**《程序设计综合实践》报告**

题 目： 重排九宫格

专 业： 信息安全

班 级： 22信安2

学生姓名： 张忆周

学 号： 2210541109

指导教师： 杜江毅

2023年 12 月 15 日

**1、小组成员及分工**

班级：22信安2 学号：2210541109 姓名：张忆周

分工: 对选题进行分析与设计，完成主要的程序开发与实现，对系统求解算法进行优化，测试样例的编写，完成PPT的文本设计。

班级：22信安2 学号：2210300411 姓名：罗锦宇

分工: 查阅文献，进行系统的分析与设计，对系统不可解情况进行优化，测试结果的统计，准备答辩材料。

**2、系统分析**

该题是一个经典的状态搜索问题，分析其有两个基本问题需要解决:

(1) 如何将每个状态存储下来？

(2) 如何求得最优解？

**3、系统设计**

(1)如何将每个状态存储下来？

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

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

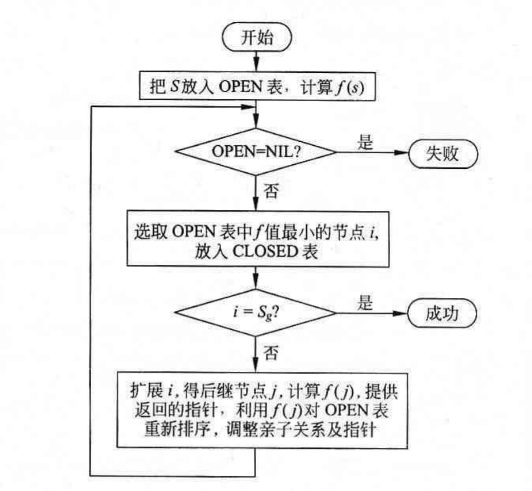
将空格位置视为0，然后以行优先展开，得到一维数组，则可将其视为一个0~8的排列，总共有9!=362880种状态，每种状态可由康托展开表示。

(2) 如何求得最优解？

采用BFS，从起始状态开始，每次移动一步，扩展出新的节点。每次都尝试访问同一层的节点，并拓展出新节点。如果同一层都访问完了，再访问下一层。这样做的结果是，BFS 算法找到的路径是从起始状态开始的最少步。

(3)算法优化

通过采用曼哈顿距离估价的A\*算法求最优解，A\*算法是对BFS添加估价函数对每次生成新的状态进行估价，价值越小表示越接近终态，每次都取出最小价值的状态(由小根堆实现),对其进行扩展。



估价函数:

从初始状态开始的距离函数 ,这里采用搜索深度。

到最终状态的距离函数，这里采用现态与终态所有相同数字的曼哈顿距离之和。

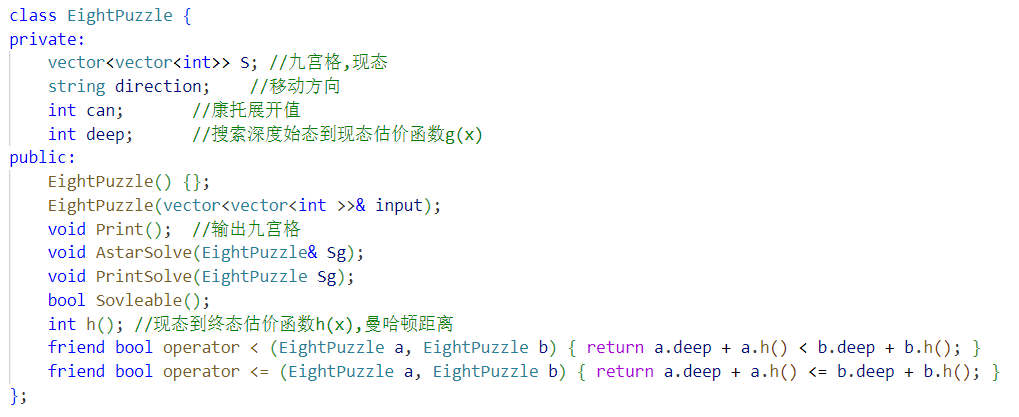
**4、实现及测试**

通过康托展开将九宫格状态映射为数字存储:

文本

描述已自动生成

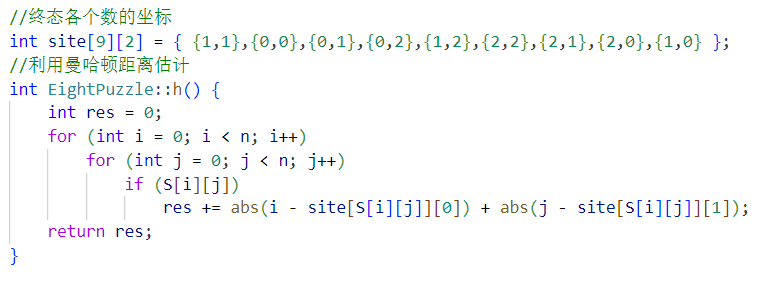
九宫格类:



以曼哈顿距离的A\*算法求最优解:

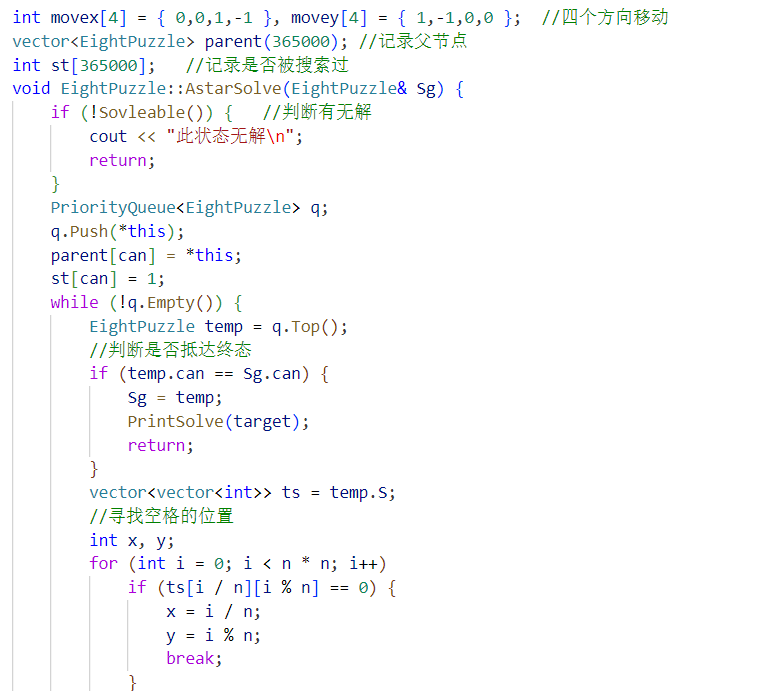
曼哈顿距离:

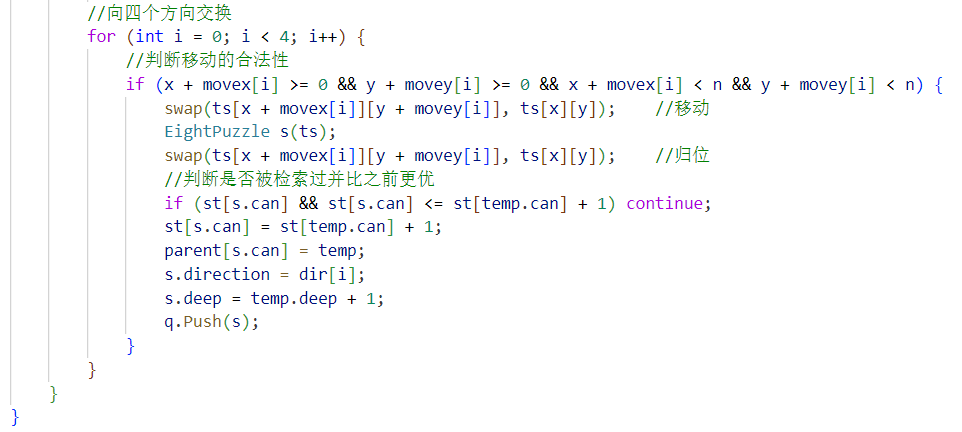
注：这里最开始采用的是未归位数字数量作为h(x)，但由测试结果来看(详见后面的测试统计表格)，时间比BFS更慢了，产生了负优化，分析并查阅资料发现是由于通过未归位数字数量来估价的问题，并不足以判断最优，改成曼哈顿距离估价解决。



A\*算法:

注:这里对A\*算法的实现有一个点要注意，由于不能保证估价函数小状态的一定是最优解中的一步，所以在原本判定是否有重复节点的基础上，当节点重复时还需判断当前是否比之前的步数更少，以此才能保证是最优解。

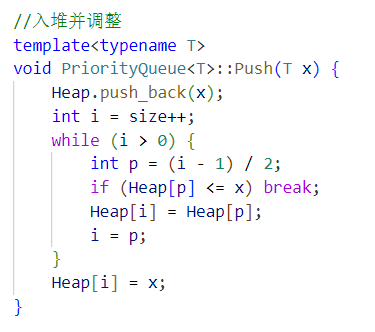


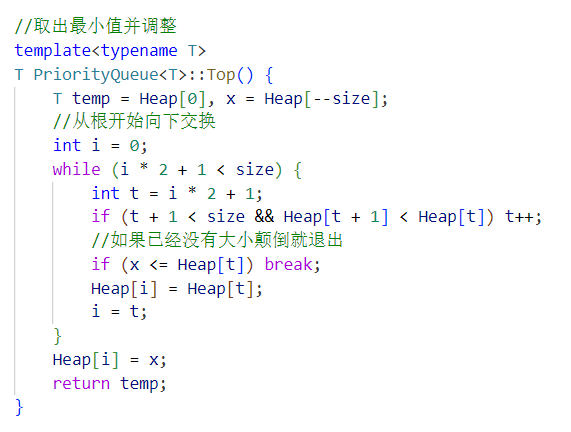


由于需要每次取出估价最小的状态，故可采用小根堆优化。

优先队列类(小根堆)：

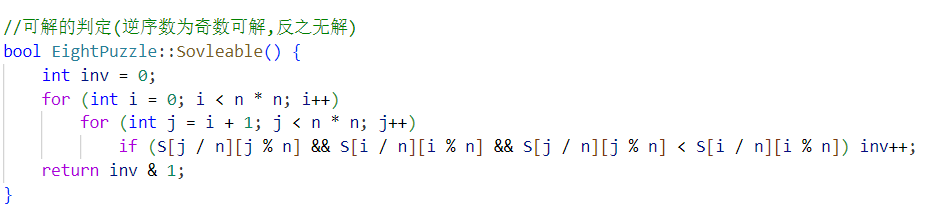




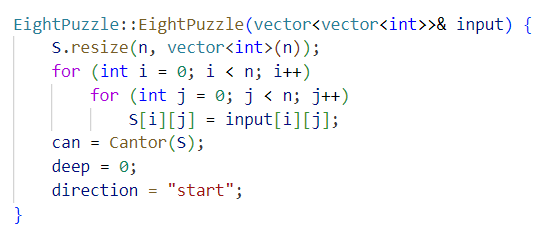


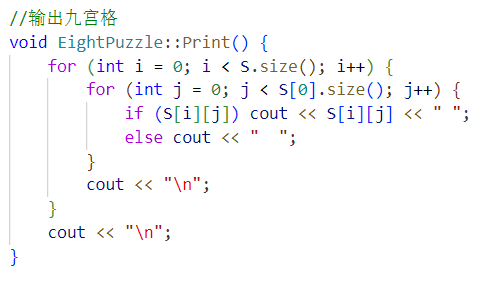
无解的判定:

对于该题的要求，n=3为奇数，且终态的逆序数为奇数，所以当始态的逆序数也为奇数时有解。



EightPuzzle类中其他函数的实现:

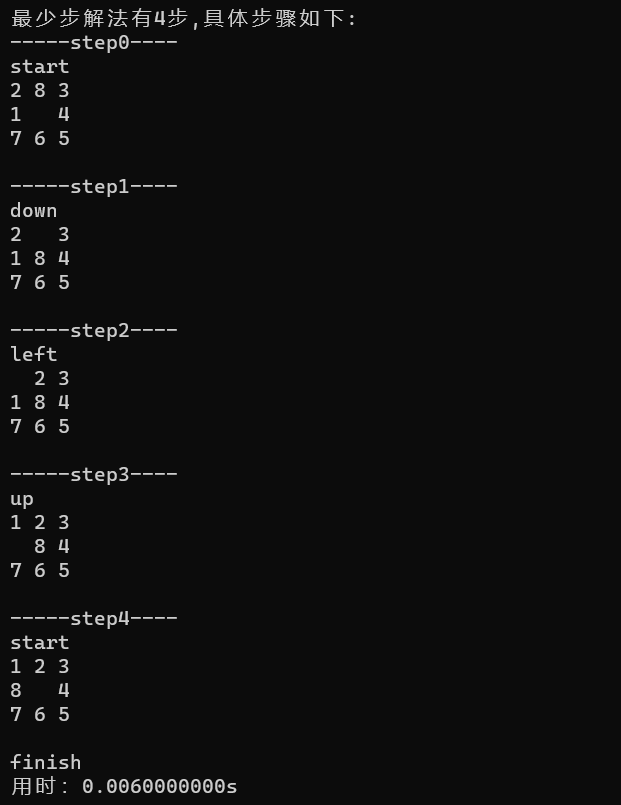




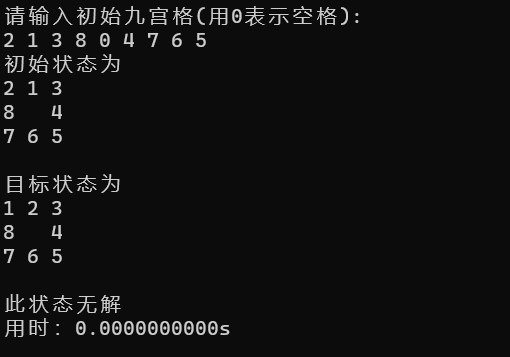


测试用例:

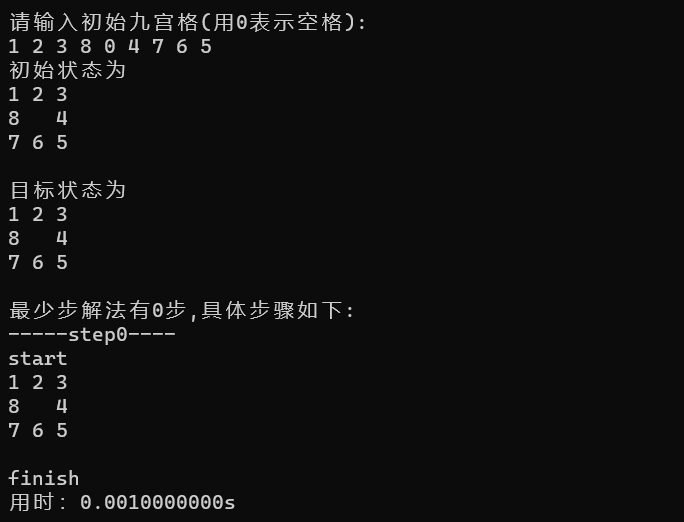
(1)题目样例: 2 8 3 1 0 4 7 6 5



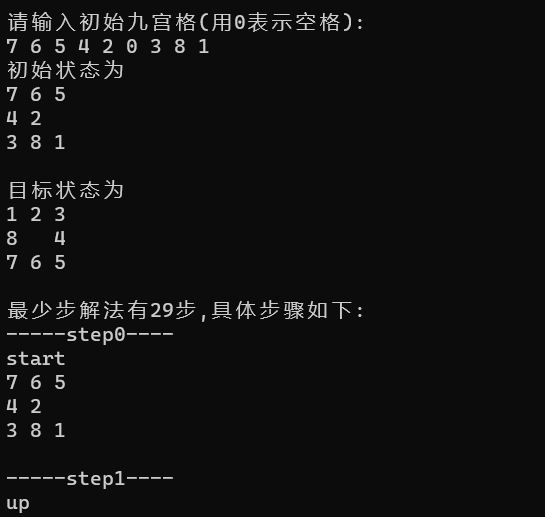
(2)无解样例：2 1 3 8 0 4 7 6 5

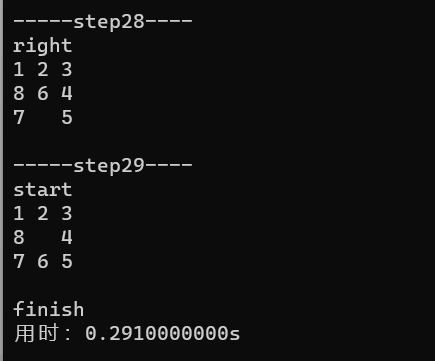


(3)终态样例: 1 2 3 8 0 4 7 6 5

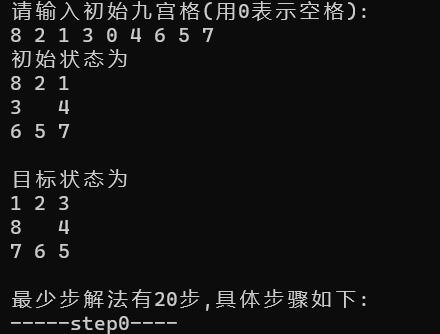


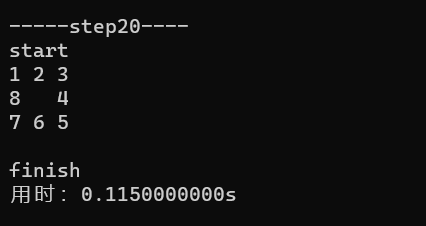
(4)随机样例: 7 6 5 4 2 0 3 8 1



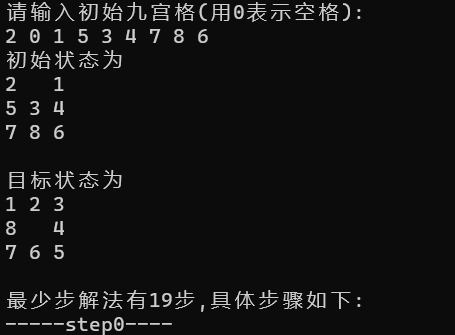


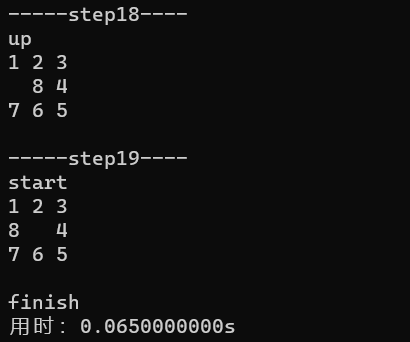
(5)随机样例:8 2 1 3 0 4 6 5 7





(6)随机样例: 2 0 1 5 3 4 7 8 6





测试结果总览(对每个测试样例测10求平均值):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 测试用例 | 特性 | 人工组（为张忆周的求解速度） | BFS运行时间 | A\*算法(未归位数字数量) | A\*算法(曼哈顿距离)+无解判定 |
| 2 8 3 1 0 4 7 6 5 | 题目样例，最优4步 | 2.323s  4步 | 0.001s  4步 | 0.724s  4步 | 0.009s  4步 |
| 1 2 3 8 0 4 7 6 5 | 始态即终态 | 0.126s  0步 | 0.000s  0步 | 0.002s  0步 | 0.001s  0步 |
| 2 1 3 8 0 4 7 6 5 | 无解 | 约10s  无解 | 9.46s  无解 | 50.624s  无解 | 0.000s  无解 |
| 7 6 5 4 2 0 3 8 1 | 随机样例，最优29步 | 11.284s  79步 | 4.231s  29步 | 25.761s  29步 | 0.924s  29步 |
| 8 2 1 3 0 4 6 5 7 | 随机样例，最优20步 | 6.397s  45步 | 0.982s  20步 | 1.639s  20步 | 0.189s  20步 |
| 2 0 1 5 3 4 7 8 6 | 随机样例，最优19步 | 9.637s  66步 | 0.703s  19步 | 0.942s  19步 | 0.098s  19步 |

**5、总结**

（1）基本实现对该问题的求解，并且同过该问题的求解，学习到了A\*算法的原理，并通过该算法对程序进行尽可能的优化。

（2）在使用A\*算法求解问题时，要采用合适的估计函数，否则可能出现负优化。

（3）由于时间关系，未能完成将其图形化展示并动态演示最少步过程，比较可惜。

**6、参考资料**

(1) 15-puzzle- OI Wiki (oi-wiki.org) (https://oi-wiki.org/misc/15-p

uzzle/):参考了此处关于15-puzzle问题求解的思路

(2)康托展开 - OI Wiki (oi-wiki.org)( [https://oi-wiki.org/math/ combinatorics/cantor/](https://oi-wiki.org/math/%20combinatorics/cantor/)):该文章中给出了康托展开的定义及其实现原理

(3)How to check if an instance of 15 puzzle is solvable?( <https://www.geeksforgeeks.org/check-instance-15-puzzle-solvable/>):该文章中给出了此类问题可解情况的判定方法

(4)A\* - OI Wiki (oi-wiki.org)( https://oi-wiki.org/search/astar/):该文章中给出了A\*算法的实现原理