = 0;
    int t = 0;
    for (int i = 0; i < MaTran.GetLength(0); i++)
    {
        if (i == 0)
            for (int j = 0; j < MaTran.GetLength(0); j++)
            {
                t++;
                if (MaTran[i, j] == t)
                    dung++;
            }
        else
        {
            if (MaTran[1, 2] == 4)
                dung++;
```

```
            if (MaTran[2, 2] == 5)
                dung++;
            if (MaTran[2, 1] == 6)
                dung++;
            if (MaTran[2, 0] == 7)
                dung++;
            if (MaTran[1, 0] == 8)
                dung++;

            break;
        }
    }
    return 8-dung;
}
```

c. Hàm tìm vị trí nhỏ nhất của danh sách Open

Đưa các nút vào tập Open, sau đó sắp xếp chúng sao cho vị trí tốt nhất đứng đầu của danh sách.

Nếu tồn tại 2 trạng thái có số mảnh sai = nhau

```
int viTriTotNhatOpen(List<Node> Open)
{
    if (Open.Count != 0)
    {
        Node min = new Node();
        min = Open[0];
        int vt = 0;

        for (int i = 1; i < Open.Count; i++)
            if (min.SoManhSai > Open[i].SoManhSai)
            {
```

```
                min = Open[i];
                vt = i;
            }
            else
            {
                if (min.SoManhSai == Open[i].SoManhSai)
                {
                    if (min.fn > Open[i].fn)
                    {
                        min = Open[i];
                        vt = i;
                    }
                }
            }
        return vt;
    }

    return 0;

}
```

d. Hàm kiểm tra trong danh sách Close

```
bool danhSachDaCoMaTran(int[,] a, List<int[,]> Close)
{
    for(int i=0;i<Close.Count;i++)
    {
        if(haiMaTranBangNhau(a,Close[i]))
        {
            return true;
        }
    }
```

```
            return false;
        }
```

e. Hàm sinh ngẫu nhiên trạng thái đầu

```
        // sinh một ma trận ngẫu nhiên để làm node bắt đầu
public int[,] randomMaTran(int kickThuoc)
{

    int[,] MaTran = new int[kickThuoc, kickThuoc];
    //khởi tạo ma trận 8 số
    MaTran[0, 0] = 1;
    MaTran[0, 1] = 2;
    MaTran[0, 2] = 3;
    MaTran[1, 0] = 8;
    MaTran[1, 1] = 0;
    MaTran[1, 2] = 4;
    MaTran[2, 0] = 7;
    MaTran[2, 1] = 6;
    MaTran[2, 2] = 5;


    //tập Close lưu lại các hướng đã đi để đảm bảo sinh ra hướng đi mới không trùng lặp
    List<int[,]> Close = new List<int[,]>();

    int n = MaTran.GetLength(0);

    int[,] Temp = new int[n,n];
    Array.Copy(MaTran, Temp, MaTran.Length);
    Close.Add(Temp);
```

```
int h = 1, c = 1;

Random rd = new Random();

int m = rd.Next(10, 200);//lấy số lần lặp sinh hướng đi
int t = rd.Next(1, 5);// t=[1...4] tương ứng với 4 hướng đi

//số lần lặp được lấy random từ đó số lượng hướng đi sẽ thay đổi theo
for (int r = 0; r < m; r++)
{
   // vì t được lấy random nên hướng đi sẽ ngẫu nhiên, có thể lên, xuống, trái, phải
tùy vào biến t

   //đi lên trên với t =1
   if (h > 0 && h <= n - 1&&t==1)
   {
      MaTran[h, c] = MaTran[h - 1, c];
      MaTran[h - 1, c] = 0;

      if (!danhSachDaCoMaTran(MaTran, Close))
      {
         h--;
         Temp = new int[n, n];
         Array.Copy(MaTran, Temp, MaTran.Length);
         Close.Add(Temp);
      }
      else
      {
         MaTran[h - 1, c] = MaTran[h, c];
         MaTran[h, c] = 0;
```

```
    }

}

t = rd.Next(1, 5);

//đi sang trái với t=2
if (c > 0 && c <= n - 1&&t==2)
{
    MaTran[h, c] = MaTran[h, c - 1];
    MaTran[h, c - 1] = 0;

    if (!danhSachDaCoMaTran(MaTran, Close))
    {
        c--;
        Temp = new int[n, n];
        Array.Copy(MaTran, Temp, MaTran.Length);
        Close.Add(Temp);
    }
    else
    {
        MaTran[h, c - 1] = MaTran[h, c];
        MaTran[h, c] = 0;
    }
}

t = rd.Next(1, 5);

//đi xuống giưới với t=3
if (h < n - 1 && h >= 0&&t==3)
```

```
{
    MaTran[h, c] = MaTran[h + 1, c];
    MaTran[h + 1, c] = 0;

    if (!danhSachDaCoMaTran(MaTran, Close))
    {
        h++;
        Temp = new int[n, n];
        Array.Copy(MaTran, Temp, MaTran.Length);
        Close.Add(Temp);
    }
    else
    {
        MaTran[h + 1, c] = MaTran[h, c];
        MaTran[h, c] = 0;
    }

}

t = rd.Next(1, 5);

//đi sang phải với t = 4
if (c < n - 1 && c >= 0&&t==4)
{
    MaTran[h, c] = MaTran[h, c + 1];
    MaTran[h, c + 1] = 0;

    if (!danhSachDaCoMaTran(MaTran, Close))
    {
        c++;
```

```
                Temp = new int[n, n];
                Array.Copy(MaTran, Temp, MaTran.Length);
                Close.Add(Temp);
            }
            else
            {
                MaTran[h, c + 1] = MaTran[h, c];
                MaTran[h, c] = 0;
            }
        }

    }

    // trả về hướng đi cuối dùng trong danh sách hướng đi
    return Close[Close.Count-1];

 }
```

f. Hàm tìm kiếm kết quả đưa về trang thái đích

```
        public Stack<int[,]> timKetQua(int[,] MaTran, int n)
{

   Stack<int[,]> stkKetQua = new Stack<int[,]>();

   List<Node> Close = new List<Node>();
   List<Node> Open = new List<Node>();

   //khai báo và khởi tạo cho node đầu tiên
   Node tSo = new Node();
   tSo.MaTran = MaTran;
   tSo.SoManhSai = soMiengSaiViTri(MaTran);
```

```
tSo.ChiSo = 0;
tSo.Cha = -1;
tSo.fn = 0;
//cho trạng thái đầu tiên vào Open;
Open.Add(tSo);

int t = 0;
while(Open.Count!=0)
{
    #region chọn node tốt nhất trong tập Open và chuyển nó sang Close
    tSo = new Node();
    tSo = Open[t];
    Open.Remove(tSo);
    Close.Add(tSo);
    #endregion

    //nếu node có số mảnh sai là 0, tức là đích thì thoát
    if (tSo.SoManhSai == 0) break;
    else
    {
        //sinh hướng đi của node hiện tại
        List<Node> lstHuongDi = new List<Node>();
        lstHuongDi = sinhHuongDi(tSo);

        for (int i = 0; i < lstHuongDi.Count; i++)
        {
            //hướng đi không thuộc Open và Close
            if (!haiNodeTrungNhau(lstHuongDi[i], Open) &&
!haiNodeTrungNhau(lstHuongDi[i], Close))
            {
```

```
            Open.Add(lstHuongDi[i]);
        }
        else
        {   //nếu hướng đi thuộc Open
            if (haiNodeTrungNhau(lstHuongDi[i], Open))
            {
                /*nếu hướng đi đó tốt hơn thì sẽ được cập nhật lại,
                ngược lại thì sẽ không cập nhật*/
                soSanhTotHon(lstHuongDi[i], Open);
            }
            else
            {
                //nếu hướng đi thuộc Close
                if (haiNodeTrungNhau(lstHuongDi[i], Close))
                {
                    /*nếu hướng đi đó tốt hơn thì sẽ được cập nhật lại,
                    ngược lại thì sẽ không cập nhật và chuyển từ Close sang Open*/
                    if (soSanhTotHon(lstHuongDi[i], Close))
                    {
                        Node temp = new Node();
                        temp = layNodeTrungTrongClose(lstHuongDi[i], Close);
                        Close.Remove(temp);
                        Open.Add(temp);
                    }
                }
            }
        }

    }
```

```
            //chọn vị trí có phí tốt nhất trong Open
            t = viTriTotNhatOpen(Open);
        }

    }

    //truy vét kết quả tổng tập Close
    stkKetQua = truyVetKetQua(Close);

    return stkKetQua;
}
```

g. Lưu các nút cha để lấy ra đường đi

```
Stack<int[,]> truyVetKetQua(List<Node> Close)
{
    Stack<int[,]> ketQua = new Stack<int[,]>();

    int t = Close[Close.Count - 1].Cha;
    Node temp = new Node();
    ketQua.Push(Close[Close.Count - 1].MaTran);

    while (t != -1)
    {
        for (int i = 0; i < Close.Count; i++)
        {
            if (t == Close[i].ChiSo)
            {
                temp = Close[i];
                break;
            }
        }
```

```
        ketQua.Push(temp.MaTran);
        t = temp.Cha;
    }

    return ketQua;
}
```

h. Hàm tạo ra hướng đi

```
List<Node> sinhHuongDi(Node tSo)
{
    int n = tSo.MaTran.GetLength(0);//lấy số hàng của ma trận

    List<Node> lstHuongDi = new List<Node>();

    #region  Xác định vị trí mảnh chống, có giá trị là 0
    int h = 0;
    int c = 0;
    bool ok = false;
    for (h = 0; h < n; h++)
    {
        for (c = 0; c < n; c++)
            if (tSo.MaTran[h, c] == 0)
            {
                ok = true;
                break;
            }

        if (ok) break;
    }
    #endregion
```

```
Node Temp = new Node();
Temp.MaTran = new int[n, n];
//Copy mảng Ma trận sang mảng ma trận tạm
Array.Copy(tSo.MaTran, Temp.MaTran, tSo.MaTran.Length);

fn++;// tăng chi phí của node con lên 1 đơn vị

#region Xét các hướng đi theo 4 hướng: trên, dưới, phải, trái
//xét hàng ngang bắt đầu từ hàng thứ 2 trở đi
if (h > 0 && h <= n - 1)
{
    // thay đổi hướng đi của ma trận
    Temp.MaTran[h, c] = Temp.MaTran[h - 1, c];
    Temp.MaTran[h - 1, c] = 0;

    //cập nhật lại thông số của node
    Temp.SoManhSai = soMiengSaiViTri(Temp.MaTran);
    ChiSo++;
    Temp.ChiSo = ChiSo;
    Temp.Cha = tSo.ChiSo;
    Temp.fn = fn + Temp.SoManhSai;
    lstHuongDi.Add(Temp);

    //sau khi thay đổi ma trận thì copy lại ma trận cha cho MaTran để xét trường hợp
tiếp theo
    Temp = new Node();
    Temp.MaTran = new int[n, n];
    Array.Copy(tSo.MaTran, Temp.MaTran, tSo.MaTran.Length);
}
```

```
    //xét hàng ngang bắt đầu từ hàng thứ cuối cùng - 1 trở xuống
    if (h < n - 1 && h >= 0)
    {
        // thay đổi hướng đi của ma trận
        Temp.MaTran[h, c] = Temp.MaTran[h + 1, c];
        Temp.MaTran[h + 1, c] = 0;

        //cập nhật lại thông số của node
        Temp.SoManhSai = soMiengSaiViTri(Temp.MaTran);
        ChiSo++;
        Temp.ChiSo = ChiSo;
        Temp.Cha = tSo.ChiSo;
        Temp.fn = fn + Temp.SoManhSai;
        lstHuongDi.Add(Temp);

        //sau khi thay đổi ma trận thì copy lại ma trận cha cho MaTran để xét trường hợp
tiếp theo
        Temp = new Node();
        Temp.MaTran = new int[n, n];
        Array.Copy(tSo.MaTran, Temp.MaTran, tSo.MaTran.Length);
    }
    //Xét cột dọc bắt đầu từ cột thứ 2 trở đi
    if (c > 0 && c <= n - 1)
    {
        // thay đổi hướng đi của ma trận
        Temp.MaTran[h, c] = Temp.MaTran[h, c - 1];
        Temp.MaTran[h, c - 1] = 0;

        //cập nhật lại thông số của node
        Temp.SoManhSai = soMiengSaiViTri(Temp.MaTran);
```

```
            ChiSo++;
            Temp.ChiSo = ChiSo;
            Temp.Cha = tSo.ChiSo;
            Temp.fn = fn + Temp.SoManhSai;
            lstHuongDi.Add(Temp);

            //sau khi thay đổi ma trận thì copy lại ma trận cha cho MaTran để xét trường hợp tiếp theo
            Temp = new Node();
            Temp.MaTran = new int[n, n];
            Array.Copy(tSo.MaTran, Temp.MaTran, tSo.MaTran.Length);
        }
        //Xét cột dọc bắt đầu từ cột cuối cùng -1 trở xuống
        if (c < n - 1 && c >= 0)
        {
            // thay đổi hướng đi của ma trận
            Temp.MaTran[h, c] = Temp.MaTran[h, c + 1];
            Temp.MaTran[h, c + 1] = 0;

            //cập nhật lại thông số của node
            Temp.SoManhSai = soMiengSaiViTri(Temp.MaTran);
            ChiSo++;
            Temp.ChiSo = ChiSo;
            Temp.Cha = tSo.ChiSo;
            Temp.fn = fn + Temp.SoManhSai;
            lstHuongDi.Add(Temp);

            //đến đây đã xét hết hướng đi nên không cần copy lại ma trận
        }
        #endregion
```

```
    return lstHuongDi;
}
```

i. Hàm di chuyển

```
public bool dichuyen(int[,] maTran, int x, int y)
{
    int n = maTran.GetLength(0);

    // Tìm vị trí ô trống (0)
    int x0 = -1, y0 = -1;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (maTran[i, j] == 0)
            {
                x0 = i;
                y0 = j;
                break;
            }
        }
    }

    // Kiểm tra xem vị trí được click có nằm cạnh ô trống không
    if ((Math.Abs(x - x0) == 1 && y == y0) || (Math.Abs(y - y0) == 1 && x == x0))
    {
        // Hoán đổi vị trí
        int temp = maTran[x, y];
        maTran[x, y] = maTran[x0, y0];
```

```
            maTran[x0, y0] = temp;
            return true;
        }

        return false;
    }
```

j. Hàm Kiểm tra hoàn thành

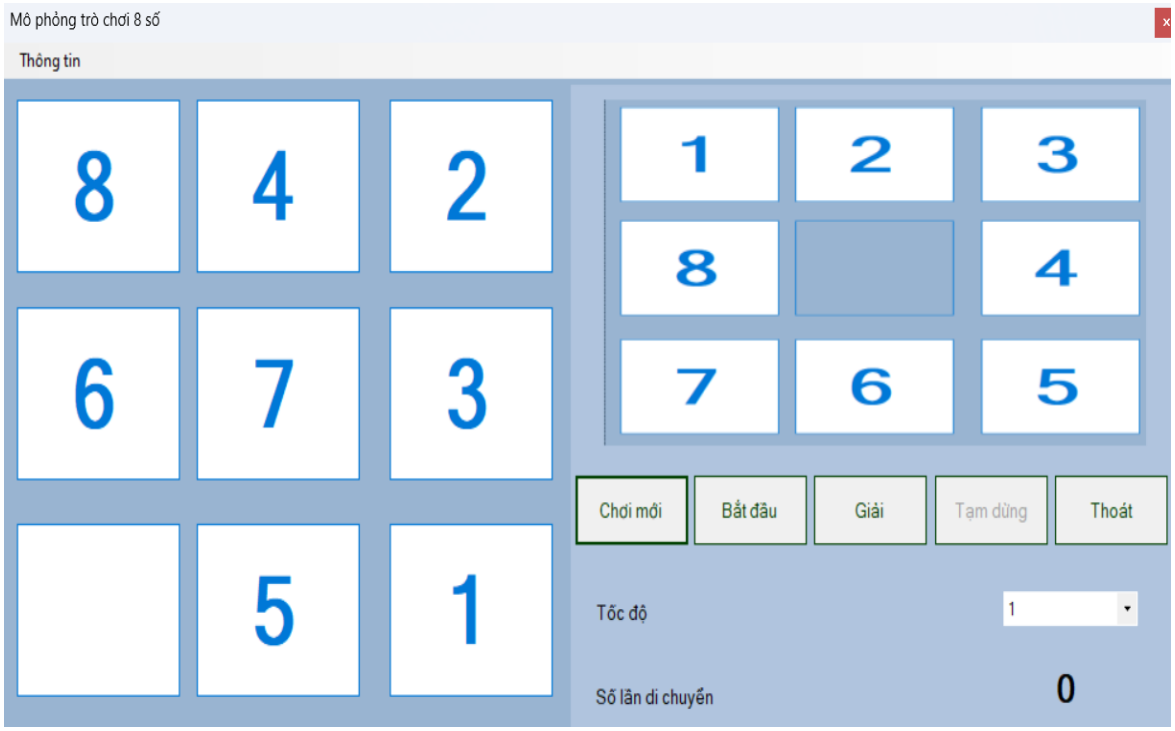
```
    public bool kiemTraHoanThanh(int[,] maTran)
    {

        // Trạng thái mong muốn khi hoàn thành trò chơi
        int[,] maTranDung = {
            { 1, 2, 3 },
            { 8, 0, 4 },
            { 7, 6, 5 }
        };

        for (int i = 0; i < 3; i++)
        {
            for (int j = 0; j < 3; j++)
            {
                if (maTran[i, j] != maTranDung[i, j])
                {
                    return false;
                }
            }
        }
        return true;
    }
```

3.5 Kết quả thu được

Trạng thái ban đầu



Hình 3.2 Trạng thái bắt đầu

Trạng thái kết thúc

Hình 3.3 Trạng thái kết thúc

KẾT LUẬN

Bài báo này được dựa trên những gì chúng em tìm hiểu ở trên lớp và rất nhiều tài liệu tham khảo khác. Sau một thời gian tìm tòi thì ngoài những ưu điểm đã được chứng minh ở trên thì thuật giải A* còn có nhược điểm là tốn bộ nhớ để lưu lại những trạng thái. Vì vậy chỉ biết mỗi thuật giải A* là chưa đủ, để phát triển bản thân thì mỗi cá nhân cần phải luôn luôn học hỏi và cố gắng từng ngày, tìm hiểu nhiều phương pháp khác nhau để linh động áp dụng cho nhiều trường hợp cụ thể. Nhìn rộng ra hơn thì AI tốt thật đấy nhưng con người không được cho phép bị lệ thuộc vào AI hay bị AI làm chủ. Mục đích của lĩnh vực AI cốt lõi phải là tạo ra những sản phẩm phục vụ con người, đem lại tiện ích cho cộng đồng và gián tiếp thúc đẩy sự phát triển của con người.

DANH MỤC THAM KHẢO

[1] Bộ thư viện chuẩn của Sun MicroSystem

[2] http://www.oracle.com/us/technologies/java/index.html

[3] Bài giảng Nhập môn trí tuệ nhân tạo – Nguyễn Hà Nam

[4] http://en.wikipedia.org/wiki/A*_search_algorithm

[5] http://en.wikipedia.org/wiki/Fifteen_puzzle