**数码搜索问题**

作者：王润洲 单位：南开大学 学号：2013467

1. **深度、宽度优先遍历**

**（一）基本问题分析**

1. 如何表示九宫格（n×n宫格）？

使用二维数组同样需要解决移动问题，不如使用一维数组储存数据，还可以缩短比较时长；实现方法Move()，共计四种移动方向，在移动前需要确保该方向可以移动。

譬如：如下图所示的九宫格便可表示为[1,2,3,4,5,6,7,8,0]。



1. 具体如何在该宫格内移动？

在一个大型表格内移动，有以下三种移动方式：

（1）空单元格和“墙”不相邻，有四种移动选择

（2）空单元格和一面“墙”相邻，有三种移动选择

（3）空单元格和两面“墙”相邻，有两种移动选择

当然，由于是搜索，如果该步搜索并非第一步，那么还要除去先前的移动，即选择数减少一。此外，由于四面墙的不同，实际操作起来会需要更细致的分类。

算法实现了四个方法：MoveDown()、MoveUp()、MoveLeft()和MoveRight()，分别对应0号单元格的向下、向上、向左和向右，这四个方法只负责移动数码，具体的限制在搜索函数内具体判断。

1. 如何记录已经试验过的位置？

在以空间换时间的版本内，记录状态的方法是设置一个足够大的布尔数组（864197532字节），初始化为0，当方法试验过该位置时，将该bool值置1；因为九宫格是用一维数组表示的，对于某种状态，只需要将该数组转换为对应的实数，即可记录当前状态对应的数组位置，当然，该实数需要首先减去基数12345678，如状态812345670对应bool[812345670-12345678]。

在以时间换空间的版本内，记录状态的方法是设置一个相对较小（400000）的布尔数组，以及一个对应大小的整型数组，在某个状态进队时，需要在整形数组内二分查找该状态的位置，然后将对应的布尔数组位置置一即可。

1. 具体实现

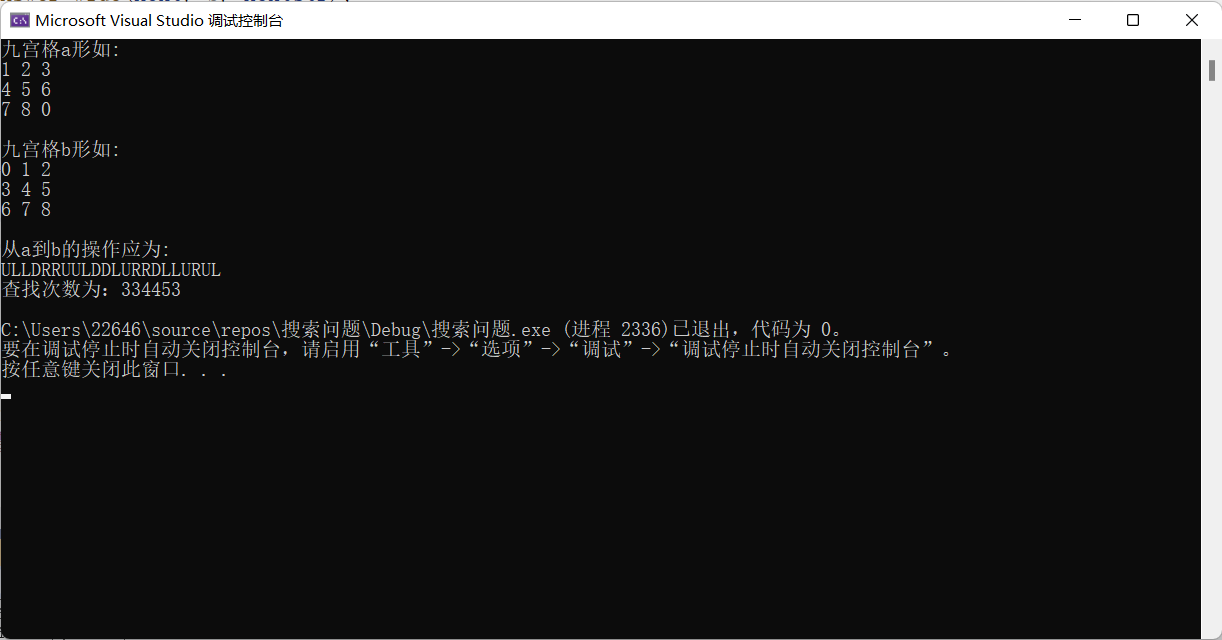
这一部分实现的完整代码请见function1.cpp和function2.cpp，前者是以空间换时间，后者是以时间换空间。方法记录了正确答案的运行逻辑，如DULR代表向下、向上、向左、向右移动0即可得到答案，全局变量globalnum记录了搜索的次数，它跟在正确答案之后输出。

1. **两种实现**
2. 空间换时间

function1.cpp实现了以空间换时间。

搜索问题需要记录每一种已经遍历过的情况，譬如状态“123456780”已经被尝试过，后续搜索必须跳过这一节点，否则很有可能陷入无限循环；function1使用数组记录尝试过的状态，其实现了一个效率很高的算法，能够将char数组转换成对应的整形，从而对应一个地址。

但需要注意的是，这种方法增大了内存消耗：九宫格搜索理应只有9！种状态（小于400000），但这种索引方法却需要876543210-12345678=864,197,562种状态，内存消耗扩大了数百倍，但无需查找对应的位置，因此是货真价实的“以空间换时间”，它的运行效率要比function2高许多。



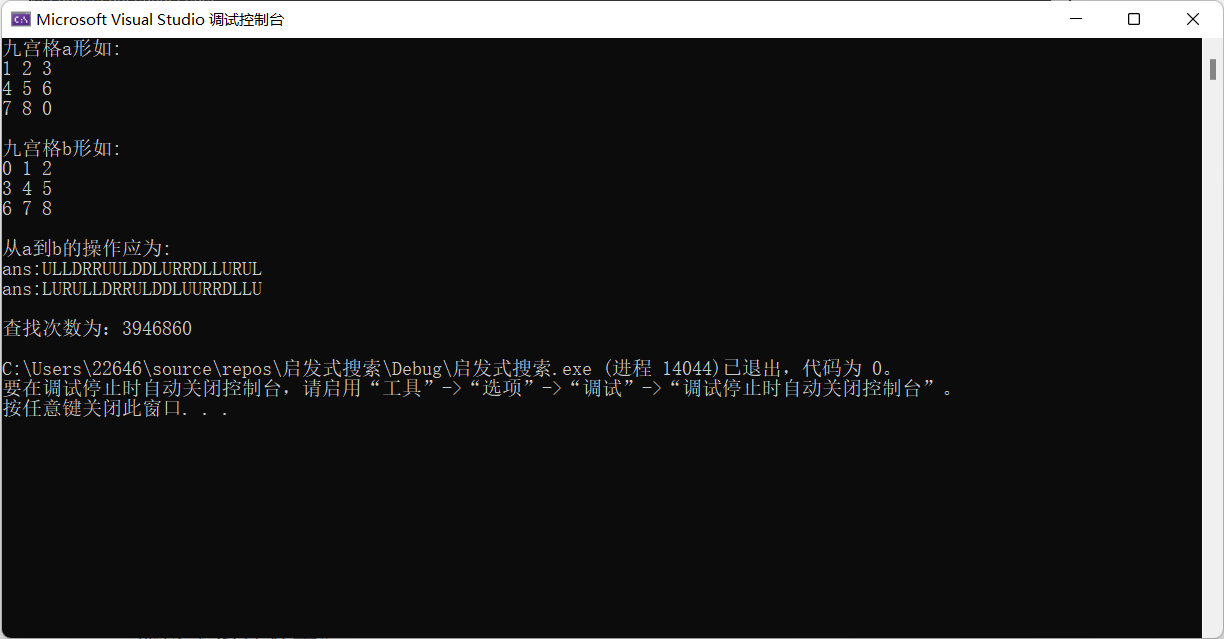
function1 运行截图

1. 时间换空间

function2.cpp实现了以时间换空间。

与function1恰恰相反，function2只开辟了大小为四十万（事实上只需要比37万略大即可）的整形数组以及同等大小的布尔数组用以记录全部状态，但这就要求方法在整形数组内查找状态，随后对布尔类型进行操作；虽然空间得到了节约，却拖慢了运行效率。

其余部分的实现与function1类似，详情请见代码。



function2 运行截图

1. **启发式搜索**
2. 基本实现

启发式搜索function3继承了function1的基本代码，在宽度优先遍历的基础上配置了两个启发式函数，分别对应方法getAnswer\_deque()和getAnswer\_nonsense()。

1. 距离正确答案最近的节点优先

getAnswer\_deque()实现了一种启发式操作，即利用双端队列，让当前距离正确答案最近的节点优先深入，关键函数为distance()，用于计算当前节点与目标节点的“距离”，譬如“123456780”和“012345678”的距离是8，而“123456780”和“123456708”的距离是1。这里，“距离”的含义是指对于字串a，至少需要移动0多少次，才能将其变成目标字串b。

因为计算实际的“距离”是不现实的，选择用更简单的方式象征性地代替距离，即当前节点与目标节点相同位置字符的个数，由于启发式函数的需要，舍弃了普通的队列，转而使用更加便利的双端队列deque。

启发函数得到的结果会与当前最好记录数“best”进行比较，若该结果大于或等于best，则将其放入双端队列的首部。best初始化为输入与目标的距离，使用这一方法的搜索在性能上具有略微提升：譬如搜索“123456780”到“012345678”，普通的宽度优先方法需要334453次搜索，而启发式方法只需要272544次。

1. 距离正确答案最远的节点优先

方法如其名，nonsense，毫无意义，我们本不指望这一方法会奏效，结果也不出所料：如果使3.距离正确答案最远的节点优先进队，搜索次数反而会增加，由334453次增加为386315次。

1. **随机搜索**
2. 基本实现

如果在getAnswer\_deque()的基础上采取随机方法，使每一个节点随机进队，搜索效果会因此得到优化吗？为此实现了方法getAnswer\_rand()，它使每一个节点随机的进入队列的头部或尾部。

1. 随机效果

经过实验，随机搜索的效果是惊人的：有时惊人的好——搜索“123456780”到“012345678”只需要261830次，有时又惊人的差，同样的搜索需要足足418688次尝试，搜索次数相差几乎一倍。

当然，这一结果仍在预料之内，毕竟随机是很不稳定的。