TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI

TRÍ TUỆ NHÂN TẠO

Ngành Kỹ thuật phần mềm

ĐỀ TÀI: XÂY DỰNG CHƯƠNG TRÌNH GIẢI BÀI TOÁN TRÒ CHƠI 8 SỐ VỚI THUẬT TOÁN CHIỀU RỘNG, CHIỀU SÂU, SÂU LẶP DẦN VÀ A* (C++)

Giảng viên HD: TS. Trần Thanh Huân

Nhóm 2: VŨ HỒNG PHƯƠNG - 2020601638
CAO VĂN SƠN - 2020600313
ĐẶNG QUANG TRUNG - 2020601369
NGUYỄN ANH TÚ - 2020600692
NGUYỄN XUÂN TRƯỜNG - 2020601349

LỜI CẢM ƠN

Để hoàn thiện được đề tài Bài tập lớn môn Trí tuệ nhân tạo, chúng em xin gửi lời cảm ơn chân thành đến trường Đại học Công nghiệp Hà Nội, khoa Công nghệ thông tin đã tạo điều kiện cho chúng em được học tập. Chúng em xin gửi lời cảm ơn chân thành đến các thầy cô trong khoa Công nghệ thông tin đã truyền đạt cho chúng em rất nhiều kiến thức trong quá trình học tập tại trường. Đặc biệt chúng em xin chân thành cảm ơn đến thầy giáo ***TS.Trần Thanh Huân***. Trong suốt quá trình làm bài tập lớn thầy luôn giúp đỡ, hướng dẫn tận tình, truyền đạt kiến thức và kinh nghiệm của mình để chúng em hoàn thành đề tài này.

Chúng em đã cố gắng hoàn thiện báo cáo bài tập lớn tốt nhất nhưng không thể tránh được những thiếu sót. Chúng em rất mong nhận được sự góp ý của các thầy cô và các bạn để báo cáo này của chúng em được hoàn thiện hơn. Chúng em xin chân thành cảm ơn!

LỜI MỞ 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. Có những thuật toán tìm kiếm "sơ đẳng" không có thông tin, đây là những phương pháp đơn giản và trực quan, trong khi đó các 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.

Để áp dụng được các giải thuật tìm kiếm, ta cần chuyển không gian tìm kiếm về dạng đồ thị. Với dạng đồ thị ta sẽ nắm bắt những mối liên hệ, những ảnh hưởng giữa các trạng thái của bài toán một cách nhanh chóng và ngắn gọn. Trong phạm vi bài báo cáo, chúng em xin trình bày ba thuật tìm kiếm toán cơ bản và tiêu biểu với lý thuyết đồ thị đó là: Tìm kiếm theo chiều rộng, Tìm kiếm theo chiều sâu và Tìm kiếm A*. Qua đó, chúng em sẽ áp dụng giải thuật tìm kiếm A* để giải bài toán 8 puzzle, một bài toán quen thuộc với những người lập trình, chúng em sẽ đưa ra cơ chế của thuật toán, ưu nhược điểm cũng như độ phức tạp của những thuật toán trên. Bài báo cáo gồm các nội dung chính sau:

MỤC LỤC

MỞ ĐẦU

Tên đề tài:

"Xây dựng chương trình giải bài toán trò chơi 8 số theo thuật toán chiều sâu, chiều rộng, sâu lặp dần và A* (C++)"

Lý do chọn đề tài:

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. Có những thuật toán tìm kiếm "sơ đẳng" không có thông tin, đây là những phương pháp đơn giản và trực quan, trong khi đó các 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.

Mục đích của đề tài:

Trình bày về 4 thuật toán chúng em tìm hiểu, nêu ra cơ chế và ưu nhược điểm của mỗi thuật toán, áp dụng vào thực tế (trò chơi 8 số)

Bốc cục đề tài:

Nội dung đề tài được trình bày trong 3 chương:

Chương 1: Tổng quan về trí tuệ nhân tạo. Giới thiệu trò chơi 8 số

Chương 2: Thuật toán chiều rộng, chiều sâu, sâu lặp dần và A*

Chương 3: Ưu điểm, nhược điểm và ứng dụng của thuật toán

Chương 4: Tổng kết

CHƯƠNG 1: TỔNG QUAN VỀ TRÍ TUỆ NHÂN TẠO GIỚI THIỆU TRÒ CHƠI 8 SỐ

1.1. TỔNG QUAN VỀ TRÍ TUỆ NHÂN TẠO

1.1.1. Tổng quan

Trong lĩnh vực Công nghệ thông tin, Trí tuệ nhân tạo (TTNT) cũng có thể hiểu là "thông minh nhân tạo", tức là sự thông minh của máy móc do con người tạo ra, đặc biệt tạo ra cho máy tính, robot, hay các máy móc có các thành phần tính toán điện tử. TTNT là một ngành mới, nhưng phát triển rất mạnh mẽ và đem lại nhiều kết quả to lớn.

Mùa hè 1956, tại hội thảo ở Darmouth John McCarthy đã đưa ra thuật ngữ trí tuệ nhân tạo (Artificial Intelligence - AI). Mốc thời gian này được xem là thời điểm ra đời thực sự của lĩnh vực nghiên cứu TTNT. TTNT là một lĩnh vực nghiên cứu của khoa học máy tính và khoa học tính toán nói chung. Có nhiều quan điểm khác nhau về TTNT. Do đó có nhiều định nghĩa khác nhau về lĩnh vực này.

Sau đây là một số định nghĩa: "Sự nghiên cứu các năng lực trí tuệ thông qua việc sử dụng các mô hình tính toán" (Charniak và McDormott, 1985). "Nghệ thuật tạo ra các máy thực hiện các chức năng đòi hỏi sự thông minh khi được thực hiện bởi con người" (Kurweil, 1990). "Lĩnh vực nghiên cứu tìm cách giải thích và mô phỏng các hành vi thông minh trong thuật ngữ các quá trình tính toán" (Schalkoff, 1990). "Sự nghiên cứu các tính toán để có thể nhận thức, lập luận và hành động" (Winston, 1992). "Một nhánh của khoa học máy tính liên quan đến sự tự động hóa các hành vi thông minh" (Luger and Stubblefield, 1993). "TTNT là sự nghiên cứu thiết kế các tác nhân thông minh" (Poole, Mackworth and Goebel, 1998). Trí tuệ nhân tạo là một nhánh của khoa học và công nghệ liên quan đến việc làm cho máy tính có những năng lực của trí tuệ cong người, tiêu biểu như các khả năng biết suy nghĩ và lập luận để giải quyết vấn đề, biết giao tiếp do hiểu ngôn ngữ và tiếng nói, biết học và tự thích nghi,….

Mong muốn làm cho máy có những khả năng của trí thông minh con người dã có từ nhiều thế kỷ trước, tuy nhiên TTNT chỉ xuất hiện khi con người sang tạo ra máy tính điện tử. Alan Turing – nhà toán học lỗi lạc người Anh, người được xem là cha đẻ của Tin học do đưa ra cách hình thức hóa các khái niệm thuật toán và tính toán trên máy Turing – một mô hình máy trừu tượng mô tả bản chất việc xử lý các ký hiệu hình thức - có đóng góp quan trọng và thú vị cho TTNT vào năm 1950, gọi là phép thử Turing.

Theo Turing: *"Trí tuệ là những gì có thể đánh giá được thông qua các trắc nghiệm thông minh"*

Phép thử Turing là một cách để trả lời câu hỏi "máy tính có biết nghĩ không?". Alan Turing đề xuất bộ kiểm thử (Turing test): Trong trắc nghiệm này, một máy tính và một người tham gia trắc nghiệm được đặt vào trong các căn phòng cách biệt với một người thứ hai (người thẩm vấn). Người thẩm vấn không biết được chính xác đối tượng nào là người hay máy tính, và cũng chỉ có thể giao tiếp với hai đối tượng đó thông qua các phương tiện kỹ thuật như một thiết bị soạn thảo văn bản, hay thiết bị đầu cuối. Người thẩm vấn có nhiệm vụ phân biệt người với máy tính bằng cách chỉ dựa trên những câu trả lời của họ đối với những câu hỏi được truyền qua thiết bị liên lạc này. Trong trường hợp nếu người thẩm vấn không thể phân biệt được máy tính với người thì khi đó theo Turing máy tính này có thể được xem là thông minh.

Khái niệm trí tuệ đưa ra trong từ điển bách khoa toàn thư: Trí tuệ là khả năng: Phản ứng một cách thích hợp những tình huống mới thông qua hiệu chỉnh hành vi một cách thích đáng. Hiểu rõ những mối liên hệ qua lại của các sự kiện của thế giới bên ngoài nhằm đưa ra những hành động phù hợp đạt tới một mục đích nào đó. Hiện nay nhiều nhà nghiên cứu quan niệm rằng, TTNT là lĩnh vực nghiên cứu sự thiết kế các tác nhân thông minh (intellegent agent). Tác nhân thông minh là bất cứ cái gì tồn tại trong môi trường và hành động một cách thông minh Môi trường Tác nhân thông minh Các thông tin đến từ môi trường Các hành động Hình 1.2. Mô hình tác nhân thông minh Theo M.Minskey: "Trí tuệ nhân tạo mô phỏng bằng máy tính để thí nghiệm một mô hình nào đó ". TTNT là một ngành của khoa học máy tính - nghiên cứu xử lý thông tin bằng máy tính, do đó TTNT đặt ra mục tiêu nghiên cứu: làm thế nào thể hiện được các hành vi thông minh bằng thuật toán, rồi nghiên cứu các phương pháp cài đặt các chương trình có thể thực hiện được các hành vi thông minh bằng thuật toán, tiếp theo chúng ta cần chỉ ra tính hiệu quả, tính khả thi của thuật toán thực hiện một nhiệm vụ, và đưa ra các phương pháp cài đặt.

Mục tiêu của ngành TTNT: Nhằm tạo ra các máy tính có khả năng nhận thức, suy luận và phản ứng. Xây dựng TTNT là tìm cách biểu diễn tri thức và phát hiện tri thức từ các thông tin có sẵn để đưa vào trong máy tính. Để máy tính có các khái niệm nhận thức, suy luận, phản ứng thì ta cần phải cung cấp tri thức cho nó

Trí tuệ nhân tạo nghiên cứu kỹ thuật làm cho máy tính có thể "suy nghĩ một cách thông minh" và mô phỏng quá trình suy nghĩ của con người khi đưa ra những quyết định, lời giải.

Trên cơ sở đó, ta có thể thiết kế các chương trình cho máy tính để giải quyết bài toán. Sự ra đời và phát triển của TTNT đã tạo ra một bước nhảy vọt về chất trong kỹ thuật và kỹ nghệ xử lý thông tin. Trí tuệ nhân tạo chính là cơ sở của công nghệ xử lý thông tin mới, độc lập với công nghệ xử lý thông tin truyền thống dựa trên văn bản giấy tờ. Điều này được thể hiện qua các mặt sau: - Nhờ những công cụ hình thức hoá (các mô hình logic ngôn ngữ, logic mờ,...), các tri thức thủ tục và tri thức mô tả có thể biểu diễn được trong máy.

Do vậy quá trình giải bài toán được thực hiện hiệu quả hơn. - Mô hình logic ngôn ngữ đã mở rộng khả năng ứng dụng của máy tính trong lĩnh vực đòi hỏi tri thức chuyên gia ở trình độ cao, rất khó như: y học, sinh học, địa lý, tự động hóa. - Một số phần mềm trí tuệ nhân tạo thể hiện tính thích nghi và tính mềm dẻo đối với các lớp bài toán thuộc nhiều lĩnh vực khác nhau. - Khi máy tính được trang bị các phần mềm trí tuệ nhân tạo, việc sử dụng mạng sẽ cho phép giải quyết những bài toán cỡ lớn và phân tán.

So sánh kỹ thuật lập trình truyền thống và kỹ thuật xử lý tri thức trong TTNT:

Chương trình truyền thống	**Kỹ thuật TTNT**
Xử lý dữ liệu	Xử lý tri thức
Bản chất chương trình là tính thuật toán, xử lý theo các thuật toán	Bản chất chương trình là lập luận, xử lý theo các thuật giải heuristics
Xử lý tuần tự theo lô	Xử lý theo chế độ tương tác
Xử lý thông tin chính xác đầy đủ	Xử lý được các thông tin không chắc chắn, không chính xác
Chương trình = Cấu trúc dữ liệu + Giải thuật	TTNT = Tri thức + Suy diễn
Không giải thích trong quá trình thực hiện	Có thể giải thích hành vi hệ thống trong quá trình thực hiện

1.1.2. Lịch sử phát triển của trí tuệ nhân tạo

Lịch sử của TTNT cho thấy ngành khoa học này có nhiều kết quả đáng ghi nhận. Theo các mốc phát triển, người ta thấy TTNT được sinh ra từ những năm 50 với các sự kiện sau: Turing được coi là người khai sinh ngành TTNT bởi phát hiện của ông về máy tính có thể lưu trữ chương trình và dữ liệu. Tháng 8/1956 J.Mc Carthy, M. Minsky, A. Newell, Shannon. Simon,… đưa ra khái niêm "trí tuệ nhân tạo". Vào khoảng năm 1960 tại Đại học MIT (Masachusets Institure of Technology) ngôn ngữ LISP ra đời, phù hợp với các nhu cầu xử lý đặc trưng của trí tuệ nhân tạo - đó là ngôn ngữ lập trình đầu tiên dùng cho trí tuệ nhân tạo.

Thuật ngữ TTNT được dùng đầu tiên vào năm 1961 cũng tại MIT. Những năm 60 là giai đoạn lạc quan cao độ về khả năng làm cho máy tính biết suy nghĩ. Trong giai đoạn này người ta đã được chứng kiến máy chơi cờ đầu tiên và các chương trình chứng minh định lý tự động. Cụ thể:

+) 1961: Chương trình tính tích phân bất định 1963: Các chương trình heuristics: Chương trình chứng minh các định lý hình học không gian có tên là "tương tự", chương trình chơi cờ của Samuel.

+) 1964: Chương trình giải phương trình đại số sơ cấp, chương trình trợ giúp ELIZA (có khả năng làm việc giống như một chuyên gia phân tích tâm lý).

+) 1966: Chương trình phân tích và tổng hợp tiếng nói 1968: Chương trình điều khiển người máy (Robot) theo đồ án "Mắt – tay", chương trình học nói. Vào những năm 60, do giới hạn khả năng của các thiết bị, bộ nhớ và đặc biệt là yếu tố thời gian thực hiện nên có sự khó khăn trong việc tổng quát hoá các kết quả cụ thể vào trong một chương trình mềm dẻo thông minh. Vào những năm 70, máy tính với bộ nhớ lớn và tốc độ tính toán nhanh nhưng các phương pháp tiếp cận TTNT cũ vẫn thất bại do sự bùng nổ tổ hợp trong quá trình tìm kiếm lời giải các bài toán đặt ra.

+) Vào cuối những năm 70, một vài kết quả như xử lý ngôn ngữ tự nhiên, biểu diễn tri thức và giải quyết vấn đề. Những kết quả đó đã tạo điều kiện cho sản phẩm thương mại đầu tiên của TTNT ra đời đó là Hệ chuyên gia, được đem áp dụng trong các lĩnh vực khác nhau (Hệ chuyên gia là một phần mềm máy tính chứa các thông tin và tri thức về một lĩnh vực cụ thể nào đó, có khả năng giải quyết những yêu cầu của người sử dụng trong một mức độ nào đó, ở một trình độ như một chuyên gia con người có kinh nghiệm khá lâu năm).

Hệ chuyên gia thay thế con người / trợ giúp con người ra quyết định.

Những năm 90 cho đến nay, các nghiên cứu nhằm vào cài đặt thành phần thông minh trong các hệ thống thông tin, gọi chung là cài đặt TTNT, làm rõ hơn các ngành của khoa học TTNT và tiến hành các nghiên cứu mới, đặc biệt là nghiên cứu về cơ chế suy lý, về các mô hình tương tác. Các nghiên cứu về AI phân tán, mạng nơron nhân tạo, logic mờ, thuật giải di truyền, khai phá dữ liệu, web ngữ nghĩa, tin sinh học, mạng xã hội,…

1.2. GIỚI THIỆU TRÒ CHƠI 8 SỐ

1.2.1. Giới thiệu bài toán

Bài toán 8-puzzle (hay còn gọi là 8 số) là một bài toán quen thuộc với những người bắt đầu tiếp cận với môn Trí tuệ nhân tạo. 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 toàn 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 như 2 hình dưới:

1	2	3
8		4
7	6	5

Trạng thái đích 1

Hoặc

	1	2
3	4	5
6	7	8

Trạng thái đích 2

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).

1.2.2. Xác định trạng thái đích

Có những trạng thái của bảng số không thể chuyển về trạng thái đích 1 mà chỉ chuyển về trạng thái 2 hoặc ngược lại. Người ta chứng minh, để có thể xác định trạng thái đích, thì ta phải kiểm tra trạng thái đầu bằng cách như sau:

Ta xét lần lượt từ trên xuống dưới, từ trái sang phải, với mỗi ô số đang xét (giả sử là ô thứ i), ta kiểm tra xem phía sau có bao nhiêu ô số có giá trị nhỏ hơn ô đó.

- Sau đó ta tính tổng $\mathbf{N = n_1 + n_2 + n_3 + n_4 + n_5 + n_6 + n_7 + n_8}$. Nếu:
  - N mod 2 = 1
    - ⇨ Ta xác định được trạng thái đích 1 như hình dưới

1	2	3
8		4
7	6	5

  - N mod 2 = 0

⇨ Ta xác định được trạng thái đích 2 như hình dưới

	1	2
3	4	5
6	7	8

Ví dụ: Cho trạng thái đầu sau:

4	7	5
6		3
2	1	8

Xét ô thứ 1 có giá trị 4: Phía sau có 3 ô nhỏ hơn (3, 2, 1) => $n_1 = 3$

Xét ô thứ 2 có giá trị 7: Phía sau có 5 ô nhỏ hơn (5, 6, 3, 2, 1) => $n_2 = 5$

Xét ô thứ 3 có giá trị 5: Phía sau có 3 ô nhỏ hơn (3, 2, 1) => $n_3 = 3$

Xét ô thứ 4 có giá trị 6: Phía sau có 3 ô nhỏ hơn (3, 2, 1) => $n_4 = 3$

Xét ô thứ 5 có giá trị 3: Phía sau có 2 ô nhỏ hơn (2, 1) => $n_5 = 2$

Xét ô thứ 6 có giá trị 2: Phía sau có 1 ô nhỏ hơn (1) => $n_6 = 1$

Xét ô thứ 7 có giá trị 1: Phía sau không còn ô nào nhỏ hơn => $n_7 = 0$

Xét ô thứ 8 có giá trị 8: Phía sau không còn ô nào => $n_8 = 0$

$$\mathbf{N = 3 + 5 + 3 + 3 + 2 + 1 + 0 + 0 = 17}$$

Ta có: **17 mod 2 = 1**

⇨ **Ta xác định được <u>trạng thái đích 1</u>**

Hàm **checkTarget()** xác định trạng thái đích (C++):

```
void checkTarget()
{
   int sum = 0;

   for (int p = 0; p < 9; p++)
   {
      int row = p / 3;
      int col = p % 3;
      int counter = puzzle[row][col];

      for (int i = 0; i <= 2; i++)
         for (int j = 0; j <= 2; j++)
            if (row < i && puzzle[i][j] < counter && puzzle[i][j] != 0)
               sum++;
            else if (row == i && col < j && puzzle[i][j] < counter
                     puzzle[i][j] != 0)
               sum++;
   }

   target = sum % 2;
}
```

- ***target = 1***
  ⇨ Trạng thái đích 1
- ***target = 0***
  ⇨ Trạng thái đích 2

1.2.3. Kiểm tra trạng thái đích

Nếu ***target = 1***, ta kiểm tra xem các giá trị trong ô đã đúng trạng thái đích 1 chưa, ngược lại ta kiểm tra trạng thái đích 2

Hàm **checkFinish()** xác định trạng thái đích (C++):

```
bool checkFinish()
   {
      if (target == 1)
      {
         for (int i = 0; i <= 2; i++)
            if (arr[0][i] != i + 1 || arr[2][i] != 7 - i)
               return false;

         return arr[1][0] != 8 || arr[1][2] != 4 ? false : true;
      }
      else
      {
         for (int i = 0; i <= 2; i++)
            if (arr[0][i] != i || arr[1][i] != i + 3
                || arr[2][i] != i + 6)
               return false;

         return true;
      }
}
```

Hàm **checkFinish()** sẽ trả về ***true*** nếu đã đúng trạng thái đích 1 hoặc 2

Ngược lại trả về ***false***.

** Trong phạm vi bài tập lớn, chúng em tìm hiểu về 4 phương pháp tìm kiếm lời giải cho bài toán 8 số, đó là: Tìm kiếm theo chiều rộng, tìm kiếm theo chiều sâu, tìm kiếm theo sâu lặp dần và tìm kiếm A*. Ba phương pháp đầu là những phương pháp tìm kiếm không có thông tin, A* là phương pháp tìm kiếm có thông tin. Cụ thể mỗi phương pháp sẽ được trình bày ngay sau đây.*

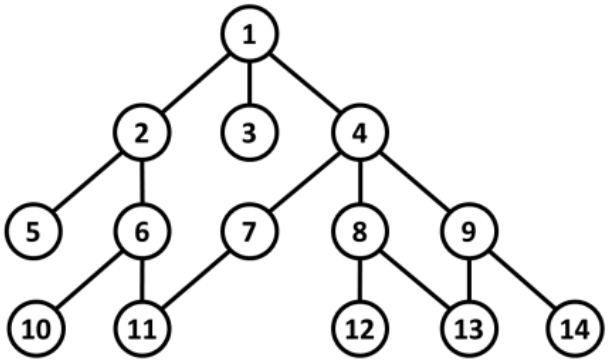
CHƯƠNG 2: CÁC THUẬT TOÁN CHIỀU RỘNG, CHIỀU SÂU, SÂU LẶP DẦN VÀ A*

Thuật toán tìm kiếm theo chiều sâu và chiều rộng là hai thuật toán tìm kiếm mù phổ biến, thường được sử dụng trong lý thuyết đồ thị. Chúng ta sẽ đi vào từng thuật toán từ tư tưởng của thuật toán cho tới giả mã và bước đi trong thuật toán để làm rõ hơn cách thức hoạt động của thuật toán. Trong quá trình tìm hiểu ta sẽ nhận thấy chúng có nhiều điểm tương đồng trong cách thực hiện, nhưng cách tổ chức thì khác nhau. Với mỗi thuật toán ta sẽ đưa ra ưu nhược điểm của chúng để có thể sử dụng chúng phù hợp hơn theo những yêu cầu riêng của bài toán đầu vào.

2.1. THUẬT TOÁN TÌM KIẾM THEO CHIỀU RỘNG (BFS)

Thuật toán **duyệt đồ thị ưu tiên chiều rộng** *(Breadth-first search - BFS)* là một trong những thuật toán tìm kiếm cơ bản và thiết yếu trên đồ thị. Mà trong đó, những đỉnh nào gần đỉnh xuất phát hơn sẽ được duyệt trước.

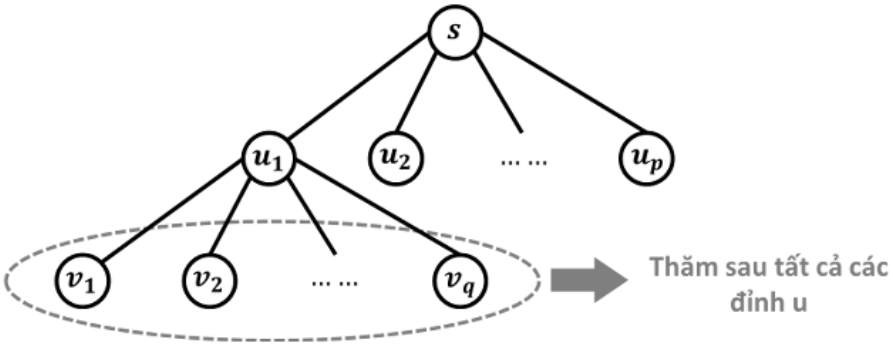
Ứng dụng của BFS có thể giúp ta giải quyết tốt một số bài toán trong thời gian và không gian **tối thiểu**. Đặc biệt là bài toán tìm kiếm đường đi ngắn nhất từ một đỉnh gốc tới tất cả các đỉnh khác. Trong đồ thị không có trọng số hoặc tất cả trọng số bằng nhau, thuật toán sẽ luôn trả ra đường đi ngắn nhất có thể. Ngoài ra, thuật toán này còn được dùng để tìm các thành phần liên thông của đồ thị, hoặc kiểm tra đồ thị hai phía, …



Thứ tự thăm các đỉnh của BFS

2.1.1. Tư tưởng

- Đầu tiên ta thăm đỉnh nguồn ***s***.
- Việc thăm đỉnh ***s*** sẽ phát sinh thứ tự thăm các đỉnh $(u_1, u_2, \ldots, u_p)$ kề với ***s*** (những đỉnh gần ***s*** nhất).
- Tiếp theo, ta thăm đỉnh u_1, khi thăm đỉnh u_1 sẽ lại phát sinh yêu cầu thăm những đỉnh $(v_1, v_2, \ldots, v_q)$ kề với u_1. Nhưng rõ ràng những đỉnh ***v*** này "xa" ***s*** hơn những đỉnh ***u*** nên chúng chỉ được thăm khi tất cả những đỉnh ***u*** đều đã được thăm. Tức là thứ tự thăm các đỉnh sẽ là: $s, u_1, u_2, \ldots, u_p, v_1, v_2, \ldots, v_q, \ldots$



Thuật toán tìm kiếm theo chiều rộng sử dụng một danh sách để chứa những đỉnh đang "chờ" thăm. Tại mỗi bước, ta thăm một đỉnh đầu danh sách, loại nó ra khỏi danh sách và cho những đỉnh kề với nó chưa được thăm xếp hàng vào cuối danh sách. Thuật toán sẽ kết thúc khi danh sách rỗng.

2.1.2. Thuật toán

Thuật toán sử dụng một cấu trúc dữ liệu ngăn xếp kiểu "trạng thái trò chơi" để lưu trữ thông tin trung gian thu được trong quá trình tìm kiếm:

Bước 1: Chèn trạng thái đầu vào ngăn xếp.

Bước 2: Duyệt hết các trạng thái trong ngăn xếp và kiểm tra nó.

- Nếu gặp trạng thái đích, dừng quá trình tìm kiếm và trả về kết quả.
- Nếu không gặp trạng thái đích, chèn tất cả các trạng thái di chuyển được từ các trạng thái đã duyệt vào ngăn xếp, xóa các trạng thái đã duyệt.

Bước 3: Quay về bước 2.

2.1.3. Cài đặt chương trình (C++)

1. Tạo 1 lớp ***state*** biểu thị cho mỗi trạng thái của puzzle, chuỗi *way* của trạng thái hiện tại sẽ lưu đường đi từ đích đến nó.

```
class state
{
public:
    int arr[3][3], x, y;
    string way;
    facing canFace;

    bool canMoveLeft()
    {
        return canFace != LEFT && y > 0;
    }
    bool canMoveRight()
    {
        return canFace != RIGHT && y < 2;
    }
    bool canMoveUp()
    {
        return canFace != UP && x > 0;
    }
    bool canMoveDown()
    {
        return canFace != DOWN && x < 2;
    }

    void moveLeft()
    {
        swap(arr[x][y], arr[x][y - 1]);
        y--;
        canFace = RIGHT;
        way += "l";
    }
    void moveRight()
    {
        swap(arr[x][y], arr[x][y + 1]);
        y++;
        canFace = LEFT;
        way += "r";
    }
    void moveUp()
    {
        swap(arr[x][y], arr[x - 1][y]);
        x--;
        canFace = DOWN;
```

```
        way += "u";
    }
    void moveDown()
    {
        swap(arr[x][y], arr[x + 1][y]);
        x++;
        canFace = UP;
        way += "d";
    }
};
```

2. Cài đặt thuật toán

```
bool check = checkFinish();
string way = "";
state stt(puzzle, "", STAY, posX, posY);
vector<state> vt;
vt.push_back(stt);

while (!check)
    {
        vector<state> open;

        for (int i = 0; i < vt.size(); i++)
        {
            numOfState++;

            if (vt.at(i).checkFinish())
            {
                way = vt.at(i).way;
                check = true;
                break;
            }
            else
            {
                if (vt.at(i).canMoveLeft())
                {
                    state stt = vt.at(i);
                    stt.moveLeft();
                    open.push_back(stt);
                }
                if (vt.at(i).canMoveRight())
                {
                    state stt = vt.at(i);
                    stt.moveRight();
                    open.push_back(stt);
                }
                if (vt.at(i).canMoveUp())
                {
                    state stt = vt.at(i);
```

```
                stt.moveUp();
                open.push_back(stt);
            }
            if (vt.at(i).canMoveDown())
            {
                state stt = vt.at(i);
                stt.moveDown();
                open.push_back(stt);
            }
        }
    }

    vt.clear();

    for (int i = 0; i < open.size(); i++)
        vt.push_back(open.at(i));
}
```

Sau khi tìm được trạng thái đích, biến ***check*** nhận giá trị ***true,*** lúc này chuỗi ***way*** sẽ lưu lại đường đi từ *trạng thái đầu* đến *trạng thái đích* và thoát khỏi vòng lặp.

2.1.4. Nhận xét

Tìm được đích, đường đi ngắn nhất, tốn nhiều "chi phí".

Độ phức tạp về thời gian của thuật toán BFS được biểu diễn dưới dạng O(V + E), trong đó V là số nút và E là số cạnh.

Độ phức tạp không gian của thuật toán là O(V).

2.2. THUẬT TOÁN TÌM KIẾM THEO CHIỀU SÂU (DFS)

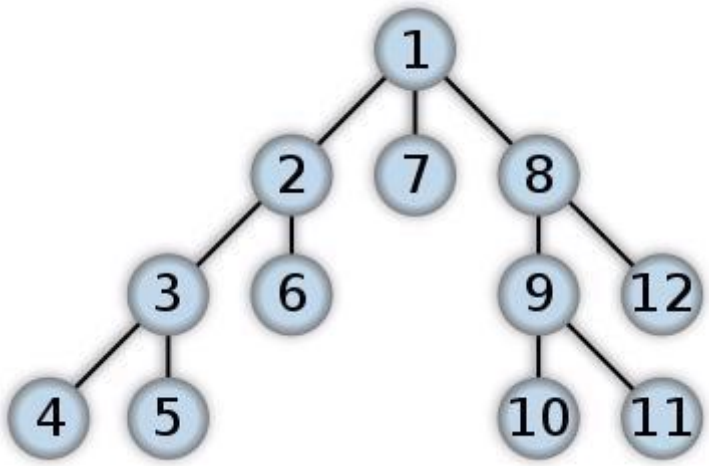
Tìm kiếm ưu tiên chiều sâu hay **tìm kiếm theo chiều sâu** (*Depth-first search*) là một thuật toán khởi đầu tại gốc (hoặc chọn một đỉnh nào đó coi như gốc) và phát triển xa nhất có thể theo mỗi nhánh.

2.2.1. Tư tưởng

- Đầu tiên ta thăm đỉnh nguồn ***s***.
- Việc thăm đỉnh ***s*** sẽ phát sinh thứ tự thăm các đỉnh $(u_1, u_2, \ldots, u_p)$ kề với ***s*** (những đỉnh gần ***s*** nhất).
- Tiếp theo, ta thăm đỉnh u_1, khi thăm đỉnh u_1 sẽ lại phát sinh yêu cầu thăm những đỉnh $(v_1, v_2, \ldots, v_q)$ kề với u_1. Nhưng không giống như ***BFS***, ta sẽ tiếp tục thăm đỉnh

v_1... Giả dụ độ sâu tối đa là 2, tức từ **s** đến u_1 đến v_1 mà chưa tìm được đích, thuật toán sẽ không phát sinh thêm đỉnh con của v_1 mà sẽ quay ngược lên u_1 và duyệt tiếp các con của u_1 là $v_1, v_2, \ldots, v_q$.

- Tức thứ tự thăm các đỉnh khi độ sâu là 2 là: $s, u_1, v_1, v_2, \ldots, v_q, u_2, t_1, t_2, \ldots, t_r, u_3, \ldots$
- Nếu đã duyệt hết các đỉnh với độ sâu tối đa mà chưa tìm được đích, thuật toán thất bại



Thứ tự thăm các đỉnh với độ sâu tối đa là 3 (DFS)

2.2.2. Thuật toán

Thuật toán sử dụng một cấu trúc dữ liệu ngăn xếp kiểu "trạng thái trò chơi" để lưu trữ thông tin trung gian thu được trong quá trình tìm kiếm:

Bước 1: Chèn trạng thái đầu vào ngăn xếp.

Bước 2: Duyệt trạng thái cao nhất trong ngăn xếp và kiểm tra nó.

- Nếu là trạng thái đích, dừng quá trình tìm kiếm và trả về kết quả.
- Nếu không phải trạng thái đích và chưa đạt độ sâu tối đa, chèn tất cả các trạng thái di chuyển được từ các trạng thái đang duyệt vào ngăn xếp, xóa trạng thái đang duyệt.
- Nếu không phải trạng thái đích và đã đạt độ sâu tối đa, xóa trạng thái đang duyệt.

Bước 3: Nếu ngăn xếp không rỗng, quay về bước 2. Nếu ngăn xếp rỗng, thông báo thuật toán thất bại, không tìm được đích.

2.2.3. Cài đặt chương trình (C++)

1. Tạo 1 lớp ***state*** biểu thị cho mỗi trạng thái của puzzle, chuỗi *way* của trạng thái hiện tại sẽ lưu đường đi từ đích đến nó, *f* là độ sâu.

```
class state
{
public:
    int arr[3][3], x, y, f;
    string way;
    facing canFace;

    bool canMoveLeft()
    {
        return canFace != LEFT && y > 0 && f < deep;
    }
    bool canMoveRight()
    {
        return canFace != RIGHT && y < 2 && f < deep;
    }
    bool canMoveUp()
    {
        return canFace != UP && x > 0 && f < deep;
    }
    bool canMoveDown()
    {
        return canFace != DOWN && x < 2 && f < deep;
    }

    void moveLeft()
    {
        swap(arr[x][y], arr[x][y - 1]);
        y--;
        canFace = RIGHT;
        way += "l";
        f++;
    }
    void moveRight()
    {
        swap(arr[x][y], arr[x][y + 1]);
        y++;
        canFace = LEFT;
        way += "r";
        f++;
    }
    void moveUp()
    {
        swap(arr[x][y], arr[x - 1][y]);
        x--;
        canFace = DOWN;
        way += "u";
        f++;
    }
    void moveDown()
    {
```

```
        swap(arr[x][y], arr[x + 1][y]);
        x++;
        canFace = UP;
        way += "d";
        f++;
    }
};
```

2. Cài đặt thuật toán

```
bool check = checkFinish();
string way = "";
state stt(puzzle, "", STAY, posX, posY, 0);
vector<state> vt;
vt.push_back(stt);

while (!check && vt.size() != 0)
    {
        vector<state> open;

        int i = vt.size() - 1;
        if (vt.at(i).checkFinish())
        {
            way = vt.at(i).way;
            check = true;
            break;
        }
        else
        {
            if (vt.at(i).canMoveUp())
            {
                state stt = vt.at(i);
                stt.moveUp();
                open.push_back(stt);
            }
            if (vt.at(i).canMoveDown())
            {
                state stt = vt.at(i);
                stt.moveDown();
                open.push_back(stt);
            }
            if (vt.at(i).canMoveRight())
            {
                state stt = vt.at(i);
                stt.moveRight();
                open.push_back(stt);
            }
            if (vt.at(i).canMoveLeft())
            {
                state stt = vt.at(i);
```

```
                stt.moveLeft();
                open.push_back(stt);
            }
        }

        vt.pop_back();

        for (int i = 0; i < open.size(); i++)
            vt.push_back(open.at(i));
}
```

Sau khi tìm được trạng thái đích, biến ***check*** nhận giá trị ***true,*** lúc này chuỗi ***way*** sẽ lưu lại đường đi từ *trạng thái đầu* đến *trạng thái đích* và thoát khỏi vòng lặp.

2.2.4. Nhận xét

Không chắc chắn tìm được đích, "chi phí" thấp.

Độ phức tạp về thời gian của thuật toán DFS được biểu diễn dưới dạng O(V+E), trong đó V là số nút và E là số cạnh.

Độ phức tạp về không gian của thuật toán là O(V).

2.3. THUẬT TOÁN TÌM KẾM THEO CHIỀU SÂU LẶP DẦN (IDS)

Tìm kiếm theo chiều sâu lặp dần **Iterative Deepening Search** *(IDS)* là một chiến lược tìm kiếm không gian / đồ thị trạng thái trong đó phiên bản tìm kiếm theo chiều sâu được chạy lặp đi lặp lại với độ sâu ngày càng tăng cho đến khi mục tiêu được tìm thấy. *IDS* tối ưu giống như tìm kiếm theo chiều rộng, nhưng sử dụng ít bộ nhớ hơn nhiều. Ở mỗi lần lặp lại, nó truy cập các nút trong cây tìm kiếm theo thứ tự giống như tìm kiếm theo chiều sâu, thứ tự tích lũy các nút được truy cập đầu tiên có hiệu quả hơn theo chiều rộng.

2.3.1. Tư tưởng

- Đầu tiên ta thăm đỉnh nguồn ***s***.
- Việc thăm đỉnh ***s*** sẽ phát sinh thứ tự thăm các đỉnh $(u_1, u_2, \ldots, u_p)$ kề với ***s*** (những đỉnh gần ***s*** nhất).
- Tiếp theo, ta thăm đỉnh u_1, khi thăm đỉnh u_1 sẽ lại phát sinh yêu cầu thăm những đỉnh $(v_1, v_2, \ldots, v_q)$ kề với u_1. Nhưng không giống như ***BFS***, ta sẽ tiếp tục thăm đỉnh

v_1... Giả dụ độ sâu tối đa là 2, tức từ ***s*** đến u_1 đến v_1 mà chưa tìm được đích, thuật toán sẽ không phát sinh thêm đỉnh con của v_1 mà sẽ quay ngược lên u_1 và duyệt tiếp các con của u_1 là $v_1, v_2, \ldots, v_q$.

- Tức thứ tự thăm các đỉnh khi độ sâu là 3 là: $s, u_1, v_1, v_2, \ldots, v_q, u_2, t_1, t_2, \ldots, t_r, u_3$,...
- Nếu đã duyệt hết các đỉnh với độ sâu tối đa mà chưa tìm được đích, thuật toán tự động tăng độ sâu và quay lại đỉnh v_1, sinh ra các đỉnh con và duyệt các đỉnh con mới đó.

2.3.2. Thuật toán

Thuật toán sử dụng một cấu trúc dữ liệu ngăn xếp kiểu "trạng thái trò chơi" để lưu trữ thông tin trung gian thu được trong quá trình tìm kiếm:

Bước 1: Chèn trạng thái đầu vào ngăn xếp.

Bước 2: Duyệt trạng thái cao nhất trong ngăn xếp và kiểm tra nó.

- Nếu là trạng thái đích, dừng quá trình tìm kiếm và trả về kết quả.
- Nếu không phải trạng thái đích và chưa đạt độ sâu tối đa, chèn tất cả các trạng thái di chuyển được từ các trạng thái đang duyệt vào ngăn xếp, xóa trạng thái đang duyệt khỏi ngăn xếp.
- Nếu không phải trạng thái đích và đã đạt độ sâu tối đa, lưu trạng thái đang duyệt vào 1 ô nhớ tạm, xóa trạng thái đang duyệt khỏi ngăn xếp.

Bước 3: Nếu ngăn xếp không rỗng, quay về bước 2. Nếu ngăn xếp rỗng, lấy các trạng thái từ ô nhớ tạm, tặng độ sâu, quay về bước 2.

2.3.3. Cài đặt chương trình (C++)

1. Tạo 1 lớp ***state*** biểu thị cho mỗi trạng thái của puzzle, chuỗi *way* của trạng thái hiện tại sẽ lưu đường đi từ đích đến nó, *f* là độ sâu, *deep* là độ sâu tối đa.

```
class state
{
public:
    int arr[3][3], x, y, f;
    string way;
    facing canFace;

    bool canMoveLeft()
    {
        return canFace != LEFT && y > 0 && f <= deep;
    }
    bool canMoveRight()
    {
        return canFace != RIGHT && y < 2 && f <= deep;
```

```
    }
    bool canMoveUp()
    {
        return canFace != UP && x > 0 && f <= deep;
    }
    bool canMoveDown()
    {
        return canFace != DOWN && x < 2 && f <= deep;
    }

    void moveLeft()
    {
        swap(arr[x][y], arr[x][y - 1]);
        y--;
        canFace = RIGHT;
        way += "l";
        f++;
    }
    void moveRight()
    {
        swap(arr[x][y], arr[x][y + 1]);
        y++;
        canFace = LEFT;
        way += "r";
        f++;
    }
    void moveUp()
    {
        swap(arr[x][y], arr[x - 1][y]);
        x--;
        canFace = DOWN;
        way += "u";
        f++;
    }
    void moveDown()
    {
        swap(arr[x][y], arr[x + 1][y]);
        x++;
        canFace = UP;
        way += "d";
        f++;
    }
};
```

2. Cài đặt thuật toán

```
bool check = checkFinish();
string way = "";
state stt(puzzle, "", STAY, posX, posY, 0);
vector<state> vt;
```

```
vt.push_back(stt);

while (!check)
    {
        if (vt.size() == 0)
        {
            for (int i = 0; i < vtBackTrack.size(); i++)
                vt.push_back(vtBackTrack.at(i));

            deep += 10;
        }

        vector<state> open;
        int i = vt.size() - 1;
        if (vt.at(i).checkFinish())
        {
            way = vt.at(i).way;
            check = true;
            break;
        }
        else
        {
            if (vt.at(i).canMoveUp())
            {
                state stt = vt.at(i);
                stt.moveUp();
                open.push_back(stt);
                if (stt.backTrack())
                    vtBackTrack.push_back(stt);
            }
            if (vt.at(i).canMoveDown())
            {
                state stt = vt.at(i);
                stt.moveDown();
                open.push_back(stt);
                if (stt.backTrack())
                    vtBackTrack.push_back(stt);
            }
            if (vt.at(i).canMoveRight())
            {
                state stt = vt.at(i);
                stt.moveRight();
                open.push_back(stt);
                if (stt.backTrack())
                    vtBackTrack.push_back(stt);
            }
            if (vt.at(i).canMoveLeft())
            {
                state stt = vt.at(i);
                stt.moveLeft();
```

```
                open.push_back(stt);
                if (stt.backTrack())
                    vtBackTrack.push_back(stt);
            }
        }

        vt.pop_back();

        for (int i = 0; i < open.size(); i++)
            vt.push_back(open.at(i));
}
```

Sau khi tìm được trạng thái đích, biến ***check*** nhận giá trị ***true,*** lúc này chuỗi ***way*** sẽ lưu lại đường đi từ *trạng thái đầu* đến *trạng thái đích* và thoát khỏi vòng lặp.

2.3.4. Nhận xét

Tìm được đích, "chi phí" thấp nếu đích ở sâu.

Độ phức tạp về thời gian của thuật toán IDS được biểu diễn dưới dạng O(V+E), trong đó V là số nút và E là số cạnh.

Độ phức tạp về không gian của thuật toán là O(V).

2.4. THUẬT TOÁN TÌM KIẾM A*

A* (đọc là *A sao*) là 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*).

2.4.1. Tư tưởng

- Đầu tiên ta thăm đỉnh nguồn ***s***.
- Việc thăm đỉnh ***s*** sẽ phát sinh thứ tự thăm các đỉnh $(u_1, u_2, \ldots, u_p)$ kề với ***s*** (những đỉnh gần ***s*** nhất).
- Tiếp theo, ta sẽ so sánh chi phí của mỗi đỉnh ***u*** rồi thăm trạng thái có chi phí nhỏ nhất, giả dụ đỉnh u_2 nhỏ nhất, khi thăm đỉnh u_2 sẽ lại phát sinh yêu cầu thăm những đỉnh $(v_1, v_2, \ldots, v_q)$ kề với u_2. Ta tiếp tục so sánh chi phí của các đỉnh ***v*** rồi chọn thăm đỉnh có chi phí nhỏ nhất... Duyệt cho đến khi tìm được đích.

2.4.2. Thuật toán

Thuật toán sử dụng một cấu trúc dữ liệu ngăn xếp kiểu "trạng thái trò chơi" để lưu trữ thông tin trung gian thu được trong quá trình tìm kiếm:

Bước 1: Chèn trạng thái đầu vào ngăn xếp.

Bước 2: Kiểm tra các trạng thái con mới, duyệt trạng thái có chi phí ít nhất.

- Nếu là trạng thái đích, dừng quá trình tìm kiếm và trả về kết quả.
- Nếu không phải trạng thái đích và chưa đạt chi phí tối đa, chèn tất cả các trạng thái di chuyển được từ các trạng thái đang duyệt vào ngăn xếp, xóa trạng thái đang duyệt khỏi ngăn xếp.
- Nếu không phải trạng thái đích và đã đạt chi phí tối đa, xóa trạng thái đang duyệt khỏi ngăn xếp.

Bước 3: Nếu ngăn xếp không rỗng, quay về bước 2. Nếu ngăn xếp rỗng, thông báo chi phí tối đa quá ít, thử lại với chi phí cao hơn.

2.4.3. Cài đặt chương trình (C++)

1. Tạo 1 lớp ***state*** biểu thị cho mỗi trạng thái của puzzle, chuỗi *way* của trạng thái hiện tại sẽ lưu đường đi từ đích đến nó, *f* là độ sâu, *cost* là chi phí tối đa.

+) Hàm **herStic()** để tính toán chi phí của trạng thái đang duyệt

```
class state
{
public:
    int arr[3][3], x, y, f;
    string way;
    facing canFace;

    bool canMoveLeft()
    {
        return canFace != LEFT && y > 0 && cost > herStic();
    }
    bool canMoveRight()
    {
        return canFace != RIGHT && y < 2 && cost > herStic();
    }
    bool canMoveUp()
    {
        return canFace != UP && x > 0 && cost > herStic();
    }
    bool canMoveDown()
```

```
    {
        return canFace != DOWN && x < 2 && cost > herStic();
    }

    int herStic()
    {
        int sum = 0;
        if (target == 1)
        {
            if (arr[0][0] != 1)
                sum++;
            if (arr[0][1] != 2)
                sum++;
            if (arr[0][2] != 3)
                sum++;
            if (arr[1][0] != 8)
                sum++;
            if (arr[1][2] != 4)
                sum++;
            if (arr[2][0] != 7)
                sum++;
            if (arr[2][1] != 6)
                sum++;
            if (arr[2][2] != 5)
                sum++;
        }
        else
        {
            if (arr[0][1] != 1)
                sum++;
            if (arr[0][2] != 2)
                sum++;
            if (arr[1][0] != 3)
                sum++;
            if (arr[1][1] != 4)
                sum++;
            if (arr[1][2] != 5)
                sum++;
            if (arr[2][0] != 6)
                sum++;
            if (arr[2][1] != 7)
                sum++;
            if (arr[2][2] != 8)
                sum++;
        }

        return sum + f;
    }

    void moveLeft()
```

```
    {
        swap(arr[x][y], arr[x][y - 1]);
        y--;
        canFace = RIGHT;
        way += "l";
        f++;
    }
    void moveRight()
    {
        swap(arr[x][y], arr[x][y + 1]);
        y++;
        canFace = LEFT;
        way += "r";
        f++;
    }
    void moveUp()
    {
        swap(arr[x][y], arr[x - 1][y]);
        x--;
        canFace = DOWN;
        way += "u";
        f++;
    }
    void moveDown()
    {
        swap(arr[x][y], arr[x + 1][y]);
        x++;
        canFace = UP;
        way += "d";
        f++;
    }
};
```

2. Cài đặt thuật toán

```
while (!check && vt.size() != 0)
    {
        vector<state> open;

        int i = vt.size() - 1;
        if (vt.at(i).checkFinish())
        {
            way = vt.at(i).way;
            check = true;
            break;
        }
        else
        {
            if (vt.at(i).canMoveUp())
            {
```

```
                state stt = vt.at(i);
                stt.moveUp();
                open.push_back(stt);
            }
            if (vt.at(i).canMoveDown())
            {
                state stt = vt.at(i);
                stt.moveDown();
                open.push_back(stt);
            }
            if (vt.at(i).canMoveRight())
            {
                state stt = vt.at(i);
                stt.moveRight();
                open.push_back(stt);
            }
            if (vt.at(i).canMoveLeft())
            {
                state stt = vt.at(i);
                stt.moveLeft();
                open.push_back(stt);
            }
        }

        vt.pop_back();

        for (int i = 0; i < open.size(); i++)
            for (int j = i + 1; j < open.size(); j++)
                if (open.at(i).herStic() <= open.at(j).herStic())
                    swap(open.at(i), open.at(j));

        for (int i = 0; i < open.size(); i++)
            if (open.at(i).herStic() == open.at(open.size() -
1).herStic())
                vt.push_back(open.at(i));
}
```

Sau khi tìm được trạng thái đích, biến ***check*** nhận giá trị ***true,*** lúc này chuỗi ***way*** sẽ lưu lại đường đi từ *trạng thái đầu* đến *trạng thái đích* và thoát khỏi vòng lặp.

2.4.4. Nhận xét

Tìm được đích, "chi phí" thấp nhất.

Độ phức tạp thời gian của A* phụ thuộc vào đánh giá heuristic. Trong trường hợp xấu nhất, số nút được mở rộng theo hàm mũ của độ dài lời giải, nhưng nó sẽ là hàm đa thức khi hàm heuristic *h* thỏa mãn điều kiện sau:

$$| h(x) - h^*(x) | \leq O(\log h^*(x))$$

Trong đó h^* là heuristic tối ưu, nghĩa là hàm cho kết quả là chi phí chính xác để đi từ x tới đích. Nói cách khác, sai số của h không nên tăng nhanh hơn lôgarit của "heuristic hoàn hảo" - hàm trả về khoảng cách thực từ x tới đích.

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

CHƯƠNG 3: ƯU ĐIỂM, NHƯỢC ĐIỂM VÀ ỨNG DỤNG CỦA CÁC THUẬT TOÁN

3.1. ƯU ĐIỂM, NHƯỢC ĐIỂM CỦA CÁC THUẬT TOÁN

3.1.1. Thuật toán tìm kiếm theo chiều rộng (BFS)

- ***Ưu điểm***
  - Kỹ thuật tìm kiếm rộng là kỹ thuật vét cạn không gian trạng thái bài toán vì vậy sẽ tìm được lời giải nếu có.
  - Đường đi tìm được thỏa mãn đi qua ít đỉnh nhất.
- ***Nhược điểm***
  - Tìm kiếm lời giải theo thuật toán đã định trước, do vậy tìm kiếm một cách máy móc; khi không có thông tin bổ trợ cho quá trình tìm kiếm, không nhận ra ngay lời giải.
  - Không phù hợp với không gian bài toán có kích thước lớn. Đối với loại bài toán này thì phương pháp tìm kiếm chiều rộng đối diện với các khó khăn về nhu cầu:
    - Cần nhiều bộ nhớ theo số nút cần lưu trữ.
    - Cần nhiều công sức xử lý các nút, nhất là khi các nhánh cây dài, số nút tăng.
    - Dễ thực hiện các thao tác không thích hợp, thừa, đưa đến việc tăng đáng kể số nút phải xử lý.
  - Không hiệu quả nếu lời giải ở sâu. Phương pháp này không phù hợp cho trường hợp có nhiều đường dẫn đến kết quả nhưng đều sâu.
  - Giao tiếp với người dùng không thân thiện. Do duyệt qua tất cả các nút, việc tìm kiếm không tập trung vào một chủ đề.

3.1.2. Thuật toán tìm kiếm theo chiều sâu (DFS)

- ***Ưu điểm***

- Nếu bài toán có lời giải, phương pháp tìm kiếm theo chiều sâu đảm bảo tìm ra lời giải.
- Kỹ thuật tìm kiếm sâu tập trung vào đích, con người cảm thấy hài lòng khi các câu hỏi tập trung vào vấn đề chính.
- Do cách tìm của kỹ thuật này, nếu lời giải ở rất sâu, kỹ thuật sâu sẽ tiết kiệm thời gian

- ***Nhược điểm***
  - Tìm sâu khai thác không gian bài toán để tìm lời giải theo thuật toán đơn giản một cách cứng nhắc. Trong quá trình tìm nó không có thông tin nào để phát hiện lời giải. Nếu nút con ban đầu không thích hợp có thể không dẫn tới đích của bài toán.
  - Không phù hợp với không gian bài toán lớn, kỹ thuật tìm kiếm sâu có thể không đi đến lời giải trong khoảng thời gian vừa phải (nếu cố định thời gian).

3.1.3. Thuật toán tìm kiếm theo chiều sâu lặp dần (IDS)

- ***Ưu điểm***
  - Giống với thuật toán tìm kiếm theo chiều sâu thì với bài toán có lời giải thì chắc chắn sẽ tìm ra lời giải
  - Độ phức tạp của thuật toán chỉ là O(n) với n được định nghĩa là độ sâu của mục tiêu
- ***Nhược điểm***
  - Lãng phí thời gian và tài nguyên máy tính để có thể đi đến từng nút
  - Thuật toán IDS có thể không thành công nếu như thuật toán BFS không thành công. Khi cần tìm nhiều câu trả lời từ IDS, nó sẽ trả lại các nút thành công và đường dẫn đến nút ngay cả khi cần được tìm lại sau nhiều lần lặp lại. Để dừng giới hạn độ sâu thì cần không được tăng độ sâu thêm nữa.

3.1.4. Thuật toán tìm kiếm A*

- ***Ưu điểm***
  - Nếu bài toán có lời giải thì cũng giống như 2 thuật toán tìm kiếm trước, A* có tính "đầy đủ" theo – có nghĩa nó sẽ luôn tìm thấy lời giải nếu bài toán đó có lời giải
  - A* có tính tối ưu nhanh chóng tìm đến lời giải với sự định hướng của hàm Heuristic. Chính vì thế mà người ta thường nói A* chính là thuật giải tiêu biểu cho Heuristic.
- ***Nhược điểm***
  - Muốn A* tối ưu thì hàm h(n) phải có tính chấp nhận được – tức là nó không bao giờ đánh giá cao hơn chi phí nhỏ nhất thực sự của việc đi tới đích

- A* rất linh động nhưng vẫn gặp một khuyết điểm cơ bản - giống như chiến lược tìm kiếm chiều rộng - đó là tốn khá nhiều bộ nhớ để lưu lại những trạng thái đã đi qua.

3.2. ỨNG DỤNG CỦA THUẬT TOÁN

Với một lập trình viên, hiểu rõ về các thuật toán vừa nêu là rất quan trọng để có thể áp dụng thực tiễn. Nếu ta viết một phần mềm, ta sẽ phải đánh giá được phần mềm đó sẽ hoạt động nhanh chậm ra sao. Những đánh giá như vậy sẽ kém chính xác hơn nhiều nếu ta không có hiểu biết về thời gian chạy hay độ phức tạp. Thêm nữa, hiểu biết về thuật toán của những gì ta đang làm sẽ giúp ta dự đoán những trường hợp đặc biệt khiến phần mềm chạy chậm đi hay xảy ra lỗi.

Tất nhiên, ta sẽ thường xuyên gặp những bài toán chưa được nghiên cứu trước đó. Lúc này ta phải tự nghĩ ra thuật mới, hoặc áp dụng thuật cũ một cách sáng tạo hơn. Càng có kiến thức về thuật toán, ta càng có khả năng giải quyết thành công vấn đề. Trong nhiều trường hợp, một vấn đề mới có thể được đưa về một vấn đề cũ hơn mà không cần quá nhiều sức lực, với điều kiện ta phải có kiến thức đủ sâu về vấn đề cũ này

Có rất nhiều ví dụ cho thấy các vấn đề thực tế đòi hỏi hiểu biết về thuật toán. Gần như mọi thứ bạn đang làm với máy tính được dựa trên một thuật toán nào đó mà có người phải rất vất vả mới tìm ra.Và dưới đây là một vài ứng dụng:

- Dự báo thời tiết với độ chính xác cực cao
    - Dù có sử dụng những thiết bị đo khí tượng tốt nhất thì việc đưa ra một bản dự báo thời tiết với độ chính xác gần như tuyệt đối vẫn là một công việc không khả thi. Nhưng với sự xuất hiện của máy tính lượng tử thì việc xây dựng một mô hình thời tiết cho một khu vực hay toàn cầu là hoàn toàn có thể làm được.
    - Với những thuật toán phù hợp có thể đưa ra những dự đoán về một cơn bão bao gồm thời gian nó bắt đầu, những nơi nào nó có thể đi qua ngay trước khi nó được sinh ra. Giám đốc kỹ thuật của Google, Hartmut Neven, cho biết ngoài việc dự báo những hiện tượng thời tiết cụ thể thì máy tính có thể dự báo được những xu hướng thời tiết của tương lai trong vài chục năm tới, đặc biệt là hậu quả của việc trái đất nóng lên.
- Phát triển những loại thuốc mới hiệu quả hơn
    - Để đưa một phương thuốc đi vào cuộc sống hiện đại thì phải trải qua hàng nghìn cuộc thử nghiệm, hàng chục năm phát triển và thậm chí tiêu tốn hàng triệu USD. Đấy là còn chưa tính đến những trường hợp thất bại trước khi được đưa vào sản xuất.
    - Nhưng mọi chuyện sẽ chỉ là quá khứ khi mà những cỗ máy hiện nay có thể đưa ra hàng nghìn kiểu kết hợp phân tử có thể xảy ra của loại thuốc đang

nghiên cứu, từ đó những nhà khoa học có thể nhanh chóng tìm ra phương án tối ưu nhất làm không phải tốn thời gian như trước.

- Giảm thiểu ùn tắc giao thông
    - Thông thường việc điều tiết giao thông bằng những máy tính hiện nay thường chỉ đáp ứng được yêu cầu khi không xảy ra ùn tắc, thậm chí chúng vẫn cần có sự can thiệp của con người để thực hiện công việc vất vả này.
    - Với sự có mặt của các thuật toán thì mọi hoạt động trên đường phố đều được xử lý một cách chớp nhoáng, ngoài ra những cỗ máy này có thể đưa ra lộ trình di chuyển của các phương tiện giao thông trong những giờ cao điểm ở mọi điểm nóng trong thành phố. Tất nhiên, hệ thống này là hoàn toàn tự động.
- Thám hiểm không gian vũ trụ xa hơn
    - Với sự hỗ trợ của kính thiên văn Kepler, các nhà thiên văn học đã phát hiện được hơn 2000 hành tinh bên ngoài hệ mặt trời nhưng những thông tin mà chúng ta nhận được thường đã là quá khứ của những hành tinh này.
    - Vì vậy, yêu cầu cần có một hệ thống máy tính có thể xử lý những thông tin này và đưa nó về dạng thông tin ban đầu để các nhà khoa học có thể biết được liệu sự sống có từng tồn tại trên những hành tinh này hay không, đây là một trong những điểm mấu chốt của ngành thiên văn học hiện đại.
- Máy móc có khả năng học hỏi
    - Những cỗ máy có khả năng học hỏi sẽ là một trong những điểm mạnh nhất của trí tuệ nhân tạo. Bằng cách áp dụng những thuật toán thì bộ xử lý của những robot tương lai sẽ có khả năng tự thu thập thông tin, xử lý chúng và tự thích nghi với những tình huống có thể xảy ra dựa vào những thông tin nhận được. Những cuộc thám hiểm không gian hay thăm dò những khu vực nguy hiểm sẽ rất cần điều này.

Số thuật toán khác nhau mà con người học cũng nhiều như số bài toán khác nhau mà ta cần giải. Thế nhưng khả năng cao là bài toán bạn đang cố gắng giải có liên quan đến một bài toán khác, theo một cách nào đó.

Có một vốn hiểu biết sâu và rộng về các thuật toán sẽ giúp bạn chọn lựa được hướng đi đúng và áp dụng thành công. Khi nghiên cứu thuật toán, nhiều bài toán nhìn có vẻ không thực tế, nhưng kĩ năng giải quyết đó lại được áp dụng trong những bài toán mà chúng ta gặp hàng ngày.

CHƯƠNG 4: TỔNG KẾT

Như vậy, mỗi một phương pháp đều có ưu và nhược điểm riêng, tuy nhiên giải thuật A* áp dụng cho bài toán 8-puzzle là hiệu quả hơn cả. Giải thuật A* là thuật toán tìm kiếm trong đồ thị có thông tin phản hồi, sử dụng đánh giá heuristic để xếp loại từng nút và duyệt nút theo hàm đánh giá này. Với các dạng bài toán tìm kiếm trên không gian trạng thái, có hai trường hợp cần tới heuristic:

- Những vấn đề không thể có nghiệm chính xác do các mệnh đề không phát biểu chặt chẽ hay do thiếu dữ liệu để khẳng định kết quả.
- Những vấn đề có nghiệm chính xác nhưng chi phí tính toán để tìm ra nghiệm là quá lớn (dẫn đến bùng nổ tổ hợp).

⇨ **Heuristic giúp ta tìm kiếm đạt kết quả với chi phí thấp hơn.**

Tài liệu tham khảo

https://en.wikipedia.org/wiki/Breadth-first_search
https://en.wikipedia.org/wiki/Depth-first_search
https://en.wikipedia.org/wiki/Iterative_deepening_depth-first_search
https://en.wikipedia.org/wiki/A*_search_algorithm
http://aptech.vn/kien-thuc-tin-hoc/n-puzzle-tim-hieu-ve-cach-giai-bai-toan.html