【Experiment name】 experiment one Eight Number Problem Solving

【Purpose】

Master the 8-digit problem solving method based on the A algorithm.

【Experimental Requirements】

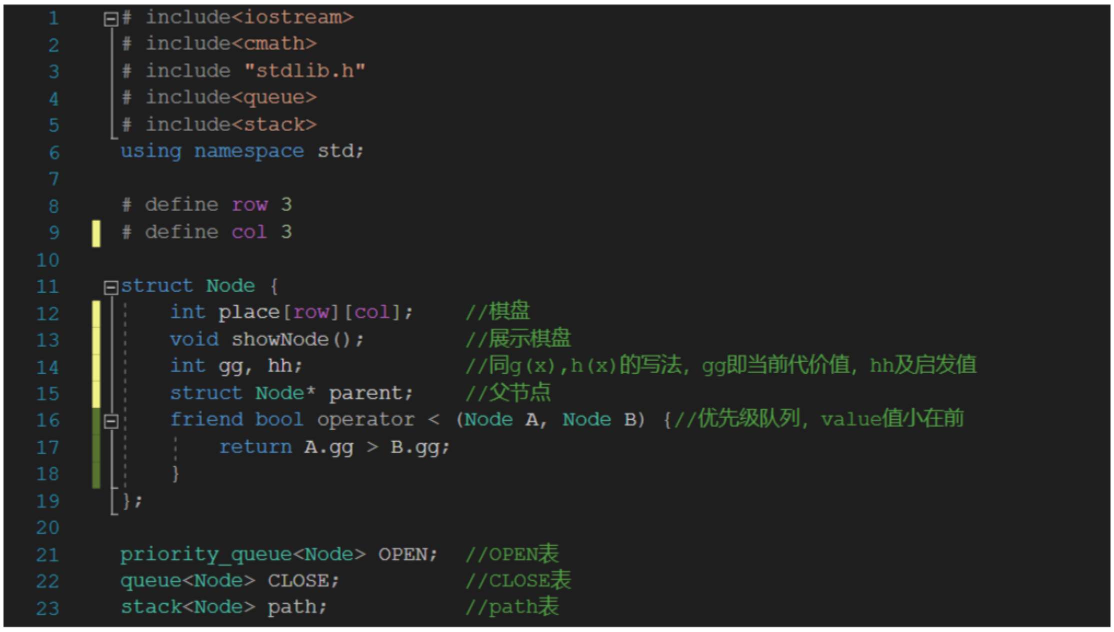
Define the cost function G(x) and the heuristic function H(X), and use the A algorithm to solve it.

Enter an initial state and a goal state.

Output the route from the initial state to the goal state.

【Experimental content】

basic data format



```
# include<iostream>
# include<cmath>
# include "stdlib.h"
# include<queue>
# include<stack>
using namespace std;

# define row 3
# define col 3

struct Node {
    int place[row][col];     //棋盘
    void showNode();         //展示棋盘
    int gg, hh;              //同g(x),h(x)的写法, gg即当前代价值, hh及启发值
    struct Node* parent;     //父节点
    friend bool operator < (Node A, Node B) {//优先级队列, value值小在前
        return A.gg > B.gg;
    }
};

priority_queue<Node> OPEN;  //OPEN表
queue<Node> CLOSE;          //CLOSE表
stack<Node> path;           //path表
```

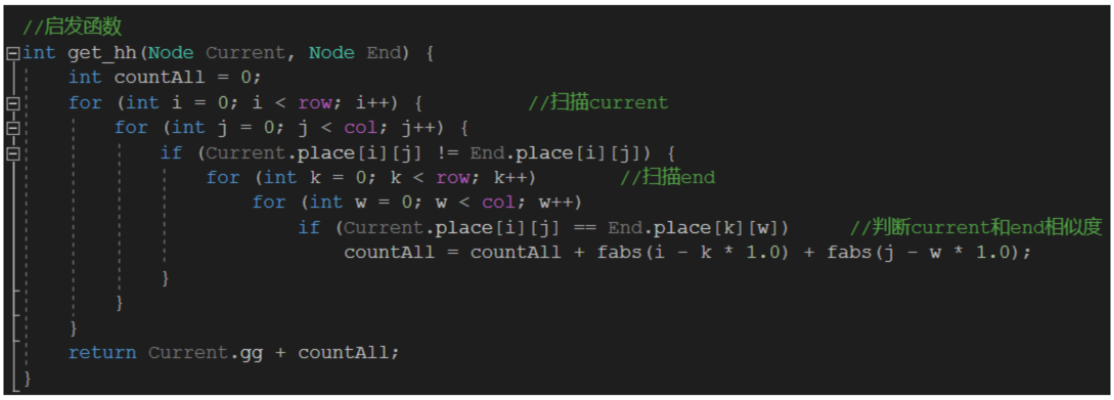
① As shown in the figure, in this experiment, I used the above library functions. Especially because of the need for calling the queue and stack in this experiment, I used two libraries, queue and stack , and because of the need for some calculations in the experiment, I used the c math library.

②I defined two global variables r ow and col . Because this experiment is specially used to solve the 8-digit problem, they are both set to 3. My code is more robust, and if the two are directly modified, it is also expected to solve other types of problems.

③ Based on the understanding of the A algorithm and the characteristics of the eight-digit problem, I designed Node to store the "chessboard" of each sentence. s howNode() function is used to show the board layout. gg stores the cost value corresponding to g (x) , and h h stores the heuristic value corresponding to h(x) .

OPEN table , CLOSE table and path table necessary for this experiment. It goes without saying that the OPEN table and CLOSE table are the core of the A algorithm, and the last path is used to store the path results.

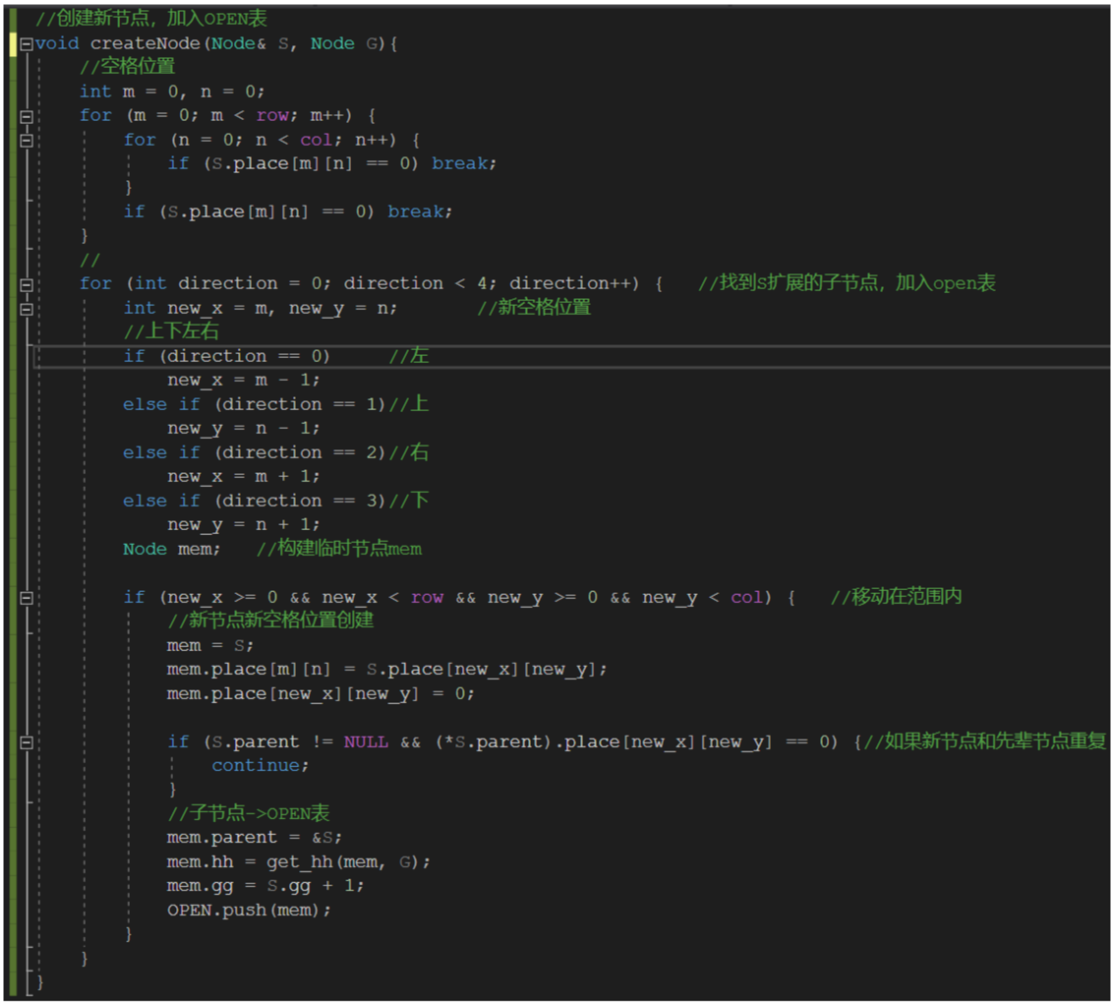
Heuristic function:



```
//启发函数
int get_hh(Node Current, Node End) {
    int countAll = 0;
    for (int i = 0; i < row; i++) {          //扫描current
        for (int j = 0; j < col; j++) {
            if (Current.place[i][j] != End.place[i][j]) {
                for (int k = 0; k < row; k++)        //扫描end
                    for (int w = 0; w < col; w++)
                        if (Current.place[i][j] == End.place[k][w])      //判断current和end相似度
                            countAll = countAll + fabs(i - k * 1.0) + fabs(j - w * 1.0);
            }
        }
    }
    return Current.gg + countAll;
}
```

As shown in the figure, this is the heuristic function in the code. This nested loop can be regarded as two parts, that is, the traversal of each position information of the current game record C current and the traversal of each position of the target game record End . Determine the gap between each position of the current game record and the target game record, and accumulate the difference values of each position, and the data obtained by the final summation is the heuristic function value of the current game record, which means the closeness to the target game record.

Node update function:



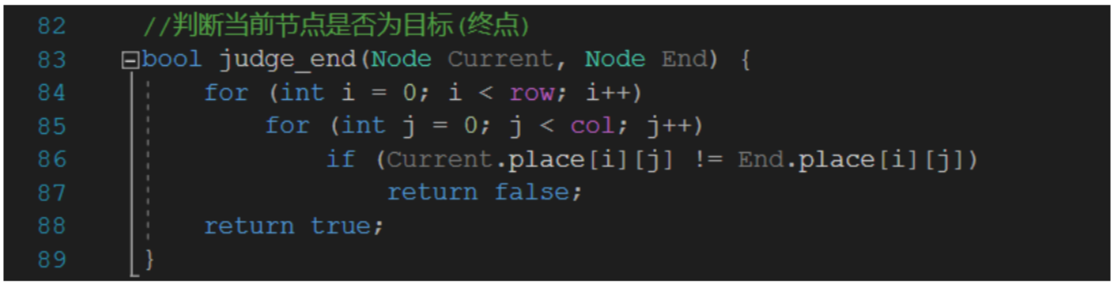
```
//创建新节点，加入OPEN表
void createNode(Node& S, Node G){
    //空格位置
    int m = 0, n = 0;
    for (m = 0; m < row; m++) {
        for (n = 0; n < col; n++) {
            if (S.place[m][n] == 0) break;
        }
        if (S.place[m][n] == 0) break;
    }
    //
    for (int direction = 0; direction < 4; direction++) {    //找到S扩展的子节点，加入open表
        int new_x = m, new_y = n;        //新空格位置
        //上下左右
        if (direction == 0)      //左
            new_x = m - 1;
        else if (direction == 1)//上
            new_y = n - 1;
        else if (direction == 2)//右
            new_x = m + 1;
        else if (direction == 3)//下
            new_y = n + 1;
        Node mem;    //构建临时节点mem

        if (new_x >= 0 && new_x < row && new_y >= 0 && new_y < col) {    //移动在范围内
            //新节点新空格位置创建
            mem = S;
            mem.place[m][n] = S.place[new_x][new_y];
            mem.place[new_x][new_y] = 0;

            if (S.parent != NULL && (*S.parent).place[new_x][new_y] == 0) {//如果新节点和先辈节点重复
                continue;
            }
            //子节点->OPEN表
            mem.parent = &S;
            mem.hh = get_hh(mem, G);
            mem.gg = S.gg + 1;
            OPEN.push(mem);
        }
    }
}
```

My cost function is implied in the node update function. The first is to judge and obtain the "vacancy", that is, the position of the space. In the experiment, I use 0 as a representation. After getting the vacant position, you also know what data can be moved currently, that is, the four numbers above, below, left, and right of the vacant position. Of course, this number may also be a wall and therefore does not exist. We construct a temporary node m em for the number that is feasible to move, that is, the existing number , and simulate the situation after the move. If the new node generated is duplicated with the predecessor node, then this situation is ignored. Otherwise, update the data of m em , update the information of the parent node, cost value and heuristic value, and push it into the OPEN table for subsequent use.

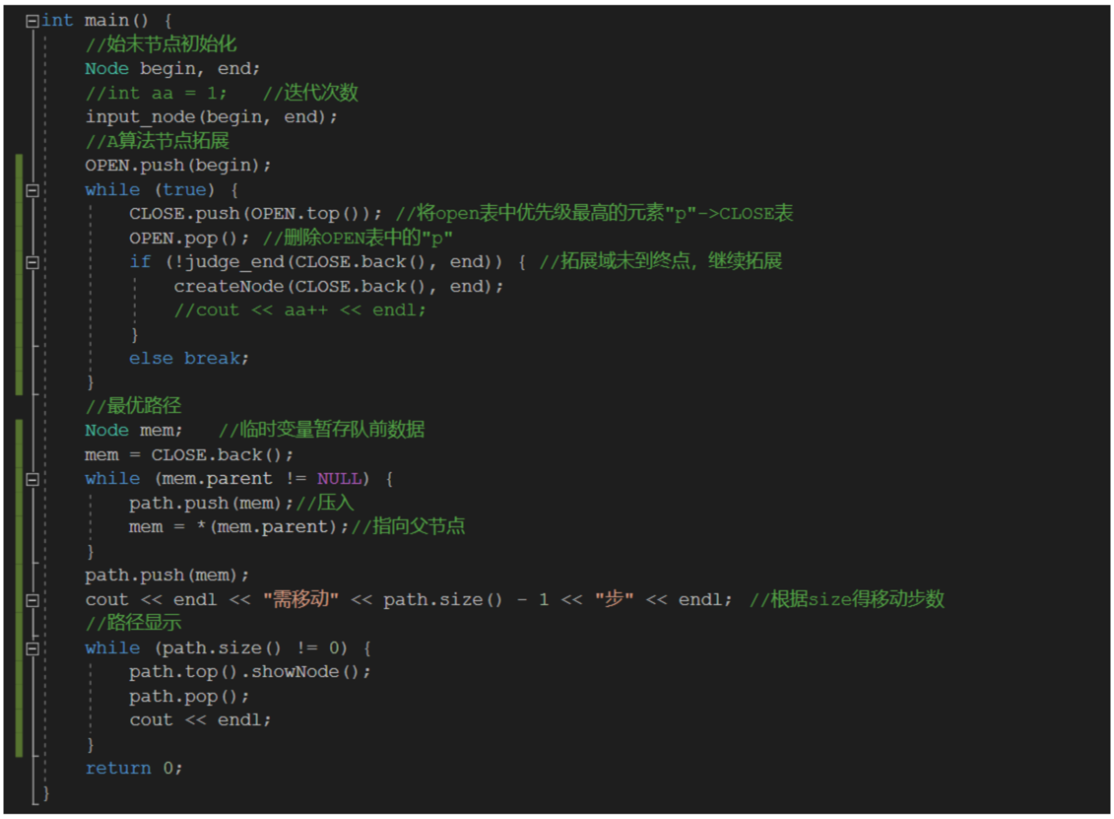
Determine whether the current node reaches the objective function:



```
//判断当前节点是否为目标(终点)
bool judge_end(Node Current, Node End) {
    for (int i = 0; i < row; i++)
        for (int j = 0; j < col; j++)
            if (Current.place[i][j] != End.place[i][j])
                return false;
    return true;
}
```

The implementation idea of this function is very straightforward, that is, use the for loop to judge whether each position of the game record is the same as the target game record, so as to judge whether we have been able to generate a path to the target.

main function:

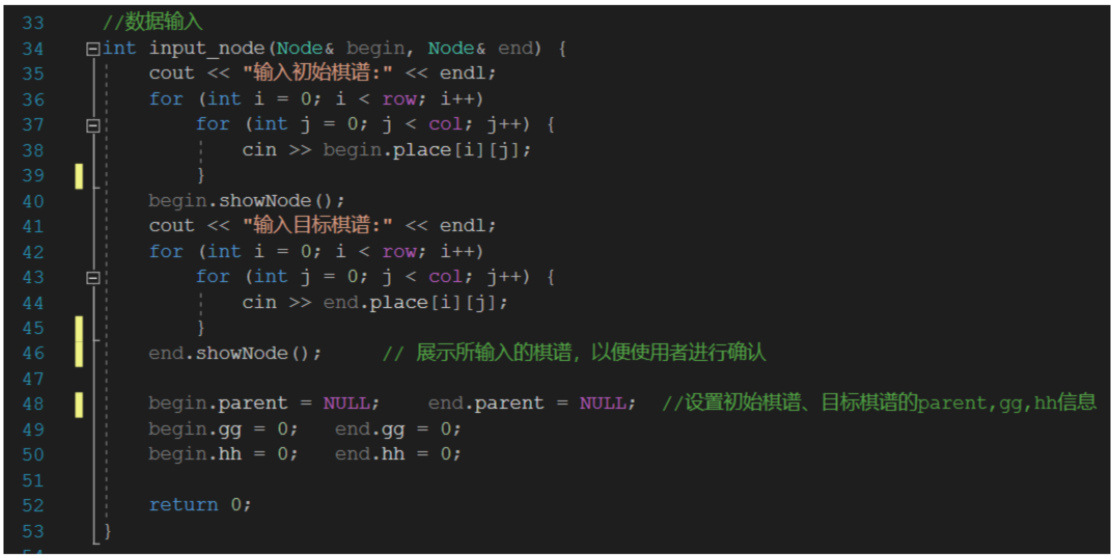


```
int main() {
    //始末节点初始化
    Node begin, end;
    //int aa = 1;   //迭代次数
    input_node(begin, end);
    //A算法节点拓展
    OPEN.push(begin);
    while (true) {
        CLOSE.push(OPEN.top()); //将open表中优先级最高的元素"p"->CLOSE表
        OPEN.pop(); //删除OPEN表中的"p"
        if (!judge_end(CLOSE.back(), end)) { //拓展域未到终点，继续拓展
            createNode(CLOSE.back(), end);
            //cout << aa++ << endl;
        }
        else break;
    }
    //最优路径
    Node mem;   //临时变量暂存队前数据
    mem = CLOSE.back();
    while (mem.parent != NULL) {
        path.push(mem);//压入
        mem = *(mem.parent);//指向父节点
    }
    path.push(mem);
    cout << endl << "需移动" << path.size() - 1 << "步" << endl;  //根据size得移动步数
    //路径显示
    while (path.size() != 0) {
        path.top().showNode();
        path.pop();
        cout << endl;
    }
    return 0;
}
```

Algorithm A in the textbook. This idea is to first input the initial game record and target game record, and then add the chess record node with the highest priority in the open table such as begin to the close table for expansion. The node is updated through the evaluation function obtained by adding the cost function and the heuristic function, and at the same time, it can consider the existing path of the parent node or the existence of the same node. Finally, through continuous expansion until the target node. From this, an optimal path can be generated, which is what this experiment seeks.

Other functions:

data entry function



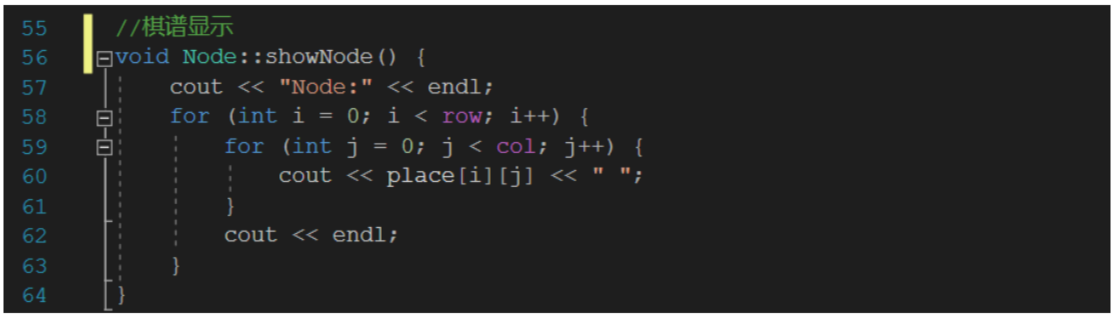
```cpp
    //数据输入
int input_node(Node& begin, Node& end) {
    cout << "输入初始棋谱:" << endl;
    for (int i = 0; i < row; i++)
        for (int j = 0; j < col; j++) {
            cin >> begin.place[i][j];
        }
    begin.showNode();
    cout << "输入目标棋谱:" << endl;
    for (int i = 0; i < row; i++)
        for (int j = 0; j < col; j++) {
            cin >> end.place[i][j];
        }
    end.showNode();      // 展示所输入的棋谱，以便使用者进行确认

    begin.parent = NULL;    end.parent = NULL;   //设置初始棋谱、目标棋谱的parent,gg,hh信息
    begin.gg = 0;   end.gg = 0;
    begin.hh = 0;   end.hh = 0;

    return 0;
}
```

This function is placed at the beginning of the main function, and is used to allow the user to input the initial game record and target game record, output the corresponding result and create the corresponding Node node and initialize the internal information of the node.

Data output function:

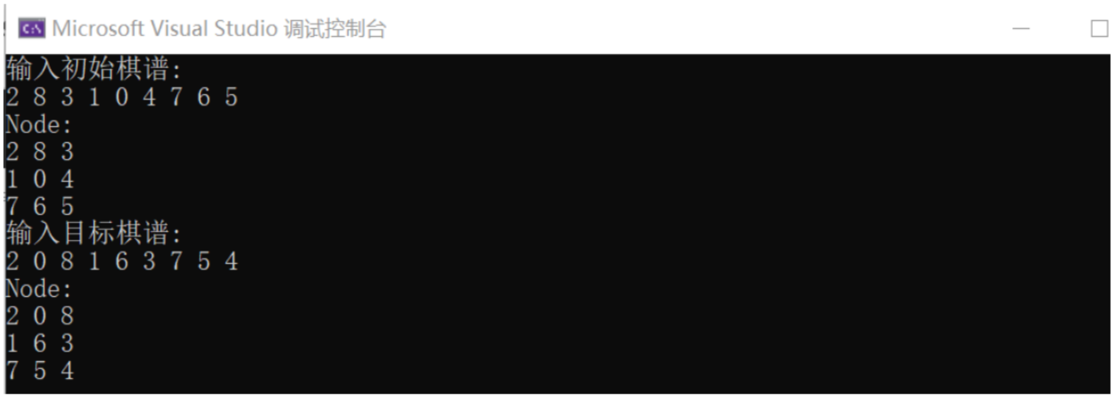


```cpp
  //棋谱显示
void Node::showNode() {
    cout << "Node:" << endl;
    for (int i = 0; i < row; i++) {
        for (int j = 0; j < col; j++) {
            cout << place[i][j] << " ";
        }
        cout << endl;
    }
}
```

This function is nested in Node and is used to output the layout of the game record represented by the Node in a standardized format.

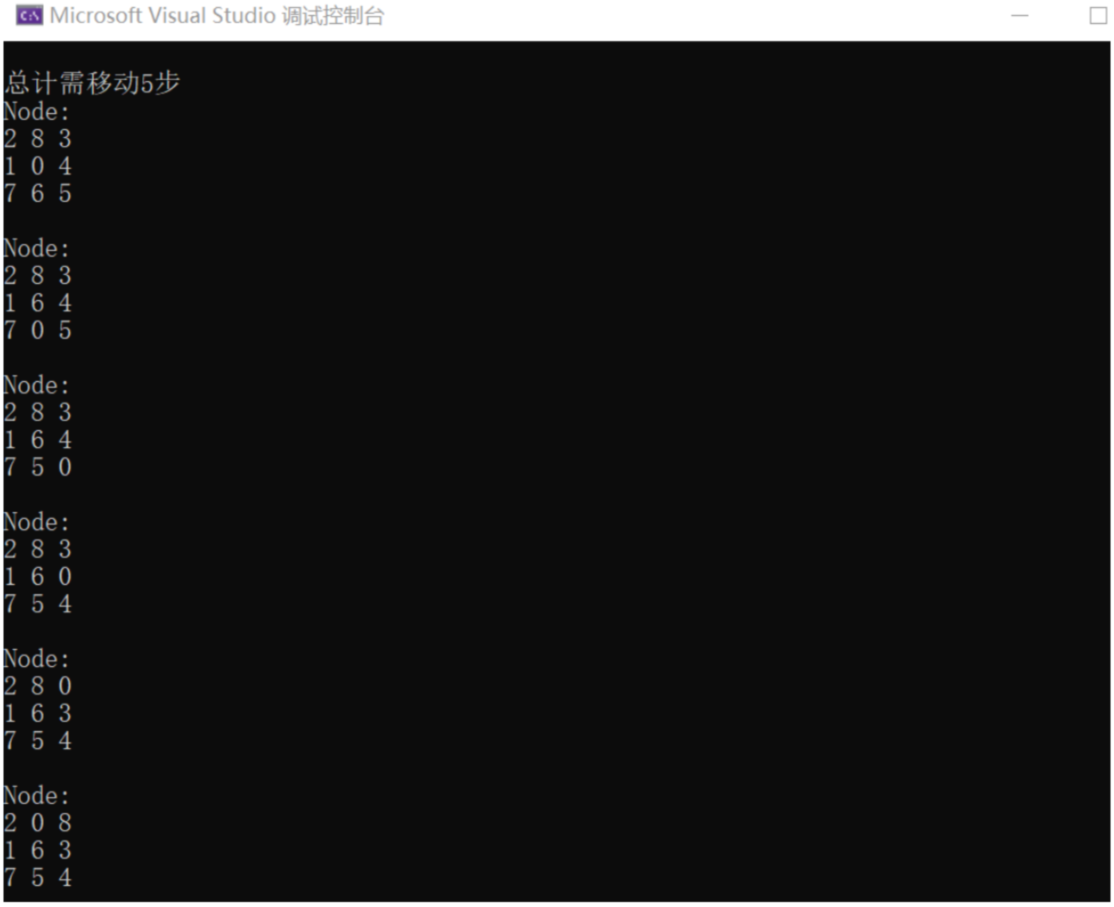
【Experimental Results】

Input data (take the example in book p 55 as an example):



```
输入初始棋谱:
2 8 3 1 0 4 7 6 5
Node:
2 8 3
1 0 4
7 6 5
输入目标棋谱:
2 0 8 1 6 3 7 5 4
Node:
2 0 8
1 6 3
7 5 4
```

System operation output results:



```
总计需移动5步
Node:
2 8 3
1 0 4
7 6 5

Node:
2 8 3
1 6 4
7 0 5

Node:
2 8 3
1 6 4
7 5 0

Node:
2 8 3
1 6 0
7 5 4

Node:
2 8 0
1 6 3
7 5 4

Node:
2 0 8
1 6 3
7 5 4
```

The output result is the same as the example given in the book, the result is correct, and the code runs successfully.

【Experiment Summary】

In this experiment, based on the experimental courseware, I re-understood and learned the 8-digit problem-solving method based on the A algorithm, and realized it through the experimental code. In the implementation process, I think the first step is the most important for the basic setting of experimental data and the design of the experimental framework. For this reason, I judged that the relevant applications of queues and stacks will be used in the experiment, so I used the library to call the related functions of queue and stack , and set up the open table and close table based on the queue, and the path table based on the stack . In order to better realize and utilize the chessboard of each step in 8 digits, I used the structure to design No de , and added parent , gg, hh and other information to it, which is convenient for later calling. In addition, the understanding of the A algorithm is also very important. In addition to writing the heuristic function, it is also necessary to fully consider whether each node already exists in the process of updating nodes, whether the cost function can be directly updated, etc., so that the code is effective. .

After this experiment, I have a deeper understanding of the A algorithm and the 8-digit problem, which have effectively promoted my understanding of artificial intelligence courses and the improvement of my programming experiment ability.