ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA KHOA HỌC MÁY TÍNH

MÔN HỌC

NGUYÊN LÝ VÀ PHƯƠNG PHÁP LẬP TRÌNH

GIẢNG VIÊN :
NGUYỄN TUẤN ĐĂNG

SINH VIÊN THỰC HIỆN:
HÀ KIỆT HÙNG - 220101031

04, 2023

LỜI CẢM ƠN

Trong suốt thời gian từ khi bắt đầu môn học, em đã nhận được sự quan tâm, hướng dẫn, giúp đỡ của giáo viên hướng dẫn, các thầy cô đã giảng dạy cũng như sự ủng hộ của gia đình và bạn bè xung quanh. Ngoài những kiến thức mà chúng em nhận được, các thầy cô, anh chị khóa trên còn truyền đạt những lời khuyên chân thành, kinh nghiệm quý báu trong suốt thời gian học tập và rèn luyện tại trường, đó không chỉ là nền tảng cho quá trình nghiên cứu khóa luận mà còn là hành trang quý báu để nhóm bước vào đời một cách vững chắc và tự tin hơn. Với lòng biết ơn sâu sắc và chân thành nhất, nhóm xin gửi đến quý Thầy Cô đã và đang công tác, giảng dạy ở Khoa Khoa học Máy Tính – Trường Đại Học Công Nghệ Thông Tin đã cùng với những tri thức và tâm huyết của mình để truyền đạt vốn kiến thức và kỹ năng quý báu cho em trong suốt thời gian học tập và rèn luyện tại trường. Em xin gửi lời cảm ơn chân thành và tri ân sâu sắc tới thầy Nguyễn Tuấn Đăng, người đã trực tiếp tận tình hướng dẫn em trong suốt quá trình thực hiện đồ án. Không chỉ gợi ý và tận tâm hướng dẫn chúng em trong quá trình tìm hiểu, đọc tài liệu và lựa chọn đề tài, qua từng buổi học trên lớp cũng như những buổi nói chuyện, thảo luận về lĩnh vực sáng tạo trong nghiên cứu khoa học, thầy còn tận tình chỉ bảo nhóm những kĩ năng phân tích, khai thác tài liệu để có được những tư liệu phù hợp với nội dung của khóa luận. Hơn nữa, thầy còn rất nhiệt tình trong việc đốc thúc quá trình viết khóa luận, đọc và đưa ra những nhận xét, góp ý để nhóm có thể hoàn thành khoá luận tốt nghiệp một cách tốt nhất. Nếu không có những lời hướng dẫn, dạy bảo của thầy thì em nghĩ bài thu hoạch rất khó có thể hoàn thiện được. Một lần nữa, em xin chân thành cảm ơn thầy. Bài thu hoạch được thực hiện trong khoảng thời gian 3 tuần. Bước đầu tìm hiểu về

lĩnh vực sáng tạo trong nghiên cứu khoa học, kiến thức, trình độ lý luận cũng như kinh nghiệm thực tiễn của em còn hạn chế và còn nhiều bỡ ngỡ. Do vậy, không tránh khỏi những thiếu sót là điều chắc chắn, em rất mong nhận được những sự chỉ bảo, ý kiến đóng góp quý báu của quý Thầy Cô và các bạn học cùng lớp để em có điều kiện được bổ sung, nâng cao kiến thức của mình trong lĩnh vực này, giúp hoàn thiện hơn và tích luỹ thêm cho bản thân nhiều kinh nghiệm, phục vụ tốt hơn cho công việc thực tế sau này. Em xin kính chúc quý Thầy Cô trong Khoa Khoa học Máy tính và Thầy Nguyễn Tuấn Đăng thật dồi dào sức khỏe, niềm tin để tiếp tục thực hiện sứ mệnh cao đẹp của mình là truyền đạt kiến thức cho thế hệ mai sau, luôn gặt hái được nhiều thành công tốt đẹp trong sự nghiệp trồng người của mình. Cuối cùng, em xin được gửi đến ba mẹ, gia đình và bạn bè lời cảm ơn, tri ân chân thành và lòng biết ơn sâu sắc vì những sự động viên, ủng hộ, giúp đỡ và cổ vũ tinh thần cho em trong suốt quá trình gian nan và vất vả này. Em xin chân thành cảm ơn!

Mục lục

Chương 1

Tổng quan

1.1 Giới thiệu bài toán

Bài toán 8-puzzle là một bài toán trò chơi trí tuệ thường được biết với cái tên là "Xếp hình". 8-puzzle vì nó bao gồm một bảng gồm 9 ô vuông (3x3) và 8 ô vuông chứa các con số từ 1 đến 8, còn lại là 1 ô trống. Mục tiêu của bài toán là di chuyển các ô vuông để sắp xếp các số từ 1 đến 8 vào vị trí đúng, sao cho ô trống nằm ở vị trí cuối cùng. Trong mỗi lần di chuyển, chỉ có thể đổi chỗ ô trống với một ô vuông kề cạnh nó. Bài toán 8-puzzle là một bài toán NP-khó, nghĩa là không có giải thuật đơn giản để giải quyết nó trong thời gian hợp lý đối với các bảng kích thước lớn. Nhiều giải thuật đã được đề xuất để giải quyết bài toán này, bao gồm giải thuật tìm kiếm theo chiều rộng (BFS), tìm kiếm theo chiều sâu (DFS), thuật toán A* và thuật toán heuristic.

1.2 Điều kiện để hoàn thành

Mỗi ô trong 8- puzzle sẽ có tối đa 4 cách di chuyển để chuyển từ trạng thái này sang trạng thái khác gồm: trái, phải, lên và xuống. Để hoàn thành được bài toán này chúng ta có một quy tắc như sau: 1/ Theo quy tắc từ trái sang phải, từ trên xuống dưới. 2/ Ở mỗi ô số duyệt đến, đếm xem có bao nhiêu ô số có giá trị nhỏ hơn giá trị hiện tại đang duyệt. 3/ Duyệt cho đến khi hết 8 ô và tính tổng số ô có giá trị nhỏ hơn ứng với mỗi ô trên Puzzle. 4/ Nếu giá trị tổng là số chẵn

thì trạng thái hiện tại có thể chuyển về trạng thái đích.

Ví dụ: Xét ô thứ nhất có giá trị 2: Phía sau có 1 ô nhỏ hơn (1) => n1 = 1

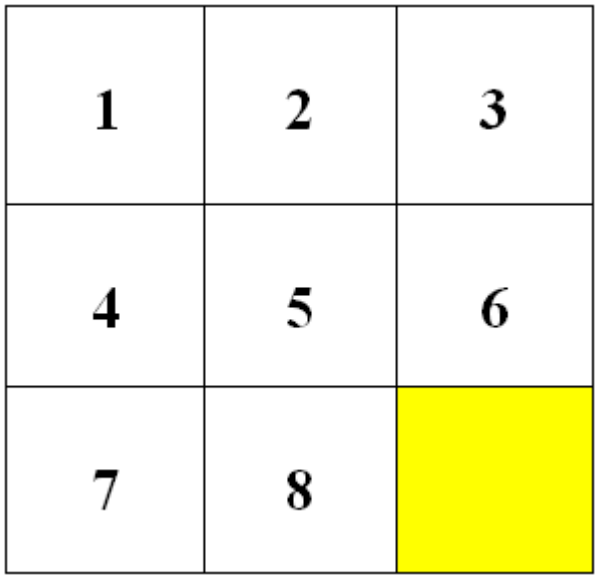
2		**4**
7	**8**	**3**
6	**1**	**5**

Hình 1.1: Trạng thái ví dụ

Xét ô thứ ba có giá trị 4: Phía sau có 2 ô nhỏ hơn (3,1) => n3 = 2
Xét ô thứ tư có giá trị 7: Phía sau có 3 ô nhỏ hơn (3,6,1,5) => n4 = 4
Xét ô thứ năm có giá trị 8: Phía sau có 4 ô nhỏ hơn (3,6,1,5) => n5 = 4
Xét ô thứ sáu có giá trị 3: Phía sau có 3 ô nhỏ hơn (6,1,5) => n6 = 3
Xét ô thứ bảy có giá trị 6: Phía sau có 2 ô nhỏ hơn (1,5) => n7 = 2
Xét ô thứ tám có giá trị 1: Phía sau không còn ô nào nhỏ hơn => n8 = 0

Như vậy ta có công thức tính N như sau:
N = 1 + 2 + 4 + 4 + 3 + 2 + 0 = 16
Ta có 16 mod 2 = 0 => Trường hợp này bài toán có thể đưa về trạng thái đích.

Những trạng thái của bảng số mà có thể chuyển về trạng thái đích gọi là cấu hình hợp lệ, ngược lại gọi là cấu hình không hợp lệ.
Đây là kết quả mà ta mong muốn:



Hình 1.2: Trạng thái đích

Chương 2

Thuật toán A*

2.1 Giới thiệu thuật toán

Trong khoa học máy tính, 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).

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

Năm 1964, Nils Nilsson phát minh ra một 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 hiển thị 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 khi 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 là A *(A sao, A-star).

Trong khoa học máy tính, A* thuật toán tìm kiếm trong đồ thị. Thuật toán này tìm một đường đi từ nút khởi đầu đến nút đích dựa trên cách đánh giá Heuristic để tìm ra con đường tốt nhất cho thuật toán. Điểm khác biệt của A* đối với tìm kiếm theo lựa chọn tốt nhất là nó còn tính khoảng cách đã đi qua. Điều này làm cho A* đầy đủ và tối ưu, nghĩa là với thuật toán này sẽ luonon tìm thấy đường đi ngắn nhất nếu tồn tại.

2.2 Hàm ước lượng Heuristic

Heuristic là phương pháp giải quyết vấn đề dựa trên phỏng đoán, ước chừng, kinh nghiệm, trực giác để tìm ra giải pháp gần như là tốt nhất và nhanh chóng. Hàm Heuristic là hàm ứng với mỗi trạng thái hay mỗi sự lựa chọn một giá trị ý nghĩa đối với vấn đề dựa vào giá trị hàm này ta lựa chọn hành động.

2.3 Mô tả thuật toán

Giả sử n0 là trạng ban đầu và n là trạng thái hiện tại, ta có đường đi từ n0 đến n được xác định bằng hàm: f(n) = g(n) + h(n). Trong đó:
1/ g(n) là khoảng cách từ đỉnh xuất phát đến đỉnh đang xét.
2/ h(n) khoảng cách ước lượng từ đỉnh đang xét đến đích.

Áp dụng vào bài toán 8 Puzzle như sau:
Đầu vào: trạng thái ban đầu và trạng thái đích.
Đầu ra: tập dữ liệu tất cả các trạng thái từ trạng thái ban đầu đến trạng thái đích.
Điều kiện dừng thuật toán: tìm thấy kết quả trạng thái trùng với trạng thái đích.

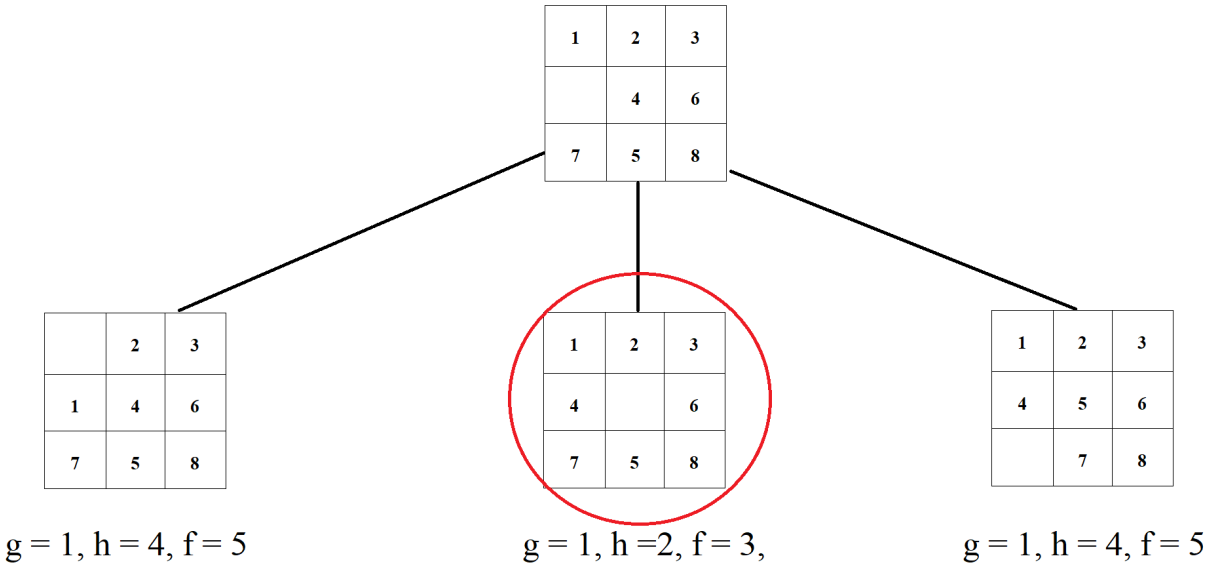
Ở đây h(n) sẽ được tính bằng tổng số vị trí còn sai so với trạng thái đích, g(n) sẽ là số bước thuật toán đã thực hiện được. Tiến hành thực bài toán như sau: B1: Tiến hành di chuyển các ô xung quanh vị trí trống. B2: Thực hiện tính toán các vị trí sai của trạng thái hiện tại so với trạng thái đích và tính f(n) theo công thức f(n) = g(n) + h(n) để chọn ra f(n) có giá trị thấp nhất. f(n) có giá trị thấp nhất sẽ là hướng giải tốt nhất.
B3: Lặp lại bước 1 cho các vị trí tìm được ở bước thứ 2. Lưu ý trạng thái đạt được phải khác với trạng thái của những lần thực hiện trước đó.

Dưới đây là ví dụ các bước chi tiết thực hiện bài toán với trạng thái ban đầu như sau:

1	**2**	**3**
	4	**6**
7	**5**	**8**

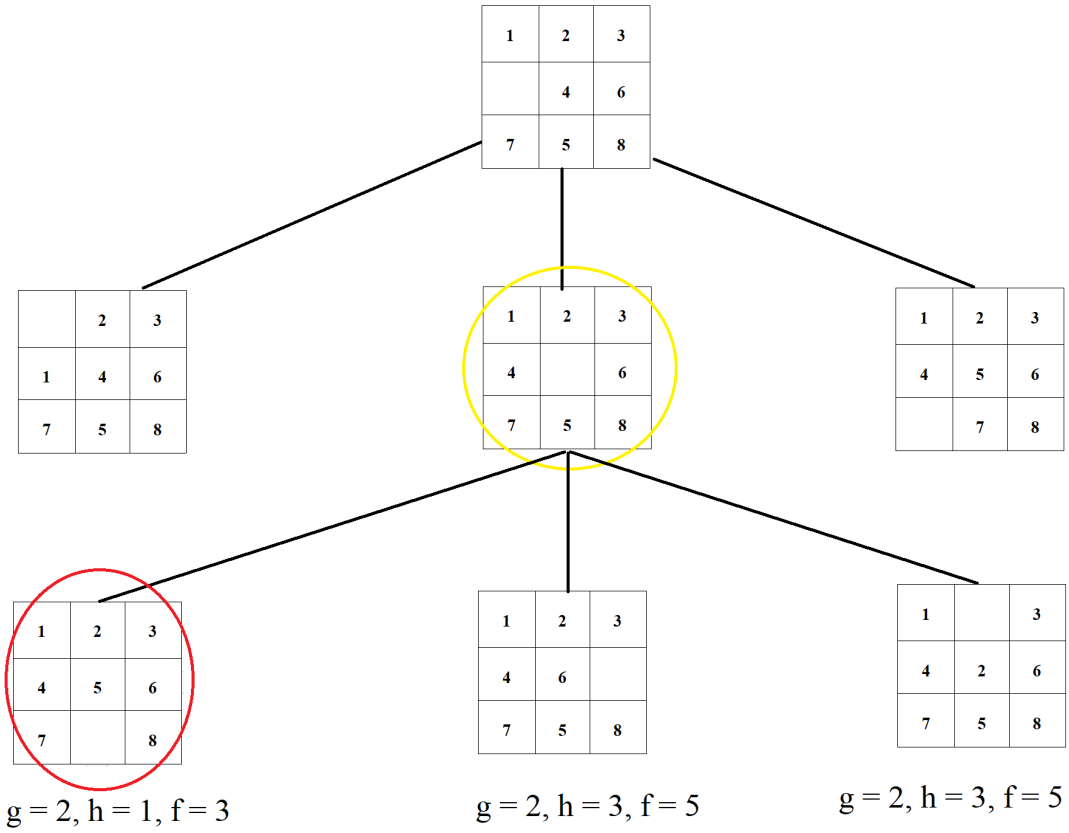
Hình 2.1: Bước 1

Bước 1: Ta có 3 cách di chuyển trong đó trạng thái ở giữa cho ta f(n) giá trị thấp nhất



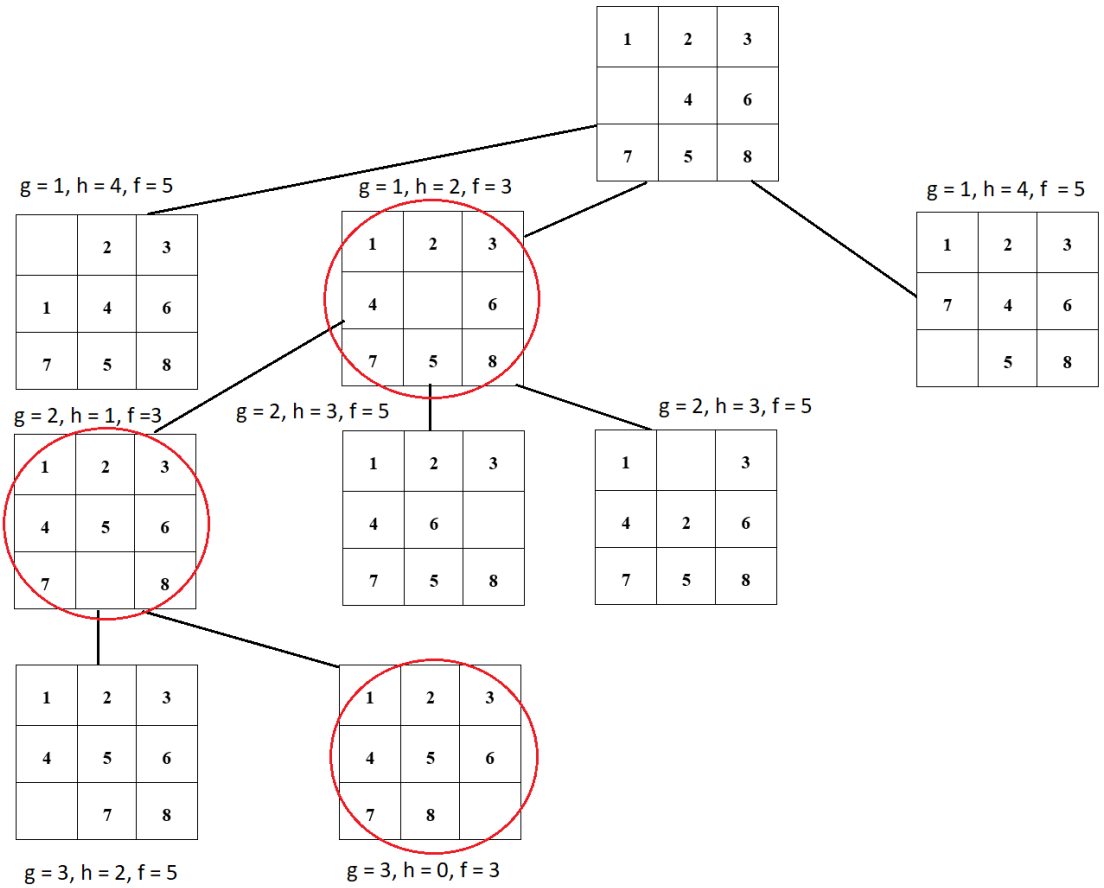
Hình 2.2: Bước 1

Bước 2: Tiếp tục tính f(n) cho các trạng thái mới tại trạng thái đã chọn ở bước 1.



Hình 2.3: Bước 2

Bước 3: Hoàn thành bài toán



Hình 2.4: Bước 3

2.4 Áp dụng thuật toán A* vào bài toán sử dụng C++

2.4.1 Khởi tạo Class và Function

Cài đặt thuật toán A* bài toán ta dùng một Class Node. Class này gồm 1 mảng 2 chiều 3x3 và các thông số khác bao gồm:

1/ Vị trí x,y nhằm xác định vị trí phần tử của mảng Node, trong đó phần tử giá trị 0 là đại diện cho ô trống trên Puzzle.
3/ Step là khoảng cách từ trạng thái xuất phát đến trạng thái hiện tại. 2/ canMove thể hiện phần tử có thể duy chuyển sang vị trí mới hay không.
5/ Cost là số bước tối đa để thực hiện cho bài toán này. Bài toán sẽ dừng lại nếu số bước thực hiện vượt quá giá trị Cost được mặc định sẵn. Ở đây chương trình đang mặc định giá trị 10000.
Bên cạnh đó, class Node còn có những Function sau:
1/ checkMoveLeft, checkMoveRight, checkMoveUp, checkMoveDown để xác định hướng mà phần tử có thể di chuyển.
2/ moveLeft, moveRight, moveUp, moveDown: hàm thực hiện đổi chỗ các phần tử liền kề với nhau.
3/ checkFinish: kiểm tra mảng đã trùng với trạng thái đích hay chưa. Nếu trùng thì bài toán sẽ kết thúc.

```
class Node
{
public:
    int arr[3][3], x, y, z, Check, Cost, defautCost;
    string Direction;
    moveWay canMove;

    Node(int a[3][3], string Direction, moveWay canMove, int x, int y, int z, int Check, int Cost);

    int defaultCost(int Check, int Value );

    bool checkMoveLeft();
    bool checkMoveRight();
    bool checkMoveUp();
    bool checkMoveDown();
    bool checkFinish();

    void moveLeft();
    void moveRight();
    void moveUp();
    void moveDown();


};
```

Hình 2.5: Khởi tạo class Node.

2.4.2 Một số hàm liên quan trong thuật toán

Hàm kiểm tra phần tử trong mảng có thể di chuyển hay không.

```
bool Node::checkMoveLeft() {
    defautCost = 0;
    int curCost = defaultCost(Check, defautCost);
    return canMove != LEFT && y > 0 && Cost > curCost;
}
bool Node::checkMoveRight() {
    defautCost = 0;
    int curCost = defaultCost(Check, defautCost);
    return canMove != RIGHT && y < 2 && Cost > curCost;
}
bool Node::checkMoveUp() {
    defautCost = 0;
    int curCost = defaultCost(Check, defautCost);
    return canMove != UP && x > 0 && Cost > curCost;
}
bool Node::checkMoveDown() {
    defautCost = 0;
    int curCost = defaultCost(Check, defautCost);
    return canMove != DOWN && x < 2 && Cost > curCost;
}

bool Node::checkFinish() {
    int counter = 0;
    if (Check == 1) {
        REP(i, 0, 2) {
            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 {
        REP(i, 0, 2) {
            if (arr[0][i] == i + 1) counter++;
            if (arr[1][i] == i + 4) counter++;
            if (arr[2][i] == i + 7) counter++;

        }

        if (counter == 8) return true;
        return false;
    }

}
```

Hình 2.6: Một số hàm kiểm tra.

Hàm di chuyển.

```
void Node::moveLeft() {
    swap(arr[x][y], arr[x][y - 1]);
    y--;
    canMove = RIGHT;
    Direction += "l";
    z++;
}

void Node::moveRight() {
    swap(arr[x][y], arr[x][y + 1]);
    y++;
    canMove = LEFT;
    Direction += "r";
    z++;
}

void Node::moveUp() {
    swap(arr[x][y], arr[x - 1][y]);
    x--;
    canMove = DOWN;
    Direction += "u";
    z++;
}

void Node::moveDown() {
    swap(arr[x][y], arr[x + 1][y]);
    x++;
    canMove = UP;
    Direction += "d";
    z++;
}
```

Hình 2.7: Hàm di chuyển

Ý tưởng và chi tiết thực hiện bài toán:
Đầu tiên, người dùng sẽ nhập dữ liệu đầu vào gồm 9 phần tử từ 0 đến 8, trong đó 0 là đại diện cho vị trí ô trống. Sau đó, chương trình sẽ kiểm tra vị trí các phần tử là lấy ra vị trí hiện tại của phần tử mang giá trị 0. Sau khi lấy được vị trí của giá trị 0, chương trình sẽ đẩy mảng khởi tạo ban đầu vào một Vector và bắt đầu thực hiện tính toán.
Tại đây, vị trí phần tử 0 sẽ được kiểm tra xem có thể di chuyển theo hướng nào (trái, phải, lên xuống). Sau đó, các phần tử sẽ tiến hành hoán đổi vị trí cho nhau, mỗi lần hoán đổi sẽ sinh ra một phần tử được đẩy vào Vector.Ứng với mỗi phần tử được sinh ra, chương trình sẽ kiểm tra trạng thái tại phần tử đó đã trùng với trạn thái đích chưa. Nếu chưa thì chương trình sẽ bắt đầu tính giá trị f(n) và lựa chọn phần tử có giá trị f(n) thấp nhất để tiếp tục thực hiện các bước hoán đổi vị trí. Thao tác này được lặp lại cho đến khi chương trình tìm ra mảng trùng với trạng thái đích hoặc số bước thực hiện vượt quá số bước quy định ban đầu.

2.4.3 Kết quả đạt được

Kết quả đạt được sau khi chạy thuật toán.

Hình 2.8: Kết quả thực hiện.

Chương 3

Thuật toán DFS (tìm kiếm theo chiều sâu)

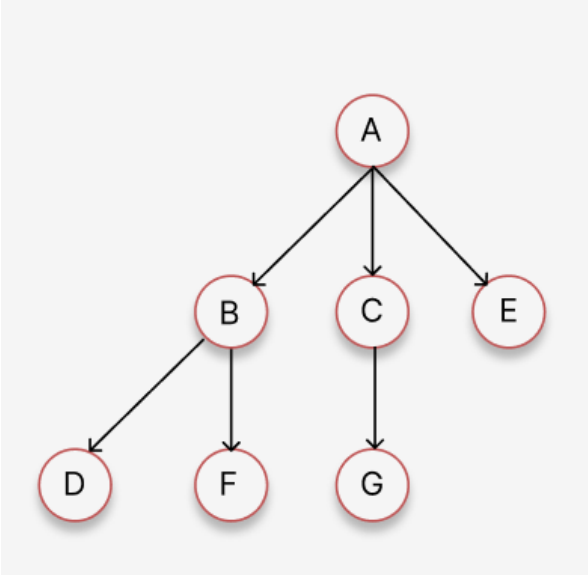
3.1 Giới thiệu thuật toán

DFS hay còn được gọi là tìm kiếm ưu tiên theo chiều sâu, tìm kiếm theo chiều sâu là một thuật toán duyệt trên một cây hoặc đồ thị. Thuật toán sẽ khởi đầu tại gốc hoặc một đỉnh bất kì (đỉnh này được xem như là gốc) và phát triển xa nhất có thể theo mỗi nhánh của gốc đó.
DFS là một dạng tìm kiếm thjoong tin không đầy đủ bởi vì quá trình tìm kiếm được phát triển tới đỉnh con đầu tiên của nút đang tìm kiếm cho tới khi gặp được đỉnh cần tìm hoặc tới một nút không có con. Khi đó giải thuật quay lui về đỉnh vừa mới tìm kiếm ở bước trước. Trong dạng không đệ quy, tất cả các đỉnh chờ được phát triển được bổ sung vào một ngăn xếp LIFO.
Xét về độ phức tạp, theo không gian DFS sẽ thấp hơn BFS nhưng sẽ tương đương nhau khi xét về thời gian.

3.2 Mô tả thuật toán

Như hình dưới đây, thuật toán DFS sẽ bắt đầu tại đỉnh A và đi theo cạnh trái và tiếp tục tìm kiếm cho đến khi hết phần nhánh cây trái rồi mới chuyển sang nhanh con bên phải. Thứ tự sau khi chạy đủ thuật toán sẽ như sau: A, B, D, F,

C, G, E.



Hình 3.1: Cây đồ thị

3.3 Áp dụng thuật toán DFS vào bài toán sử dụng Prolog

3.3.1 Khởi tạo trạng thái đích

```
% Define the goal state of the 8-puzzle.
goal_state([1,2,3, 4,5,6, 7,8,0]).
```

Hình 3.2: Khởi tạo trạng thái đích

3.3.2 Thuật toán DFS trong Prolog

```
%dfs
dfs([State|States], [], Path) :-goal(State), !, reverse([State|States], Path).
dfs([State|States], [Move|Moves], Path) :-
    moveState(State, Next, Move),
    \+member(Next, [State|States]),
    dfs([Next,State|States], Moves, Path).
```

Hình 3.3: Thuật toán DFS trong Prolog

Thuật toán trên gồm ba tham số đầu vào là 'States', 'Moves' và 'Path'. Trong đó, 'States' là các danh sách cách trạng thái đã duyệt qua. 'Moves' là danh sách các thao tác di chuyển và 'Path' là đường đi xuất phát từ trạng thái khởi tạo đến trạng thái đích.

3.3.3 Định nghĩa các hàm di chuyển

Dưới đây là định nghĩa các thao tác di chuyển trong bà toán, trong đó '*' là ô trống. Mỗi hàm moveState sẽ nhận vào 1 trạng thái sẽ trả về một trạng thái mới sau khi di chuyển.

```
%Bottom Move
moveState(state(X1,X2,X3,X4,X5,X6,*,X8,X9), state(X1,X2,X3,X4,X5,X6,X8,*,X9), botleft ).
moveState(state(X1,X2,X3,X4,X5,X6,*,X8,X9), state(X1,X2,X3,*,X5,X6,X4,X8,X9), botup).
moveState(state(X1,X2,X3,X4,X5,X6,X7,*,X9), state(X1,X2,X3,X4,X5,X6,*,X7,X9), botlef).
moveState(state(X1,X2,X3,X4,X5,X6,X7,*,X9), state(X1,X2,X3,X4,*,X6,X7,X5,X9), botup).
moveState(state(X1,X2,X3,X4,X5,X6,X7,*,X9), state(X1,X2,X3,X4,X5,X6,X7,X9,*), botright).
moveState(state(X1,X2,X3,X4,X5,X6,X7,X8,*), state(X1,X2,X3,X4,X5,*,X7,X8,X6), botup).
moveState(state(X1,X2,X3,X4,X5,X6,X7,X8,*), state(X1,X2,X3,X4,X5,X6,X7,*,X8), botleft).

%Top Move
moveState(state(*,X2,X3,X4,X5,X6,X7,X8,X9), state(X2,*,X3,X4,X5,X6,X7,X8,X9), topright).
moveState(state(*,X2,X3,X4,X5,X6,X7,X8,X9), state(X4,X2,X3,*,X5,X6,X7,X8,X9), topdown).
moveState(state(X1,*,X3,X4,X5,X6,X7,X8,X9), state(*,X1,X3,X4,X5,X6,X7,X8,X9), topleft).
moveState(state(X1,*,X3,X4,X5,X6,X7,X8,X9), state(X1,X3,*,X4,X5,X6,X7,X8,X9), topright).
moveState(state(X1,*,X3,X4,X5,X6,X7,X8,X9), state(X1,X5,X3,X4,*,X6,X7,X8,X9), topdown).
moveState(state(X1,X2,*,X4,X5,X6,X7,X8,X9), state(X1,*,X2,X4,X5,X6,X7,X8,X9), topleft).
moveState(state(X1,X2,*,X4,X5,X6,X7,X8,X9), state(X1,X2,X6,X4,X5,*,X7,X8,X9), topdown).

%Middle Move
moveState(state(X1,X2,X3,*,X5,X6,X7,X8,X9), state(*,X2,X3,X1,X5,X6,X7,X8,X9), midup).
moveState(state(X1,X2,X3,*,X5,X6,X7,X8,X9), state(X1,X2,X3,X5,*,X6,X7,X8,X9), midright).
moveState(state(X1,X2,X3,*,X5,X6,X7,X8,X9), state(X1,X2,X3,X7,X5,X6,*,X8,X9), middown).
moveState(state(X1,X2,X3,X4,*,X6,X7,X8,X9), state(X1,*,X3,X4,X2,X6,X7,X8,X9), midup).
moveState(state(X1,X2,X3,X4,*,X6,X7,X8,X9), state(X1,X2,X3,X4,X6,*,X7,X8,X9), midright).
moveState(state(X1,X2,X3,X4,*,X6,X7,X8,X9), state(X1,X2,X3,X4,X8,X6,X7,*,X9), middown).
moveState(state(X1,X2,X3,X4,*,X6,X7,X8,X9), state(X1,X2,X3,*,X4,X6,X7,X8,X9), midleft).
moveState(state(X1,X2,X3,X4,X5,*,X7,X8,X9), state(X1,X2,*,X4,X5,X3,X7,X8,X9), midup).
moveState(state(X1,X2,X3,X4,X5,*,X7,X8,X9), state(X1,X2,X3,X4,*,X5,X7,X8,X9), midleft).
moveState(state(X1,X2,X3,X4,X5,*,X7,X8,X9), state(X1,X2,X3,X4,X5,X9,X7,X8,*), middown).
```

Hình 3.4: Định nghĩa các hàm di chuyển

3.3.4 Cài đặt thuật toán và khởi chạy

```
% Define the DFS algorithm.
dfs(State, _, []) :-goal_state(State).

dfs(State, Visited, [Move|Moves]) :-move(State, NextState)
    ,\+member(NextState, Visited)
    ,dfs(NextState, [NextState|Visited], Moves)
    ,Move = NextState.

% Define function to print the path.
print_path([]).
print_path([State|States]) :-
    print_path(States),
    write(State), nl.
```

Hình 3.5: Thuật toán DFS

Thuật toán bắt đầu bằng cách gọi hàm dfs với tham số đầu vào là danh sách chứa đỉnh xuất phát và danh sách rỗng để lưu các bước đi. Nếu đỉnh xuất phát là đỉnh đích, hàm sẽ trả về danh sách chứa đỉnh xuất phát. Nếu không, hàm sẽ lấy ra tất cả các đỉnh kề với đỉnh hiện tại, đánh dấu đỉnh hiện tại là đã thăm và gọi đệ quy lại hàm dfs để tiếp tục tìm kiếm đường đi. Nếu đường đi đã được tìm thấy, hàm sẽ trả về danh sách các đỉnh tạo thành đường đi từ đỉnh xuất phát đến đỉnh đích.

3.3.5 Kết quả đạt được

Kết quả đạt được sau khi chạy thuật toán.

moves = 16

Moves	N	Path
[botup, midleft, midleft, middown, botleft, botright, botup, midup, topleft, topdown, middown, botlef, botup, midright, midright, middown]	16	[*state*(1,3,5,2,6,4,7,8,*), *state*(1,3,5,2,6,*,7,8,4), *state*(1,3,5,2,*,6,7,8,4), *state*(1,3,5,*,2,6,7,8,4), *state*(1,3,5,7,2,6,*,8,4), *state*(1,3,5,7,2,6,8,*,4), *state*(1,3,5,7,2,6,8,4,*), *state*(1,3,5,7,2,*,8,4,6), *state*(1,3,*,7,2,5,8,4,6), *state*(1,*,3,7,2,5,8,4,6), *state*(1,2,3,7,*,5,8,4,6), *state*(1,2,3,7,4,5,8,*,6), *state*(1,2,3,7,4,5,*,8,6), *state*(1,2,3,*,4,5,7,8,6), *state*(1,2,3,4,*,5,7,8,6), *state*(1,2,3,4,5,*,7,8,6), *state*(1,2,3,4,5,6,7,8,*)]

0.663 seconds cpu t

```
?- length(Moves, N), dfs([state(1,3,5, 2,6,4, 7,8,*)], Moves, Path), !, print_path([start|Moves], Path),
format('~nmoves = ~w~n', [N]).
```

Hình 3.6: Kết quả sau khi chạy

Chương 4

Kết luận

Như vậy, với thuật toán DFS hay A* thì chúng ta đều có thể giải được bài toán 8-Puzzle, tuy nhiên giải thuật A* sẽ mang lại hiệu quả cao hơn. Đây là giải thuật dùng Heruistic để đánh giá các nút duyệt. Từ đó ta sẽ giải bài toán 8-Puzzle mà không tốn quá nhiều chi phí.
A* là một thuật toán tìm kiếm đường đi tốt nhất, dựa trên việc ước lượng khoảng cách còn lại từ trạng thái hiện tại đến trạng thái đích. Vì vậy, nếu ta sử dụng A* cho bài toán 8 puzzle, thuật toán có thể tìm được đường đi tối ưu từ trạng thái ban đầu đến trạng thái đích, tuy nhiên, thời gian thực hiện có thể lâu hơn so với DFS.

DFS là một thuật toán tìm kiếm theo chiều sâu, tức là nó tìm kiếm các đường đi sâu nhất trước khi quay lại tìm kiếm các đường đi khác. Khi áp dụng DFS vào bài toán 8 puzzle, thuật toán có thể tìm được một đường đi từ trạng thái ban đầu đến trạng thái đích, tuy nhiên, đường đi này không nhất thiết là đường đi tối ưu và thời gian thực hiện có thể lâu hơn so với A*.

Nguồn tham khảo và Code

Thuật toán DFS: https://vi.wikipedia.org/wiki/T%C3%ACm_ki%E1%BA%BFm_theo_chi%E1%BB%81u_s%C3%A2u
Thuật toán A*: https://vi.wikipedia.org/wiki/Gi%E1%BA%A3i_thu%E1%BA%ADt_t%C3%ACm_ki%E1%BA%BFm_A*
Source Code: https://github.com/HaKietHung2905/8Puzzle