# 8数码问题实验报告

|  |  |
| --- | --- |
| 班级: | 软工2203 |
| 学号: | 221310332 |
| 姓名: | 周立成 |
| 老师: | 肖黎丽 |

## 问题重述:

在3×3的棋盘，摆有八个棋子，每个棋子上标有1至8的某一数字，不同棋子上标的数字不相同。棋盘上还有一个空格，与空格相邻的棋子可以移到空格中。要求解决的问题是：给出一个初始状态和一个目标状态，找出一种从初始转变成目标状态的移动棋子步数最少的移动步骤。

## 启发式函数定义

定义启发式函数,其中是从初始状态到需要进行的移动操作次数,是x状态下错放的棋子数。

用C++描述如下:

1. int get\_heuristic(vector<vector<int>> src, int depth) {

2. int ret = 0;

3. for (int i = 0; i < src.size(); i++) {

4. for (int j = 0; j < src[i].size(); j++) {

5. if (src[i][j] != target[i][j]) {

6. ret++;

7. }

8. }

9. }

10. return ret + depth;

11. }

## 结构设计与分析

* 关于一个节点的定义：

我们认为一个节点不应该只是一个二维数组,还应该包含以下信息:

1. **0（空格）的位置**:这便于移动和交换
2. **当前的递归深度**:便于送入启发式函数以计算值
3. **从哪里来**:通俗的说,如果一个节点刚从上面下来,我们让他再回到上面,无疑是费时的,所以标注从哪里来,可以在判断的时候节省时间
4. **二维数组:**最基本的子结构,用于记录当前的棋盘

所以我们可以设计如下节点:

1. struct node {

2. vector<vector<int>> Node;

3. pair<int, int> zero\_index;//0的位置

4. string From;//看从什么地方来的 UP DOWN RIGHT LEFT

5. int depth{};//递归深度

6. node(vector<vector<int>> &Node, pair<int, int> zero, string From, int depth) {

7. this->Node = Node;

8. this->zero\_index = zero;

9. this->From = From;

10. this->depth = depth;

11. }

12.

13. node() {}

14.

15. };

* Open\_List与Close\_List：根据定义,Open表中的节点应该按照启发式函数的值升序排列,所以我们考虑用set<node>作为数据结构以达到自动排序的功能,排序时需要考虑几个问题:

1. 启发式函数的值是否可能一样
2. 当前的深度是否可能一样

若当前启发式函数的值,和深度均一样set会认为两个节点是“相同的”,故我们将当前的“状态”即是从哪个方向转移到这个节点的加入判断条件,为node重载小于运算符：

1. bool operator < (node a1,node a2)

2. {

3. if( get\_heuristic(a1.Node,a1.depth)== get\_heuristic(a2.Node,a2.depth))

4. {

5. if(a1.depth==a2.depth)

6. {

7. return a1.From<a2.From;

8. }

9. return a1.depth>a2.depth;//节点深的排在前面

10. }

11. else

12. {

13. return get\_heuristic(a1.Node,a1.depth)< get\_heuristic (a2.Node,a2.depth);

14. }

15.

16. }

* Class Eight\_Puzzle:包含Open表和Close表,同时负责初始化

1. Eight\_Puzzle(vector<vector<int>> &src, vector<vector<int>> &tar) {

2. target = tar;

3. pair<int, int> zero = get\_zero\_index(src);

4. int depth = 0;//初始化深度为0

5. node s = node(src, zero, "NULL", depth);

6. Open\_List.insert(s);

7. }

* Solve 函数:负责处理寻找的步骤,每次负责从上(Up),下(Down),左(Left),右(Right)四个方向试探,直到找到为止.

1. void solve() {

2. node current = \*Open\_List.begin();

3. print\_process(current.Node, "Open List");

4. while (!check(current.Node)) {

5. //当前节点不是目标节点,但可以扩展,放入close表

6. Open\_List.erase(Open\_List.begin());

7. print\_process(current.Node, "Close List");

8. Close\_List.insert(current);

9. int z\_row = current.zero\_index.first;

10. int z\_col = current.zero\_index.second;

11. int depth = current.depth + 1;//每一个加入深度均要加1

12. if (current.From != "UP" && current.zero\_index.first + 1 < 3)//判断是不是刚从下面上去

13. {

14. vector<vector<int>> new\_matrix = current.Node;

15. swap(new\_matrix[z\_row][z\_col], new\_matrix[z\_row + 1][z\_col]);

16. string From = "Down";

17. int new\_z\_row = z\_row + 1;

18. node p = node(new\_matrix, make\_pair(new\_z\_row, z\_col), From, depth);

19. Open\_List.insert(p);//加入open 表

20. print\_process(new\_matrix, "Open List");

21. }

22. if (current.From != "Down" && current.zero\_index.first > 0)//判断是不是刚从上面下来

23. {

24. vector<vector<int>> new\_matrix = current.Node;

25. swap(new\_matrix[z\_row][z\_col], new\_matrix[z\_row - 1][z\_col]);

26. string From = "UP";

27. int new\_z\_row = z\_row - 1;

28. node p = node(new\_matrix, make\_pair(new\_z\_row, z\_col), From, depth);

29. Open\_List.insert(p);//加入open 表

30. print\_process(new\_matrix, "Open List");

31. }

32. if (current.From != "Right" && current.zero\_index.second > 0)//判断是不是刚从左面来

33. {

34. vector<vector<int>> new\_matrix = current.Node;

35. swap(new\_matrix[z\_row][z\_col], new\_matrix[z\_row][z\_col - 1]);

36. string From = "Left";

37. int new\_z\_col = z\_col - 1;

38. node p = node(new\_matrix, make\_pair(z\_row, new\_z\_col), From, depth);

39. Open\_List.insert(p);//加入open 表

40. print\_process(new\_matrix, "Open List");

41. }

42.

43. if (current.From != "Left" && current.zero\_index.second + 1 < 3)//判断是不是刚从右面来

44. {

45. vector<vector<int>> new\_matrix = current.Node;

46. swap(new\_matrix[z\_row][z\_col], new\_matrix[z\_row][z\_col + 1]);

47. string From = "Right";

48. int new\_z\_col = z\_col + 1;

49. node p = node(new\_matrix, make\_pair(z\_row, new\_z\_col), From, depth);

50. Open\_List.insert(p);//加入open 表

51. print\_process(new\_matrix, "Open List");

52. }

53. current = \*Open\_List.begin();

54. }

55. cout << "Success!";

56. print\_process(current.Node, "Final Node");

57. }

58. };

* 总体代码包含main函数,输入输出函数见附件或下方



## 实验测试:

### 实验环境:

* CPU: 11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz
* 内存:16GB
* 操作系统:WINDOWS 11 22H2
* 编译工具:MVSC 17.0
* 编程语言:C++
* 编程环境:CLION 2023.2

### 测试

* 测试数据

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

* 输出结果:

输入3\*3的矩阵src

**2 8 3**

**1 6 4**

**7 0 5**

输入3\*3的矩阵tar

**1 2 3**

**8 0 4**

**7 6 5**

-----------------------------------------

The matrix:

2 8 3

1 6 4

7 0 5

has been put in to theOpen List

------------------------------------------

-----------------------------------------

The matrix:

2 8 3

1 6 4

7 0 5

has been put in to theClose List

------------------------------------------

-----------------------------------------

The matrix:

2 8 3

1 0 4

7 6 5

has been put in to theOpen List

------------------------------------------

-----------------------------------------

The matrix:

2 8 3

1 6 4

0 7 5

has been put in to theOpen List

------------------------------------------

-----------------------------------------

The matrix:

2 8 3

1 6 4

7 5 0

has been put in to theOpen List

------------------------------------------

-----------------------------------------

The matrix:

2 8 3

1 0 4

7 6 5

has been put in to theClose List

------------------------------------------

-----------------------------------------

The matrix:

2 0 3

1 8 4

7 6 5

has been put in to theOpen List

------------------------------------------

-----------------------------------------

The matrix:

2 8 3

0 1 4

7 6 5

has been put in to theOpen List

------------------------------------------

-----------------------------------------

The matrix:

2 8 3

1 4 0

7 6 5

has been put in to theOpen List

------------------------------------------

-----------------------------------------

The matrix:

2 8 3

0 1 4

7 6 5

has been put in to theClose List

------------------------------------------

-----------------------------------------

The matrix:

2 8 3

7 1 4

0 6 5

has been put in to theOpen List

------------------------------------------

-----------------------------------------

The matrix:

0 8 3

2 1 4

7 6 5

has been put in to theOpen List

------------------------------------------

-----------------------------------------

The matrix:

2 0 3

1 8 4

7 6 5

has been put in to theClose List

------------------------------------------

-----------------------------------------

The matrix:

0 2 3

1 8 4

7 6 5

has been put in to theOpen List

------------------------------------------

-----------------------------------------

The matrix:

2 3 0

1 8 4

7 6 5

has been put in to theOpen List

------------------------------------------

-----------------------------------------

The matrix:

0 2 3

1 8 4

7 6 5

has been put in to theClose List

------------------------------------------

-----------------------------------------

The matrix:

1 2 3

0 8 4

7 6 5

has been put in to theOpen List

------------------------------------------

-----------------------------------------

The matrix:

1 2 3

0 8 4

7 6 5

has been put in to theClose List

------------------------------------------

-----------------------------------------

The matrix:

1 2 3

7 8 4

0 6 5

has been put in to theOpen List

------------------------------------------

-----------------------------------------

The matrix:

1 2 3

8 0 4

7 6 5

has been put in to theOpen List

------------------------------------------

Success!-----------------------------------------

The matrix:

1 2 3

8 0 4

7 6 5

has been put in to theFinal Node

------------------------------------------

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

* 分析:

将输出结果整理至下图:



**可知运行结果正确,和上课的PPT一致!**

## 改进与不足

1. 只能判断3阶,可以将solve函数稍加改进（边界判定情况的改进）将其推广至3阶以上
2. 呈现方式不美观,需要人为的手工处理,望以后能用可视化的方式呈现结果