

BỘ MÔN KHOA HỌC MÁY TÍNH
CT332: TRÍ TUỆ NHÂN TẠO
BÀI THỰC HÀNH SỐ 2
TÌM KIẾM HEURISTIC

I. Mục tiêu

- ✓ Biểu diễn bài toán trên không gian trạng thái và áp dụng các luật của heuristic để chọn những nhánh nào có nhiều khả năng nhất đến một giải pháp chấp nhận được – áp dụng heuristic trong bài toán 8 puzzle
- ✓ Ngôn ngữ sử dụng: C.

II. Nội dung

1. Mô tả bài toán

Cho một trạng thái bắt đầu gồm 8 ô số như hình bên dưới. Trạng thái bắt đầu là tuỳ theo mỗi bài toán. Làm cách nào để giải quyết bài toán này sao cho giống với trạng thái đích. Biết rằng chúng ta có thể thực hiện các thao tác / hành động sau đây để di chuyển các ô số.

Trạng thái khởi đầu

2	8	3
1	6	4
7		5

Trạng thái đích

1	2	3
8		4
7	6	5

Hành động 1. Di chuyển ô trống sang trái một ô. Nghĩa là đổi giá trị vị trí ô trống với ô bên trái nó. Còn với trường hợp ô trống nằm sát vị trí biên bên trái thì không thực hiện hành động.

Ví dụ: Với vị trí ô trống như hình bên dưới sau khi thực hiện hành động ta sẽ được

<table border="1"><tr><td>2</td><td>8</td><td>3</td></tr><tr><td>6</td><td></td><td>4</td></tr><tr><td>7</td><td>1</td><td>5</td></tr></table>	2	8	3	6		4	7	1	5	<table border="1"><tr><td>2</td><td>8</td><td>3</td></tr><tr><td></td><td>6</td><td>4</td></tr><tr><td>7</td><td>1</td><td>5</td></tr></table>	2	8	3		6	4	7	1	5
2	8	3																	
6		4																	
7	1	5																	
2	8	3																	
	6	4																	
7	1	5																	
Trước khi thực hiện hành động	Sau khi thực hiện hành động																		

Ví dụ: Với vị trí ô trống như hình bên dưới thì việc di chuyển sang trái là không thực hiện được, vì nó đang ở vị trí biên.

	8	3
2	6	4
7	1	5

6	8	3
	2	4
7	1	5

2	8	3
6	7	4
	1	5

Hành động 2. Di chuyển ô trống sang phải một ô. Nghĩa là đổi giá trị vị trí ô trống với ô bên phải nó. Còn với trường hợp ô trống nằm sát vị trí biên bên phải thì không thực hiện hành động.

Ví dụ: Với vị trí ô trống như hình bên dưới sau khi thực hiện hành động ta sẽ được

<table border="1" style="margin: auto;"> <tr><td>2</td><td>8</td><td>3</td></tr> <tr><td>6</td><td></td><td>4</td></tr> <tr><td>7</td><td>1</td><td>5</td></tr> </table>	2	8	3	6		4	7	1	5	<table border="1" style="margin: auto;"> <tr><td>2</td><td>8</td><td>3</td></tr> <tr><td>6</td><td>4</td><td></td></tr> <tr><td>7</td><td>1</td><td>5</td></tr> </table>	2	8	3	6	4		7	1	5
2	8	3																	
6		4																	
7	1	5																	
2	8	3																	
6	4																		
7	1	5																	
Trước khi thực hiện hành động	Sau khi thực hiện hành động																		

Ví dụ: Với vị trí ô trống như hình bên dưới thì việc di chuyển sang phải là không thực hiện được, vì nó đang ở vị trí biên.

<table border="1" style="margin: auto;"> <tr><td>2</td><td>8</td><td></td></tr> <tr><td>6</td><td>4</td><td>3</td></tr> <tr><td>7</td><td>1</td><td>5</td></tr> </table>	2	8		6	4	3	7	1	5	<table border="1" style="margin: auto;"> <tr><td>2</td><td>8</td><td>3</td></tr> <tr><td>6</td><td>4</td><td></td></tr> <tr><td>7</td><td>1</td><td>5</td></tr> </table>	2	8	3	6	4		7	1	5	<table border="1" style="margin: auto;"> <tr><td>2</td><td>8</td><td>3</td></tr> <tr><td>6</td><td>4</td><td>5</td></tr> <tr><td>7</td><td>1</td><td></td></tr> </table>	2	8	3	6	4	5	7	1	
2	8																												
6	4	3																											
7	1	5																											
2	8	3																											
6	4																												
7	1	5																											
2	8	3																											
6	4	5																											
7	1																												

Hành động 3. Di chuyển ô trống lên trên một ô. Nghĩa là đổi giá trị vị trí ô trống với ô phía trên nó. Còn với trường hợp ô trống nằm sát vị trí biên trên thì không thực hiện hành động.

Ví dụ: Với vị trí ô trống như hình bên dưới sau khi thực hiện hành động ta sẽ được

<table border="1" style="margin: auto;"> <tr><td>2</td><td>8</td><td>3</td></tr> <tr><td>6</td><td></td><td>4</td></tr> <tr><td>7</td><td>1</td><td>5</td></tr> </table>	2	8	3	6		4	7	1	5	<table border="1" style="margin: auto;"> <tr><td>2</td><td></td><td>3</td></tr> <tr><td>6</td><td>8</td><td>4</td></tr> <tr><td>7</td><td>1</td><td>5</td></tr> </table>	2		3	6	8	4	7	1	5
2	8	3																	
6		4																	
7	1	5																	
2		3																	
6	8	4																	
7	1	5																	
Trước khi thực hiện hành động	Sau khi thực hiện hành động																		

Ví dụ: Với vị trí ô trống như hình bên dưới thì việc di chuyển lên trên là không thực hiện được, vì nó đang ở vị trí biên.

<table border="1" style="margin: auto;"> <tr><td>2</td><td>8</td><td></td></tr> <tr><td>6</td><td>4</td><td>3</td></tr> <tr><td>7</td><td>1</td><td>5</td></tr> </table>	2	8		6	4	3	7	1	5	<table border="1" style="margin: auto;"> <tr><td>2</td><td></td><td>8</td></tr> <tr><td>6</td><td>4</td><td>3</td></tr> <tr><td>7</td><td>1</td><td>5</td></tr> </table>	2		8	6	4	3	7	1	5	<table border="1" style="margin: auto;"> <tr><td></td><td>2</td><td>8</td></tr> <tr><td>6</td><td>4</td><td>3</td></tr> <tr><td>7</td><td>1</td><td>5</td></tr> </table>		2	8	6	4	3	7	1	5
2	8																												
6	4	3																											
7	1	5																											
2		8																											
6	4	3																											
7	1	5																											
	2	8																											
6	4	3																											
7	1	5																											

Hành động 4. Di chuyển ô trống xuống dưới một ô. Nghĩa là đổi giá trị vị trí ô trống với ô phía dưới nó. Còn với trường hợp ô trống nằm sát vị trí biên dưới thì không thực hiện hành động.

Ví dụ: Với vị trí ô trống như hình bên dưới sau khi thực hiện hành động ta sẽ được

<table border="1" style="margin: auto;"> <tr><td>2</td><td>8</td><td>3</td></tr> <tr><td>6</td><td></td><td>4</td></tr> <tr><td>7</td><td>1</td><td>5</td></tr> </table>	2	8	3	6		4	7	1	5	<table border="1" style="margin: auto;"> <tr><td>2</td><td>8</td><td>3</td></tr> <tr><td>6</td><td>1</td><td>4</td></tr> <tr><td>7</td><td></td><td>5</td></tr> </table>	2	8	3	6	1	4	7		5
2	8	3																	
6		4																	
7	1	5																	
2	8	3																	
6	1	4																	
7		5																	
Trước khi thực hiện hành động	Sau khi thực hiện hành động																		

Ví dụ: Với vị trí ô trống như hình bên dưới thì việc di chuyển xuống dưới là không thực hiện được, vì nó đang ở vị trí biên.

<table border="1" style="margin: auto;"> <tr><td>2</td><td>8</td><td>3</td></tr> <tr><td>6</td><td>4</td><td>5</td></tr> <tr><td>7</td><td>1</td><td></td></tr> </table>	2	8	3	6	4	5	7	1		<table border="1" style="margin: auto;"> <tr><td>2</td><td>8</td><td>3</td></tr> <tr><td>6</td><td>4</td><td>5</td></tr> <tr><td>7</td><td></td><td>1</td></tr> </table>	2	8	3	6	4	5	7		1	<table border="1" style="margin: auto;"> <tr><td>2</td><td>8</td><td>3</td></tr> <tr><td>6</td><td>4</td><td>5</td></tr> <tr><td></td><td>7</td><td>1</td></tr> </table>	2	8	3	6	4	5		7	1
2	8	3																											
6	4	5																											
7	1																												
2	8	3																											
6	4	5																											
7		1																											
2	8	3																											
6	4	5																											
	7	1																											

2. Phân tích bài toán

Bài toán được mô tả trong không gian trạng thái với các đặc điểm sau:

- Trạng thái bài toán: ma trận biểu diễn bằng mảng 2 chiều 3x3, vị trí ô trống (hàng máy, cột máy).
- Các thao tác/ hành động (operator) để tạo ra trạng thái mới: di chuyển ô trống sang trái, di chuyển ô trống sang phải, di chuyển ô trống lên trên, di chuyển ô trống xuống dưới.
- Trạng thái đầu: Giá trị của mỗi vị trí trong ma trận. Vị trí hàng và cột của ô trống
- Trạng thái cuối: Vị trí các ô trùng vò trạng thái mục tiêu (GOAL: mục tiêu).

Việc tìm giải pháp cho trò chơi là tìm đường đi từ trạng thái bắt đầu đến trạng thái đích. Hay nói cách khác là tìm loạt các thao tác/ hành động để đến được trạng thái mục tiêu (vị trí các ô số giống với mục tiêu).

3. Cài đặt

3. 1. Cài đặt cấu trúc trạng thái:

Cho trạng thái bắt đầu với giá trị các ô như hình:

➤ Đầu vào: với ô trống, **sẽ mang giá trị là 0**

eightPuzzel[0][0] = 3

eightPuzzel[0][1] = 4

eightPuzzel[0][2] = 5

eightPuzzel[1][0] = 1

3	4	5
1		2
6	7	8

```

eightPuzzel[1][1] = 0
eightPuzzel[1][2] = 2
eightPuzzel[2][0] = 6
eightPuzzel[2][1] = 7
eightPuzzel[2][2] = 8

```

➤ Đầu ra:

```

eightPuzzel[0][0] = 0
eightPuzzel[0][1] = 1
eightPuzzel[0][2] = 2
eightPuzzel[1][0] = 3
eightPuzzel[1][1] = 4
eightPuzzel[1][2] = 5
eightPuzzel[2][0] = 6
eightPuzzel[2][1] = 7
eightPuzzel[2][2] = 8

```

	1	2
3	4	5
6	7	8

Sử dụng một cấu trúc gồm các trường sau để biểu diễn cho một trạng thái:

- eightPuzzel[ROWS][COLS]: các ô trong trạng thái của 8 ô số, với ROWS là số lượng hàng, COLS là số lượng cột.
- emptyRow: vị trí hàng của ô trống
- emptyCol: vị trí cột của ô trống

```

#include<stdio.h>
#include<stdlib.h>
#define ROWS 3
#define COLS 3
#define EMPTY 0
#define MAX_OPERATOR 4
#define Maxlength 500

const char* action[] = {"First State", "Move cell EMPTY to UP", "Move cell EMPTY to DOWN",
    "Move cell EMPTY to LEFT", "Move cell EMPTY to RIGHT"};
```

//Khai báo cấu trúc trạng thái của 8_puzzel

```

typedef struct{
    int eightPuzzel[ROWS][COLS];
    int emptyRow;
    int emptyCol;
}state;
```

3. 2. In trạng thái (State):

Viết hàm để in trạng thái của một puzzel

```
//In trang thai cua 8 - puzzel
void printState(State state){
    int row, col;
    printf("\n-----\n");
    for (row = 0; row<ROWS; row++){
        for(col = 0; col<COLS; col++)
            printf("|%d ", state.eightPuzzel[row][col]);
        printf("|\n");
    }
    printf("-----\n");
}
```

3. 3. So sánh hai trạng thái

Viết hàm kiểm tra trạng thái một giống với trạng thái hai hay không?

```
//So sanh trang thai state1 co giuong voi trang thai state2
int compareStates(State state1, State state2){
    if(state1.emptyRow != state2.emptyRow || state1.emptyCol != state2.emptyCol)
        return 0;
    int row, col;
    for(row=0; row<ROWS; row++)
        for(col=0; col<COLS; col++)
            if(state1.eightPuzzel[row][col] != state2.eightPuzzel[row][col])
                return 0;
    return 1;
}
```

3. 4. Kiểm tra trạng thái có phải trạng thái mục tiêu:

Viết hàm kiểm tra xem trạng thái hiện tại có bằng với trạng thái mục tiêu

```
//Ham kiem tra trang thai muc tieu
int goalcheck(State state, State goal){
    return compareStates(state, goal);
}
```

3. 5. Xây dựng các hành động làm thay đổi trạng thái:

Trạng thái hiện tại là **state** và kết quả sau khi thực hiện thao tác/ hành động được lưu vào ***result**. Nếu thao tác thực hiện thành công thì **result** nhận vào một trạng thái khác với **state**.

STT	Tên hàm	Ý nghĩa hàm
1	void upOperator(State state, State *result)	Di chuyển ô trống lên trên một ô. Đổi giá trị của ô trống với ô phía trên nó (nếu ô trống không phải ở biên trên)

2	void downOperator(State state, State *result)	Di chuyển ô trống xuống dưới một ô. Đổi giá trị của ô trống với ô phía trên nó (nếu ô trống không phải ở biên dưới)
3	void leftOperator (State state, State *result)	Di chuyển ô trống sang trái một ô. Đổi giá trị của ô trống với ô bên trái nó (nếu ô trống không phải ở biên trái)
4	void rightOperator(State state, State *result)	Di chuyển ô trống sang phải một ô. Đổi giá trị của ô trống với ô bên trái nó (nếu ô trống không phải ở biên phải)

3.5.1. Hành động di chuyển ô trống lên trên: hành động này được thực hiện nếu ô trống hiện tại có vị trí hàng > 0 , nếu bằng 0 thì hàng đang là vị trí biên không thể thực hiện việc di chuyển. Cho ***result** bằng với **state** (trạng thái hiện tại), gán giá trị cho **empRowCurrent** và **emptyColCurrent** bằng với giá trị tương ứng của **state**. Hoán đổi giá trị của ô trống với giá trị của ô phía trên nó.

```
//Hanh dong chuyen o trong len tren
int upoperator(State state, State *result){
    *result = state;
    int empRowCurrent = state.emptyRow, empColCurrent = state.emptyCol;
    if(empRowCurrent > 0){
        result->emptyRow = empRowCurrent - 1;
        result->emptyCol = empColCurrent;
        result->eightPuzzel[empRowCurrent][empColCurrent] = state.eightPuzzel[empRowCurrent - 1][empColCurrent];
        result->eightPuzzel[empRowCurrent - 1][empColCurrent] = EMPTY;
        return 1;
    }
    return 0;
}
```

3.5.2. Hành động di chuyển ô trống xuống: hành động này được thực hiện nếu ô trống hiện tại có vị trí hàng nhỏ hơn 2 (do sử dụng mảng biểu diễn trạng thái và đang xét puzzle 3x3), nếu bằng 2 thì hàng đang là vị trí biên không thể thực hiện việc di chuyển. Cho ***result** bằng với **state** (trạng thái hiện tại), gán giá trị cho **empRowCurrent** và **emptyColCurrent** bằng với giá trị tương ứng của **state**. Hoán đổi giá trị của ô trống với giá trị của ô phía dưới nó.

```
//Hanh dong chuyen o trong xuong duoi
int downOperator(State state, State *result){
    // Sinh viên tự code nhu ham upoperator
}
```

3.5.3. Hành động di chuyển ô trống sang trái: hành động này được thực hiện nếu ô trống hiện tại có vị trí cột lớn hơn 0 (do dùng mảng), nếu bằng 0 thì cột của ô trống đang là vị trí biên không thể thực hiện việc di chuyển. Cho *result bằng với state (trạng thái hiện tại), gán giá trị cho empRowCurrent và emptyColCurrent bằng với giá trị tương ứng của state. Hoán đổi giá trị của ô trống với giá trị của ô bên trái nó.

```
//Hanh dong chuyen o trong sang trai
int leftOperator(State state, State *result){
    *result = state;
    int empRowCurrent = state.emptyRow, empColCurrent = state.emptyCol;
    if(empColCurrent > 0){
        result->emptyRow = empRowCurrent;
        result->emptyCol = empColCurrent - 1;
        result->eightPuzzel[empRowCurrent][empColCurrent] = state.eightPuzzel[empRowCurrent][empColCurrent - 1];
        result->eightPuzzel[empRowCurrent][empColCurrent - 1] = EMPTY;
        return 1;
    }
    return 0;
}
```

3.5.4. Hành động di chuyển ô trống sang phải: hành động này được thực hiện nếu ô trống hiện tại có vị trí cột nhỏ hơn 2 (do dùng mảng), nếu bằng 2 thì cột của ô trống đang là vị trí biên không thể thực hiện việc di chuyển. Cho *result bằng với state (trạng thái hiện tại), gán giá trị cho empRowCurrent và emptyColCurrent bằng với giá trị tương ứng của state. Hoán đổi giá trị của ô trống với giá trị của ô bên phải nó.

```
//Hanh dong chuyen o trong sang phai
int rightOperator(State state, State *result){
    // Sinh vien tu code nhu ham leftOperator
}
```

3. 6. Xây dựng hàm để gọi các hành động: Trạng thái hiện tại là **state** và trạng thái kết quả là ***result**. opt (options) là số nguyên từ 1 đến 4 tương ứng với 4 hành động chuyển trạng

```
//Goi cach hanh dong chuyen o trong cho trang thai hien tai
int callOperators(State state, State *result, int opt){
    switch(opt){
        case 1: return upOperator(state, result);
        case 2: return downOperator(state, result);
        case 3: return leftOperator(state, result);
        case 4: return rightOperator(state, result);
        default: printf("Cannot call operators");
        return 0;
    }
}
```

3. 7. Bài tập 1: Biểu diễn trạng thái và các phép toán (hành động) của bài toán, gọi các hành động và in ra kết quả tương ứng từng hành động.

Các bước thực hiện:

- Khai báo cấu trúc trạng thái: State
- Cài đặt hàm compareState
- Cài đặt các hàm hành động chuyển trạng thái: di chuyển ô trống sang trái, di chuyển ô trống sang phải, di chuyển ô trống lên, di chuyển ô trống xuống.

- Cài đặt hàm callOperators để gọi các hành động.
- Cài đặt hàm in trạng thái ra màn hình.
- Cài đặt hàm main()

```

int main(){
    State state, result;
    state.emptyRow = 1;
    state.emptyCol = 1;
    state.eightPuzzle[0][0] = 3;
    state.eightPuzzle[0][1] = 4;
    state.eightPuzzle[0][2] = 5;
    state.eightPuzzle[1][0] = 1;
    state.eightPuzzle[1][1] = 0;
    state.eightPuzzle[1][2] = 2;
    state.eightPuzzle[2][0] = 6;
    state.eightPuzzle[2][1] = 7;
    state.eightPuzzle[2][2] = 8;
    printf("Trang thai bat dau \n");
    printState(state);
    int opt;
    for(opt= 1; opt<=4; opt++) {
        callOperators(state, &result, opt);
        if(!compareStates(state, result)){
            printf("Hanh dong %s thanh cong\n", action[opt]);
            printState(result);
        } else {
            printf("Hanh dong %s khong thanh cong\n", action[opt]);
        }
    }
    return 0;
}

```

Kết quả chương trình:

```

Trang thai bat dau

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

Hanh dong Move cell EMPTY to UP thanh cong

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

Hanh dong Move cell EMPTY to DOWN thanh cong

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

Hanh dong Move cell EMPTY to LEFT thanh cong

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

Hanh dong Move cell EMPTY to RIGHT thanh cong

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

3.8. Tính heuristic

- Với bài toán 8 puzzel ta có 2 cách để tính heuristic:

Cách 1: đếm số lượng ô số có vị trí sai so với goal

Ví dụ: Với trạng thái như hình: các ô có vị trí sai sẽ là: 3, 4, 5, 1, 2, ô trống => heuristic = 6

3	4	5
1		2
6	7	8

Trạng thái ban đầu

	1	2
3	4	5
6	7	8

Trạng thái mục tiêu

```
//Ham heuristic 1
//Dem so o sai khac so voi trang thai muc tieu
int heuristic1(State state, State goal){
    int row, col, count=0;
    for(row=0; row<ROWS; row++)
        for(col=0; col<COLS; col++)
            if(state.eightPuzzel[row][col] != goal.eightPuzzel[row][col])
                count++;
    return count;
}
```

Cách 2: Đếm số bước để chuyển ô sai về vị trí đúng.

Ví dụ: Với trạng thái như hình:

- Với số 3 ở trạng thái bắt đầu ở dòng 0 - cột 0, nó sẽ cần mất 1 lần di chuyển xuống để di chuyển về đúng vị trí của nó trong trạng thái mục tiêu tại dòng 1 - cột 0.
- Với số 1 ở trạng thái bắt đầu ở dòng 1 - cột 0, nó sẽ cần mất 2 lần di chuyển xuống để di chuyển về đúng vị trí của nó trong trạng thái mục tiêu tại dòng 0 – cột 1 (xem hình mũi tên).

3	4	5
1		2
6	7	8

Trạng thái ban đầu

	1	2
3	4	5
6	7	8

Trạng thái mục tiêu

```

//Ham heuristic 2 - Manhattan
//Dem so buoc chuyen o sai ve o dung cua trang thai muc tieu
int heuristic2(State state, State goal) {
    int count = 0;
    int row, col, row_g, col_g;
    for (row = 0; row < ROWS; row++) {
        for (col = 0; col < COLS; col++) {
            if (state.eightPuzzel[row][col] != EMPTY) {
                for (row_g = 0; row_g < ROWS; row_g++) {
                    for (col_g = 0; col_g < COLS; col_g++) {
                        if (state.eightPuzzel[row][col] == goal.eightPuzzel[row_g][col_g]) {
                            count += abs(row - row_g) + abs(col - col_g);
                            col_g = COLS; //break loop col_g
                            row_g = ROWS; //break loop row_g
                        }
                    }
                }
            }
        }
    }
    return count;
}

```

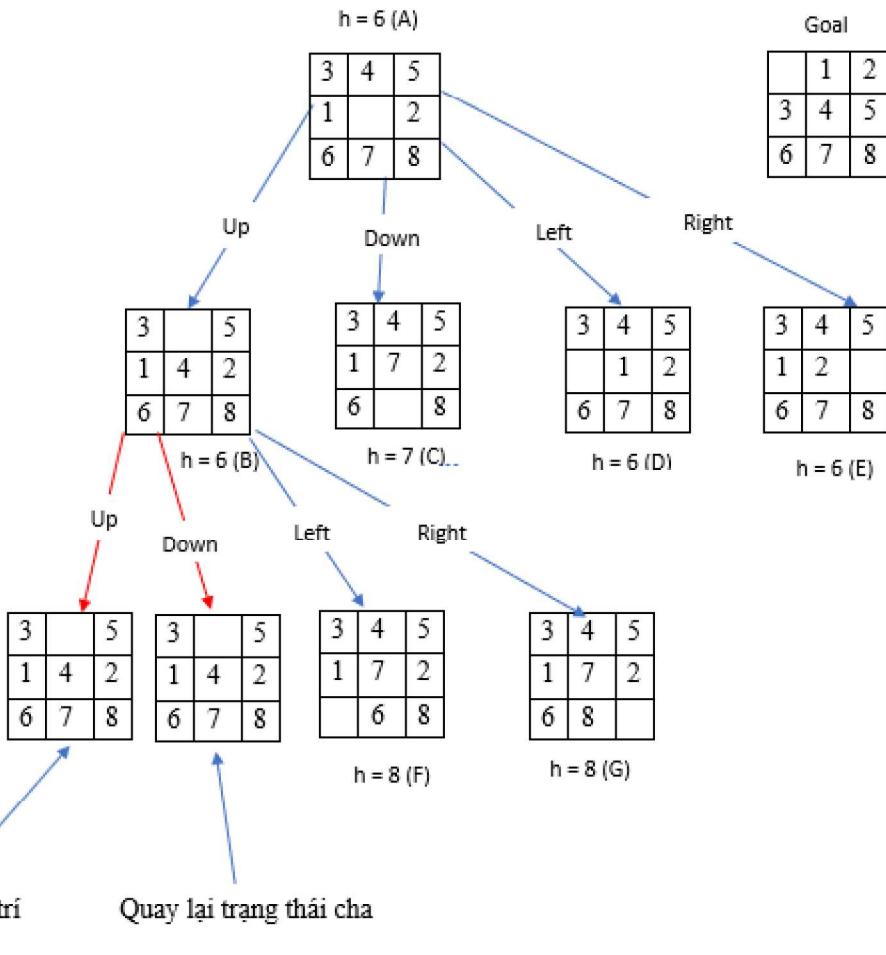
3. 9. Mô hình minh họa dựng cây tìm kiếm – duyệt theo best first search (BFS) (Sử dụng tìm kiếm h้าu ăn):

Lưu ý: Những cung triển khai không gian trạng thái có màu xanh dương là phép toán tương ứng hợp lệ, những cung màu đỏ (nhạt) là những phép toán triển khai không hợp lệ. Trong đó h là số lượng các vị trí còn sai.

Open	Open (Sorted)	Close
A (6)	A (6)	

Open	Open (Sorted)	Close
B (6)	B (6)	A (6)
C (7)	D (6)	
D (6)	E (6)	
E (6)	C (7)	

Open	Open (Sorted)	Close
D (6)	D (6)	A (6)
E (6)	E (6)	B (6)
C (7)	C (7)	
F (8)	F (8)	
G (8)	G (8)	



3. 10. Cài đặt cấu trúc để xây dựng cây tìm kiếm Không gian trạng thái:

Sử dụng một cấu trúc gồm các trường sau để biểu diễn cho một tráng thái:

- state: dùng để lưu trạng thái bài 8 ô số.
- Parent: dùng để lưu nút cha của một nút mới được triển khai.
- no_function: Lưu số thứ tự phép toán đã thực hiện tương ứng.
- heuristic: để lưu số vị trí sai so với goal

```
//Khai bao cau truc Node de dung cay tim kiem
typedef struct Node{
    State state;
    struct Node* parent;
    int no_function;
    int heuristic;
}Node;
```

3. 11. Cài đặt cấu trúc danh sách: để lưu trạng thái (đã duyệt/ đang chờ duyệt) trong quá trình duyệt để tìm đến trạng thái mục tiêu.

```
//Khai bao cau truc danh sach
typedef struct{
    Node* Elements[Maxlength];
    int size;
}List;

//Khai tao danh sach rong
void makeNull_List(List *list){
    // Sinh viên tự cài đặt
}

//Kiem tra danh sach rong
int empty_List(List list){
    // Sinh viên tự cài đặt
}

//Kiem tra danh sach FULL
int full_List(List list){
    // Sinh viên tự cài đặt
}

//Truy van gia tri cua phan tu vitri p
Node* element_at(int p, List list){
    // Sinh viên tự cài đặt
}
```

```

//Them phan tu vao vi tri position trong danh sach
void push_List(Node* x, int position, List *list){
    if(!full_List(*list)){
        int q;
        for(q=list->size; q>=position; q--)
            list->Elements[q] = list->Elements[q-1];
        list->Elements[position-1]=x;
        list->size++;
    }
    else printf("List is full\n");
}

//Xoa phan tu tai vi tri position ra khoi danh sach
void delete_List(int position, List *list){
    if(empty_List(*list))
        printf("List is empty\n");
    else if(position<1 || position>list->size)
        printf("Position is not possible to delete\n");
    else{
        int i;
        for (i=position-1; i<list->size; i++)
            list->Elements[i]=list->Elements[i+1];
        list->size--;
    }
}

```

3.12. Tìm trạng thái

- Kiểm tra trạng thái có trong Open/Close hay không, có thì trả về nút đó và vị trí.

```

//Tim trang thai state co thuoc Open hoac Close hay ko?
//Luu vi tri tim duoc vao bien *position
Node* find_State(State state, List list, int *position){
    int i;
    for(i=1; i<=list.size; i++)
        if(compareStates(element_at(i,list)->state,state)){
            *position = i;
            return element_at(i,list);
        }
    return NULL;
}

```

3.13. Sắp xếp danh sách theo h

Sắp xếp danh sách theo hàm f (f = h) theo thứ tự tăng dần

```

//Sap xep danh sach theo so heuristic
void sort_List(List *list){
    int i, j;
    for(i=0; i<list->size-1; i++)
        for(j=i+1; j<list->size; j++)
            if(list->Elements[i]->heuristic>list->Elements[j]->heuristic){
                Node* node = list->Elements[i];
                list->Elements[i] = list->Elements[j];
                list->Elements[j] = node;
            }
}

```

3.14. Thuật toán tìm kiếm tốt nhất đầu tiên (Best First Search):

```
PNode bestFirstSearch(PState init_state) {  
    PNode root = new Node();  
    root->state = init_state;  
    root->parent = NULL;  
    root->f = 0;  
    frontier.insert(root);  
    explored.clear();  
    while (!empty(frontier)) {  
        //Lấy 1 nút từ đường biên có f(n) nhỏ nhất  
        //và loại bỏ nó ra khỏi đường biên  
        Node* node = frontier.pop();  
        if (node là nút mục tiêu)  
            return node;  
        insert(node, explored);  
    }  
}
```

```
for (child là nút con của node) {  
    Tính child->f;  
    if (child->state không thuộc frontier và  
        child->state không thuộc explored) {  
        child->parent = node;  
        frontier.insert(child);  
    } (*) else if (child->state nằm trong đường biên  
        và có f lớn hơn child->f)  
        Thay thế nút nằm trên đường biên bằng child  
    (***) else if (child->state nằm trong explored  
        và có f lớn hơn child->f) {  
        Loại bỏ nút chứa child->state  
        ra khỏi explored  
        frontier.insert(child);  
    }  
}  
return NULL; //Thất bại, không tìm thấy lời giải  
}
```

3.15. Chương trình thuật toán BFS cho bài toán 8 puzzel:

```
//Thuật toán tìm kiếm tốt nhất đầu tiên
//Hàm f = h
Node* best_first_search(State state, State goal){
    List Open_BFS;
    List Close_BFS;
    makeNull_List(&Open_BFS);
    makeNull_List(&Close_BFS);
    Node* root = (Node*)malloc(sizeof(Node));
    root->state = state;
    root->parent = NULL;
    root->no_function = 0;
    root->heuristic = heuristic1(root->state, goal);
    push_List(root, Open_BFS.size+1, &Open_BFS);
    while(!empty_List(Open_BFS)){
        Node* node = element_at(1,Open_BFS);
        delete_List(1, &Open_BFS);
        push_List(node, Close_BFS.size+1, &Close_BFS);
        if(goalcheck(node->state, goal))
            return node;
        int opt;
        for(opt=1; opt<=MAX_OPERATOR; opt++){
            State newstate;
            newstate = node->state;
            if(callOperators(node->state, &newstate, opt)){
                Node* newNode = (Node*)malloc(sizeof(Node));
                newNode->state = newstate;
                newNode->parent = node;
                newNode->no_function = opt;
                newNode->heuristic = heuristic1(newstate, goal);
                //Kiểm tra trạng thái mới sinh ra có thuộc Open_BFS/ Close_BFS
                int pos_Open, pos_Close;
                Node* nodeFoundOpen = find_State(newstate, Open_BFS, &pos_Open);
                Node* nodeFoundClose = find_State(newstate, Close_BFS, &pos_Close);
                if(nodeFoundOpen == NULL && nodeFoundClose == NULL){
                    push_List(newNode, Open_BFS.size+1, &Open_BFS);
                }
                else if(nodeFoundOpen != NULL && nodeFoundOpen->heuristic > newNode->heuristic){
                    delete_List(pos_Open, &Open_BFS);
                    push_List(newNode, pos_Open, &Open_BFS);
                }
                else if(nodeFoundClose != NULL && nodeFoundClose->heuristic > newNode->heuristic){
                    delete_List(pos_Close, &Close_BFS);
                    push_List(newNode, Open_BFS.size+1, &Open_BFS);
                }
                sort_List(&Open_BFS);
            }
        }
    }
    return NULL;
}
```

3.16. Sinh viên viền tự viết lại hàm best_first_search với hàm tính heuristic2

3. 17. In kết quả đường đi của best first search (BFS)

In ra kết quả đường đi của best first search

```
//Ham in ket qua cua thuật toán BFS
void print_WaysToGetGoal(Node* node) {
    List listPrint;
    makeNull_List(&listPrint);
    //Duyệt ngược về nút parent
    while(node->parent != NULL){
        push_List(node, listPrint.size+1, &listPrint);
        node = node->parent;
    }
    push_List(node, listPrint.size+1, &listPrint);
    //In ra thứ tự hành động chuyển ô trong
    int no_action = 0, i;
    for(i=listPrint.size; i>0; i--){
        printf("\nAction %d: %s", no_action, action[element_at(i, listPrint)->no_function]);
        printState(element_at(i, listPrint)->state);
        no_action++;
    }
}
```

3. 18. Bài tập 2:

Biểu diễn trạng thái và các phép toán (hành động) của bài toán, gọi các hành động và xây dựng cây tìm kiếm không gian trạng thái cho bài toán 8 ô số.

Các bước thực hiện:

- Khai báo các hằng cần thiết cho bài toán 8 ô số
- Khai báo cấu trúc trạng thái: State
- Cài đặt các hàm hành động chuyển trạng thái: hàm di chuyển ô lên trên, hàm di chuyển ô xuống dưới, hàm di chuyển ô sang trái, hàm di chuyển ô sang phải.
- Cài đặt cấu trúc Node để hỗ trợ xây dựng cây tìm kiếm.
- Cài đặt cấu trúc danh sách lưu trữ trạng thái tìm kiếm.
- Cài đặt thuật toán tìm kiếm tốt nhất đầu tiên (best first search)
- Cài đặt hàm in kết quả tìm kiếm đến trạng thái mục tiêu.
- Viết hàm **main()** cho chương trình.

```

#include<stdio.h>
#include<stdlib.h>
// Các hằng cần thiết

const char* action[] = {"First State", "Move cell EMPTY to UP", "Move cell EMPTY to DOWN",
                        "Move cell EMPTY to LEFT", "Move cell EMPTY to RIGHT"};

//Khai bao cau truc trang thai cua 8_puzzel
typedef struct{
    ...
}State;

//So sánh trạng thái state1 có giống với trạng thái state2
int compareStates(State state1, State state2){
    ...
}

//Hàm kiểm tra trạng thái mục tiêu
int goalcheck(State state, State goal){
    return compareStates(state, goal);
}

//Hành động chuyển ô trong len trên
int upOperator(State state, State *result){
    ...
}

//Hành động chuyển ô trong xuống dưới
int downOperator(State state, State *result){
    ...
}

//Hành động chuyển ô trong sang trái
int leftOperator(State state, State *result){
    ...
}

//Hành động chuyển ô trong sang phải
int rightOperator(State state, State *result){
    ...
}

//Gọi cách hành động chuyển ô trong cho trạng thái hiện tại
int callOperators(State state, State *result, int opt){
    ...
}

//In trạng thái của 8 - puzzel
void printState(State state){
    int row, col;
    printf("\n-----\n");
    for (row = 0; row<ROWS; row++){
        for (col = 0; col<COLS; col++){
            printf("|%d ", state.eightPuzzel[row][col]);
        }
        printf("|\n");
    }
    printf("-----\n");
}

//Hàm heuristic 1
//Đếm số ô sai khác so với trạng thái mục tiêu
int heuristic1(State state, State goal){
    ...
}

```

```

//Ham heuristic 2 - Manhattan
//Dem so buoc chuyen o sai ve o dung cua trang thai muc tieu
int heuristic2(State state, State goal) {
    ...
}

//Khai bao cau truc Node de dung cay tim kiem
typedef struct Node{
    State state;
    struct Node* parent;
    int no_function;
    int heuristic;
}Node;

//Khai bao cau truc danh sach
typedef struct{
    Node* Elements[Maxlength];
    int size;
}List;

//Khoi tao danh sach rong
void makeNull_List(List *list){
    ...
}

//Kiem tra danh sach rong
int empty_List(List list){
    ...
}

//Kiem tra danh sach FULL
int full_List(List list){
    ...
}

//Truy van gia tri cua phan tu vitri p
Node* element_at(int p, List list){
    ...
}

//Them phan tu vao vi tri position trong danh sach
void push_List(Node* x, int position, List *list){
    if(!full_List(*list)){
        int q;
        for(q=list->size; q>=position; q--)
            list->Elements[q] = list->Elements[q-1];
        list->Elements[position-1]=x;
        list->size++;
    }
    else printf("List is full\n");
}

//Tim trang thai state co thuoc Open hoac Close hay ko?
//Luu vi tri tim duoc vao bien *position
Node* find_State(State state, List list, int *position){
    int i;
    for(i=1; i<=list.size; i++)
        if(compareStates(element_at(i,list)->state,state)){
            *position = i;
            return element_at(i,list);
        }
    return NULL;
}

```

```

//Xoa phan tu tai vi tri position ra khoi danh sach
void delete_List(int position, List *list){
    if(empty_List(*list))
        printf("List is empty\n");
    else if(position<1 || position>list->size)
        printf("Position is not possible to delete\n");
    else{
        int i;
        for (i=position-1; i<list->size; i++)
            list->Elements[i]=list->Elements[i+1];
        list->size--;
    }
}

//Sap xep danh sach theo trung so heuristic
void sort_List(List *list){
    int i, j;
    for(i=0; i<list->size-1; i++)
        for(j=i+1; j<list->size; j++)
            if(list->Elements[i]->heuristic>list->Elements[j]->heuristic){
                Node* node = list->Elements[i];
                list->Elements[i] = list->Elements[j];
                list->Elements[j] = node;
            }
}

```

```

//Thuat toan tim kiem tot nhat dau tien
//Ham f = h
Node* best_first_search(State state, State goal){
    List Open_BFS;
    List Close_BFS;
    makeNull_List(&Open_BFS);
    makeNull_List(&Close_BFS);
    Node* root = (Node*)malloc(sizeof(Node));
    root->state = state;
    root->parent = NULL;
    root->no_function = 0;
    root->heuristic = heuristic1(root->state, goal);
    push_List(root, Open_BFS.size+1, &Open_BFS);
    while(!empty_List(Open_BFS)){
        Node* node = element_at(1,Open_BFS);
        delete_List(1, &Open_BFS);
        push_List(node, Close_BFS.size+1, &Close_BFS);
        if(goalcheck(node->state, goal))
            return node;
        int opt;
        for(opt=1; opt<=MAX_OPERATOR; opt++){
            State newstate;
            newstate = node->state;
            if(callOperators(node->state, &newstate, opt)){
                Node* newNode = (Node*)malloc(sizeof(Node));
                newNode->state = newstate;
                newNode->parent = node;
                newNode->no_function = opt;
                newNode->heuristic = heuristic1(newstate, goal);
                //Kiem tra trang thai moi sinh ra co thuoc Open_BFS/ Close_BFS
                int pos_Open, pos_Close;
                Node* nodeFoundOpen = find_State(newstate, Open_BFS, &pos_Open);
                Node* nodeFoundClose = find_State(newstate, Close_BFS, &pos_Close);
                if(nodeFoundOpen == NULL && nodeFoundClose == NULL){
                    push_List(newNode, Open_BFS.size+1, &Open_BFS);
                }
                else if(nodeFoundOpen != NULL && nodeFoundOpen->heuristic > newNode->heuristic){
                    delete_List(pos_Open, &Open_BFS);
                    push_List(newNode, pos_Open, &Open_BFS);
                }
                else if(nodeFoundClose != NULL && nodeFoundClose->heuristic > newNode->heuristic){
                    delete_List(pos_Close, &Close_BFS);
                    push_List(newNode, Open_BFS.size+1, &Open_BFS);
                }
                sort_List(&Open_BFS);
            }
        }
    }
    return NULL;
}

//Ham in ket qua cua thuat toan BFS
void print_WaysToGetGoal(Node* node){
    ...
}

```

```

int main(){
    State state;
    state.emptyRow = 1;
    state.emptyCol = 1;
    state.eightPuzzel[0][0] = 3;
    state.eightPuzzel[0][1] = 4;
    state.eightPuzzel[0][2] = 5;
    state.eightPuzzel[1][0] = 1;
    state.eightPuzzel[1][1] = 0;
    state.eightPuzzel[1][2] = 2;
    state.eightPuzzel[2][0] = 6;
    state.eightPuzzel[2][1] = 7;
    state.eightPuzzel[2][2] = 8;

    State goal;
    goal.emptyRow = 0;
    goal.emptyCol = 0;
    goal.eightPuzzel[0][0] = 0;
    goal.eightPuzzel[0][1] = 1;
    goal.eightPuzzel[0][2] = 2;
    goal.eightPuzzel[1][0] = 3;
    goal.eightPuzzel[1][1] = 4;
    goal.eightPuzzel[1][2] = 5;
    goal.eightPuzzel[2][0] = 6;
    goal.eightPuzzel[2][1] = 7;
    goal.eightPuzzel[2][2] = 8;
    Node* p = best_first_search(state, goal);
    print_WaysToGetGoal(p);
    return 0;
}

```

Test case có thể dùng để kiểm tra:

Trạng thái ban đầu

3	4	5
1	0	2
6	7	8

Trạng thái mục tiêu

0	1	2
3	4	5
6	7	8

Kết quả chương trình:

Action 0: First State

3	4	5
1	0	2
6	7	8

Action 1: Move cell EMPTY to LEFT

3	4	5
0	1	2
6	7	8

Action 2: Move cell EMPTY to UP

0	4	5
3	1	2
6	7	8

Action 3: Move cell EMPTY to RIGHT

4	0	5
3	1	2
6	7	8

Action 4: Move cell EMPTY to DOWN

4	1	5
3	0	2
6	7	8

Action 5: Move cell EMPTY to RIGHT

4	1	5
3	2	0
6	7	8

Action 6: Move cell EMPTY to UP

4	1	0
3	2	5
6	7	8

Action 21: Move cell EMPTY to RIGHT

1	0	5
3	2	4
6	7	8

Action 22: Move cell EMPTY to DOWN

1	2	5
3	0	4
6	7	8

Action 23: Move cell EMPTY to RIGHT

1	2	5
3	4	0
6	7	8

Action 24: Move cell EMPTY to UP

1	2	0
3	4	5
6	7	8

Action 25: Move cell EMPTY to LEFT

1	0	2
3	4	5
6	7	8

Action 26: Move cell EMPTY to LEFT

0	1	2
3	4	5
6	7	8

.....

3.19. Bài tập 3:

Biểu diễn trạng thái và các phép toán (hành động) của bài toán, gọi các hành động và xây dựng cây tìm kiếm không gian trạng thái theo best first search (BFS) cho bài toán 8 puzzel. Để cài đặt thuật toán, chúng ta cần sử dụng Vector (giống như danh sách) để lưu trữ các đỉnh trong quá trình tìm kiếm. Bài tập này các em không cần cài đặt cấu trúc List mà các em sử dụng thư viện có sẵn của thư viện STL. Để sử dụng tính năng này, em phải viết chương trình bằng ngôn ngữ C++ và phải đặt tên chương trình có phần mở rộng *.cpp hoặc *.cc.

Các bước thực hiện:

- Khai báo các hằng cần thiết cho bài toán 8 ô số
- Khai báo cấu trúc trạng thái: State
- Cài đặt các hàm hành động chuyển trạng thái: hàm di chuyển ô lên trên, hàm di chuyển ô xuống dưới, hàm di chuyển ô sang trái, hàm di chuyển ô sang phải.
- Cài đặt cấu trúc Node để hỗ trợ xây dựng cây trong quá trình tìm kiếm.
- Cài đặt thuật toán tìm kiếm tốt nhất đầu tiên (Sử dụng vector) dưới sự hỗ trợ của thư viện STL
- Cài đặt hàm in kết quả tìm kiếm.
- Cài đặt hàm heuristic
- Viết hàm **main()** cho chương trình.

Thuật toán best_first_search được cài đặt dưới sự hỗ trợ của thư viện C++

3.20. Tìm kiếm một trạng thái

```
//Tim trang thai state co thuoc Open hoac Close hay ko?  
//Luu vi tri tim duoc vao bien *position  
Node* find_State(State state, vector<Node*> v, vector<Node *>::iterator *position){  
    vector<Node *>::iterator it = v.begin();  
    if(v.size() == 0) return NULL;  
    //lặp qua các phần tử cho đến khi biến it là giá trị cuối cùng  
    while(it != v.end()){  
        if(compareStates((*it)->state,state)){  
            *position = it;  
            return *it;  
        }  
        // Xoá phần tử it ra khỏi v  
        it = v.erase(it);  
    }  
    return NULL;  
}
```

Tiêu chí sắp xếp một vector

```
// Sap xep danh sach theo so heuristic  
// Tiêu chí để sort trong vector  
// phần tử trong vector sẽ được truyền lần lượt vào Node* a và Node *b để so sánh  
bool compareHeuristic(Node* a, Node *b){  
    return a->heuristic > b->heuristic;  
}
```

3.21. Hàm best_first_search

```
// Thuật toán tìm kiếm tốt nhất đầu tiên
// Hàm f = h
Node* best_first_search(State state, State goal) {
    vector<Node*> Open_BFS(Maxlength);
    Open_BFS.clear(); // Làm rỗng Open_BFS
    vector<Node*> Close_BFS(Maxlength);
    Close_BFS.clear(); // Làm rỗng Close_BFS
    Node* root = (Node*)malloc(sizeof(Node));
    root->state = state;
    root->parent = NULL;
    root->no_function = 0;
    root->heuristic = heuristic1(root->state, goal);
    // Đưa root vào Open_BFS vào vị trí cuối cùng
    Open_BFS.push_back(root);
    while(!Open_BFS.empty()){
        // Lấy một phần tử trong Open_BFS ở vị trí cuối cùng
        Node* node = Open_BFS.back();
        // Xoá phần tử cuối cùng trong Open_BFS
        Open_BFS.pop_back();
        // Đưa node vào Close_BFS
        Close_BFS.push_back(node);
        if(goalcheck(node->state, goal)){
            cout<<"Goal\n";
            return node;
        }
        int opt;
        for(opt=1; opt<=MAX_OPERATOR; opt++){
            State newstate;
            newstate = node->state;
            if(callOperators(node->state, &newstate, opt)){
                Node* newNode = (Node*)malloc(sizeof(Node));
                newNode->state = newstate;
                newNode->parent = node;
                newNode->no_function = opt;
                newNode->heuristic = heuristic1(newstate, goal);
                // Kiểm tra trạng thái mới sinh ra có thuộc Open_BFS/ Close_BFS
                vector<Node*>::iterator pos_Open, pos_Close;
                Node* nodeFoundOpen = find_State(newstate, Open_BFS, &pos_Open);
                Node* nodeFoundClose = find_State(newstate, Close_BFS, &pos_Close);
                if(nodeFoundOpen == NULL && nodeFoundClose == NULL){
                    Open_BFS.push_back(newNode);
                }
                else if(nodeFoundOpen != NULL && nodeFoundOpen->heuristic > newNode->heuristic){
                    // Xoá một phần tử trong Open_BFS tại pos_Open
                    Open_BFS.erase(pos_Open);
                    Open_BFS.push_back(newNode);
                }
                else if(nodeFoundClose != NULL && nodeFoundClose->heuristic > newNode->heuristic){
                    // Xoá một phần tử trong Close_BFS tại pos_Close
                    Close_BFS.erase(pos_Close);
                    Open_BFS.push_back(newNode);
                }
            }
        }
        // Sắp xếp Open_BFS theo hàm compareHeuristic
        sort(Open_BFS.begin(), Open_BFS.end(), compareHeuristic);
    }
    return NULL;
}
```

3.22. Hàm in kết quả của thuật toán

```
//Ham in ket qua cua thuật toán BFS
void print_WaysToGetGoal(Node* node){
    vector<Node*> vectorPrint;
    //Duyệt ngược về nút parent
    while(node->parent != NULL){
        vectorPrint.push_back(node);
        node = node->parent;
    }
    vectorPrint.push_back(node);
    //In ra thứ tự hành động di chuyển ở trong
    int no_action = 0, i;
    for(i=vectorPrint.size() - 1; i >= 0; i--){
        printf("\nAction %d: %s", no_action, action[vectorPrint.at(i)->no_function]);
        printstate(vectorPrint.at(i)->state);
        no_action++;
    }
}
```

Cấu trúc toàn bộ chương trình

```
#include<stdio.h>
#include<stdlib.h>
// Các hằng cần thiết
#include <vector>
#include <algorithm> // std::sort
using namespace std;
const char* action[] = {"First State", "Move cell EMPTY to UP", "Move cell EMPTY to DOWN",
                       "Move cell EMPTY to LEFT", "Move cell EMPTY to RIGHT"};

//Khai báo cấu trúc trạng thái của 8_puzzle
typedef struct{
    ...
}State;

//So sánh trạng thái state1 có giống với trạng thái state2
int compareStates(State state1, State state2){
    ...
}

//Hành động chuyển ở trong len trên
int upOperator(State state, State *result){
    ...
}

//Hành động chuyển ở trong xuống dưới
int downOperator(State state, State *result){
    ...
}

//Hành động chuyển ở trong sang trái
int leftOperator(State state, State *result){
    ...
}

//Hành động chuyển ở trong sang phải
int rightOperator(State state, State *result){
    ...
}

//Gọi cách hành động chuyển ở trong cho trạng thái hiện tại
int callOperators(State state, State *result, int opt){
    ...
}
```

```

//Ham heuristic 1
//Dem so o sai khac so voi trang thai muc tieu
int heuristic1(State state, State goal){
    ...
}

//Ham heuristic 2 - Manhattan
//Dem so buoc chuyen o sai ve o dung cua trang thai muc tieu
int heuristic2(State state, State goal) {
    ...
}

//Khai bao cau truc Node de dung cay tim kiem
typedef struct Node{
    ...
}Node;

//In trang thai cua 8 - puzzle
void printState(State state){
    ...
}

//Tim trang thai state co thuoc Open hoac Close hay ko?
//Luu vi tri tim duoc vao bien *position
Node* find_state(State state, vector<Node*> v, vector<Node *>::iterator *position){
    ...
}

//Ham kiem tra trang thai muc tieu
int goalcheck(State state, State goal){
    ...
}

// Sap xep danh sach theo trung so heuristic
// Tiêu chí để sort trong vector
// phần tử trong vector sẽ được truyền lần lượt vào Node* a và Node *b để so sánh
bool compareHeuristic(Node* a, Node *b){
    ...
}

//Thuật toán tìm kiếm tốt nhất đầu tiên
//Ham f = h
Node* best_first_search(State state, State goal){
    ...
}

//Ham in ket qua cua thuat toan BFS
void print_WaysToGetGoal(Node* node){
    ...
}

int main(){
    ...
}

```