八数码搜索问题

一、 问题介绍

八数码问题（又称为九宫排字问题）是人工智能中比较出名的难题之一。在一个3×3的方格中，放有八个数字，剩下的第九个数字为空，每次可以将空格与其周围的数字位置交换。本实验中给出的目标位置为

1 2 3

4 5 6

7 8 0 （0为空格）

通过BFS、DFS、UCS三种搜索方式解决八数码问题。

二、问题求解

建立一个搜索树，根节点为初始给定的状态，并将位置转换后的结果作为其子节点，不断地重复建立。采用BFS、DFS、 UCS方法依次搜索该搜索树，直至达到目标状态。

其中BFS算法利用队列实现、DFS算法利用栈实现、UCS算法利用优先队列实现。

1. 是否有解

八数码问题存在无解的可能性。将八数码的一个状态表示成一维的形式，求出除0之外所有数字的逆序数之和，也就是每个数字前面比它大的数字的个数的和，称为这个状态的逆序。如果两个状态的逆序奇偶性相同，则两个状态可以相互到达，即初始状态到目标状态有解，否则不可相互到达，初始状态到目标状态无解。

2. 避免重复的状态

八数码问题状态众多，但是仍不可避免可能在搜索的过程出现重复状态。重复状态的出现即意味着下一步的搜索是无效的，应该回退到上一步重新搜索，避免代码的循环。这里采用了数据结构map，map分为键和值两个部分，键是唯一的，可以将所有搜索过的状态加入map中，保证状态的不重复。

三、具体算法与运行结果

1. BFS（广度优先搜索）

BFS使用数据结构队列实现。先将根节点入队，然后取出队首的节点，然后将其子节点依次入队，如此不断重复进行搜索。

得到的结果如下：

初始状态	求解过程	步数

初始状态1	step: 000: 5 1 2 6 3 0 4 7 8 step: 001: 5 1 2 6 0 3 4 7 8 step: 002: 5 1 2 0 6 3 4 7 8 step: 003: 0 1 2 5 6 3 4 7 8 step: 004: 1 0 2 5 6 3 4 7 8 step: 005: 1 2 0 5 6 3 4 7 8 step: 006: 1 2 3 5 6 0 4 7 8 step: 007: 1 2 3 5 0 6 4 7 8 step: 008: 1 2 3 0 5 6 4 7 8 step: 009: 1 2 3 4 5 6 0 7 8 step: 010: 1 2 3 4 5 6 7 0 8 step: 011: 1 2 3 4 5 6 7 8 0	11步
初始状态2	no solution for the initial state:	无解
初始状态3	step: 006: 1 4 2 6 3 0 7 8 5 step: 007: 1 4 2 6 0 3 7 8 5 step: 008: 1 4 2 0 6 3 7 8 5 step: 009: 1 4 2 7 6 3 0 8 5 step: 010: 1 4 2 7 6 3 8 0 5 step: 011: 1 4 2 7 0 3 8 6 5 step: 012: 1 0 2 7 4 3 8 6 5 step: 013: 1 2 0 7 4 3 8 6 5 step: 014: 1 2 3 7 4 0 8 6 5 step: 015: 1 2 3 7 4 5 8 6 0 step: 016: 1 2 3 7 4 5 8 0 6 step: 017: 1 2 3 7 4 5 0 8 6 step: 018: 1 2 3 0 4 5 7 8 6 step: 019: 1 2 3 4 0 5 7 8 6 step: 020: 1 2 3 4 5 0 7 8 6 step: 021: 1 2 3 4 5 6 7 8 0	21步
初始状态4	no solution for the initial state	无解

初始状态5	step: 007: 1 7 3 4 0 8 5 6 2 step: 008: 1 0 3 4 7 8 5 6 2 step: 009: 0 1 3 4 7 8 5 6 2 step: 010: 4 1 3 0 7 8 5 6 2 step: 011: 4 1 3 7 0 8 5 6 2 step: 012: 4 1 3 7 6 8 5 0 2 step: 013: 4 1 3 7 6 8 5 2 0 step: 014: 4 1 3 7 6 0 5 2 8 step: 015: 4 1 3 7 0 6 5 2 8 step: 016: 4 1 3 7 2 6 5 0 8 step: 017: 4 1 3 7 2 6 0 5 8 step: 018: 4 1 3 0 2 6 7 5 8 step: 019: 0 1 3 4 2 6 7 5 8 step: 020: 1 0 3 4 2 6 7 5 8 step: 021: 1 2 3 4 0 6 7 5 8 step: 022: 1 2 3 4 5 6 7 0 8 step: 023: 1 2 3 4 5 6 7 8 0	23步

2. DFS（深度优先搜索）

DFS算法使用数据结构栈实现。先将根节点压入栈中，再将栈顶元素出栈，并将其子节点按顺序依次压入栈中，不断重复，直至搜索到目标状态。

优先级分别是上、右、左、下。

初始状态	求解过程	步数
初始状态1	step: 347: 1 2 3 6 8 4 7 5 0 step: 348: 1 2 3 6 8 0 7 5 4 step: 349: 1 2 3 6 0 8 7 5 4 step: 350: 1 2 3 0 6 8 7 5 4 step: 351: 1 2 3 7 6 8 0 5 4 step: 352: 1 2 3 7 6 8 5 0 4 step: 353: 1 2 3 7 6 8 5 4 0 step: 354: 1 2 3 7 6 0 5 4 8 step: 355: 1 2 3 7 0 6 5 4 8 step: 356: 1 2 3 7 4 6 5 0 8 step: 357: 1 2 3 7 4 6 0 5 8 step: 358: 1 2 3 0 4 6 7 5 8 step: 359: 1 2 3 4 0 6 7 5 8 step: 360: 1 2 3 4 5 6 7 0 8 step: 361: 1 2 3 4 5 6 7 8 0	361步
初始状态2	no solution for the initial state:	无解

初始状态3	step: 709: 2 3 0 1 6 8 5 4 7 step: 710: 2 0 3 1 6 8 5 4 7 step: 711: 0 2 3 1 6 8 5 4 7 step: 712: 1 2 3 0 6 8 5 4 7 step: 713: 1 2 3 5 6 8 0 4 7 step: 714: 1 2 3 5 6 8 4 0 7 step: 715: 1 2 3 5 6 8 4 7 0 step: 716: 1 2 3 5 6 0 4 7 8 step: 717: 1 2 3 5 0 6 4 7 8 step: 718: 1 2 3 0 5 6 4 7 8 step: 719: 1 2 3 4 5 6 0 7 8 step: 720: 1 2 3 4 5 6 7 0 8 step: 721: 1 2 3 4 5 6 7 8 0	721步
初始状态4	no solution for the initial state	无解
初始状态5	step: 526: 1 2 3 7 8 0 5 6 4 step: 527: 1 2 3 7 0 8 5 6 4 step: 528: 1 2 3 7 6 8 5 0 4 step: 529: 1 2 3 7 6 8 5 4 0 step: 530: 1 2 3 7 6 0 5 4 8 step: 531: 1 2 3 7 0 6 5 4 8 step: 532: 1 2 3 0 7 6 5 4 8 step: 533: 1 2 3 5 7 6 0 4 8 step: 534: 1 2 3 5 7 6 4 0 8 step: 535: 1 2 3 5 0 6 4 7 8 step: 536: 1 2 3 0 5 6 4 7 8 step: 537: 1 2 3 4 5 6 0 7 8 step: 538: 1 2 3 4 5 6 7 0 8 step: 539: 1 2 3 4 5 6 7 8 0	509步

3. UCS（代价一致性搜索）

UCS算法采用数据结构优先队列实现，先将根节点压入队中，队首出队，然后根据其代价大小将所有的子节点压入队中，依次重复 。

初始状态	求解过程	步数

初始状态1	step: 000: 5 1 2 6 3 0 4 7 8 step: 001: 5 1 2 6 0 3 4 7 8 step: 002: 5 1 2 0 6 3 4 7 8 step: 003: 0 1 2 5 6 3 4 7 8 step: 004: 1 0 2 5 6 3 4 7 8 step: 005: 1 2 0 5 6 3 4 7 8 step: 006: 1 2 3 5 6 0 4 7 8 step: 007: 1 2 3 5 0 6 4 7 8 step: 008: 1 2 3 0 5 6 4 7 8 step: 009: 1 2 3 4 5 6 0 7 8 step: 010: 1 2 3 4 5 6 7 0 8 step: 011: 1 2 3 4 5 6 7 8 0	11步
初始状态2	no solution for the initial state:	无解
初始状态3	step: 003: 1 4 2 6 3 5 0 7 8 step: 004: 1 4 2 6 3 5 7 0 8 step: 005: 1 4 2 6 3 5 7 8 0 step: 006: 1 4 2 6 3 0 7 8 5 step: 007: 1 4 2 6 0 3 7 8 5 step: 008: 1 4 2 0 6 3 7 8 5 step: 009: 1 4 2 7 6 3 0 8 5 step: 010: 1 4 2 7 6 3 8 0 5 step: 011: 1 4 2 7 0 3 8 6 5 step: 012: 1 0 2 7 4 3 8 6 5 step: 013: 1 2 0 7 4 3 8 6 5 step: 014: 1 2 3 7 4 0 8 6 5 step: 015: 1 2 3 7 4 5 8 6 0 step: 016: 1 2 3 7 4 5 8 0 6 step: 017: 1 2 3 7 4 5 0 8 6 step: 018: 1 2 3 0 4 5 7 8 6 step: 019: 1 2 3 4 0 5 7 8 6 step: 020: 1 2 3 4 5 0 7 8 6 step: 021: 1 2 3 4 5 6 7 8 0	21步
初始状态4	no solution for the initial state	无解

初始状态5	step: 006: 1 7 3 0 4 8 5 6 2 step: 007: 1 7 3 4 0 8 5 6 2 step: 008: 1 7 3 4 6 8 5 0 2 step: 009: 1 7 3 4 6 8 5 2 0 step: 010: 1 7 3 4 6 0 5 2 8 step: 011: 1 7 3 4 0 6 5 2 8 step: 012: 1 0 3 4 7 6 5 2 8 step: 013: 0 1 3 4 7 6 5 2 8 step: 014: 4 1 3 0 7 6 5 2 8 step: 015: 4 1 3 7 0 6 5 2 8 step: 016: 4 1 3 7 2 6 5 0 8 step: 017: 4 1 3 7 2 6 0 5 8 step: 018: 4 1 3 0 2 6 7 5 8 step: 019: 0 1 3 4 2 6 7 5 8 step: 020: 1 0 3 4 2 6 7 5 8 step: 021: 1 2 3 4 0 6 7 5 8 step: 022: 1 2 3 4 5 6 7 0 8 step: 023: 1 2 3 4 5 6 7 8 0	23步

四、算法比较

本次编程作业使用了BFS、DFS、UCS三种方法，最终都得到了目标解。

根据对比三种方法的输出情况及调试过程可以发现：BFS和UCS算法的步骤差不多，DFS相比之下效率较低，也慢了很多。

可以看出三种算法的优缺点。

BFS：广度遍历算法速度和代价一致性速度差不多，因为UCS算法本质上也是在BFS基础上加入了一个代价，根据代价大小选择了优先队列，所以搜索方式相似，但是在代价明显的时候更加有利于节省成本。

DFS：深度遍历是一直朝着一个方向搜索，直至找到结果或者发现找到叶子节点再返回。而本题目中树的深度比较大，所以导致可能搜索得到的结果也是比较长的。

UCS：代价一致性算法是通过优先队列，循着最小代价进行搜索，其速度在这三者中最快，效率也最高。对于一些实际应用中的成本问题，UCS应该相比于BFS更加优秀，更利于控制成本。