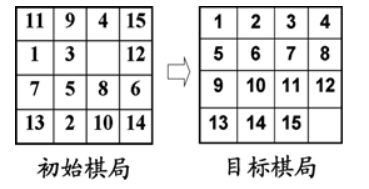
**1 问题描述**

* 1. **待解决问题的解释**

数码问题常被用来演示如何在状态空间中生成动作序列。一个典型的例子是15数码问题，它是由放在一个4×4的16宫格棋盘中的15个数码(1-15)构成，棋盘中的一个单元是空的，它的邻接单元中的数码可以移到该单元中，通过这样不断地移动数码来改变棋盘布局，使棋盘从给定的初始棋局变为目标棋局，比如下图

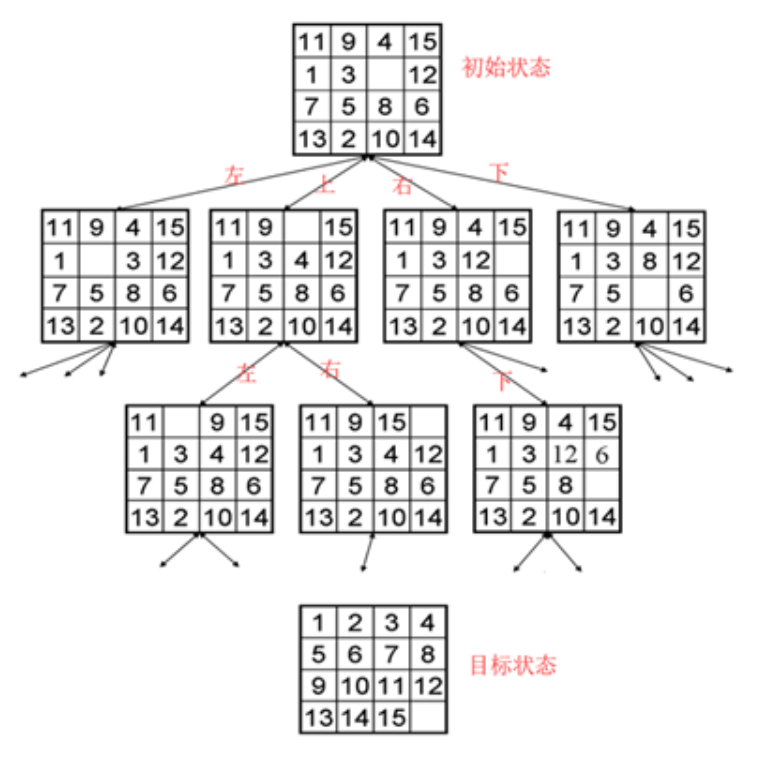


* 1. **问题的搜索形式描述**

(1)**状态S**：十五数码问题中，每种棋局就是一个状态，所有棋局就是状态集合S，其中共有16！=209227898888000个状态。如图输入的状态是**初始状态**，如果目标可达，结束时的状态是**目标状态**

(2)**操作符F**：使用最简化4个操作：分别向上、下、左、右移动空白单元，将操作符作用到某一状态即可从该状态转移到另一状态。但值得注意的是，并不是所有状态都可以执行这4个操作符。F = {上，下，左，右}

(3)**状态空间(S, F, G)**，其中状态空间图G的一部分如图



* 1. **解决方案介绍**

从A\*算法对于15数码问题的可解性讲，在状态空间中从初始状态到目标状态有很多条路径，每条路径与其他路径也有很多的交点从而形成网状结构，其中每条路的每一步都可以计算出相应的代价，因此可以使用A\*算法在网格中搜索出代价最小的路径

**2 算法介绍**

**2.1 搜索算法一般介绍**

搜索算法实际上是根据初始条件和扩展规则构造一棵“解答树”并寻找符合目标状态的节点的过程。所有的搜索算法从最终的算法实现上来看，都可以划分成两个部分——控制结构（扩展节点的方式）和产生系统（扩展节点），而所有的算法优化和改进主要都是通过修改其控制结构来完成的。其实，在这样的思考过程中，我们已经不知不觉地将一个具体的问题抽象成了一个图论的模型——树，即搜索算法的使用第一步在于搜索树的建立。

因此，这样形成的一棵树叫搜索树。初始状态对应着根节点，目标状态对应着目标结点。排在前的结点叫父结点，其后的结点叫子结点，同一层中的结点是兄弟结点，由父结点产生子结点叫扩展。完成搜索的过程就是找到一条从根结点到目标结点的路径，找出一个最优的解。这种搜索算法的实现类似于图或树的遍历，通常可以有两种不同的实现方法，即深度优先搜索（DFS）和广度优先搜索（BFS）。

A\*算法是BFS的一个变种，不同于BFS的是，每次选择节点进行生成的时候，优先选择估价函数最小的节点，把原来的BFS算法的无启发式的搜索改成了启发式的搜索，可以有效的减少节点的搜索个数。

**2.2 A\*算法伪代码**

F表示按照当前路径解决问题代价的估值函数

G表示从起点移动到网格当前节点的代价

H表示从当前节点移动到终点的预计代价

open:=[start];closed:=[]; F(n) = G(n) + H(n);

While open!=[] do

从open表中删除F(n)值最小的状态，称之为n;

If n=目前状态Then Return（success）；

生成n的所有子状态；

If n没有任何的子状态Then Continue；

For n的每个状态 Do

Case子状态is not already on open表or closed表：

计算该子状态的估价函数值（F(n),G(n)和H(n)）；将该子状态加到open表中；

Case子状态is already on open表：

If该子状态是沿着一条比在open表已有的更短路径而到达（G(n)值更小）

Then 记录更短路径及其估价函数值；

将n放入closed表中;

根据估价函数值，从小到大重新排列open表；

Return(failure) #open表中节点已耗尽

**3 算法实现**

**3.1 实验环境与问题规模**

**操作系统**：Windows10

**编译环境**：Python 3.9.7

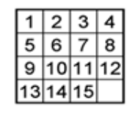
**编译器**：Pycharm

**时间复杂度**：因为不同起点到终点的搜索路径的长度差异很大，所以设搜索路径长度为n，因为H(n)选择的是与目标相比错位的数目，G(n)是初始结点到当前结点的路径长度，因此每个节点的F(n)是固定值，因此时间复杂度为O(n)

**空间复杂度**：每个轮次需要的空间都是固定的，因此空间复杂度为O(n)

**3.2 数据结构**

搜索树的每个节点表示一个15数码矩阵的状态，该状态使用一个16进制的数值字符串表示，分别为从左上按行读取到右下，比如'123456789abcdef0'表示下图的状态



使用一个字典，key是每个矩阵格子的编号，value是相邻矩阵格子的编号，代表key和value中的任意一个格子中的元素可以互相交换，比如其中的一个pair为 0: [1, 4]，代表第0个格子，即左上角第一个格子可以与其下方的4和它右边的1互换元素

使用汉明距离来衡量两个状态的距离，即两个状态对应字符串不同的位数来衡量从一个状态到另一个状态的代价

F(n),G(n)和H(n)都使用字典表示，key是node的字符串，value是对应的代价

使用parent字典来记录所走过的路径，key是当前节点的字符串，value是它父亲节点的字符串，这样就可以在算法结束时，通过不断迭代查询字典从路径的终点返回路径的起点并输出到屏幕上

**3.3 实验结果**

因为15数码这个问题本身就存在着状态不可达的问题，所以使用一个计数器，每在状态空间中改变一次状态，就给计数器加一，当计数器大于500时视为状态不可达，实验发现，有相当一部分初始状态都无法达到最终状态，如果从目标状态逆推有限步到初始状态，大多数都能在500步以内收敛至目标状态。

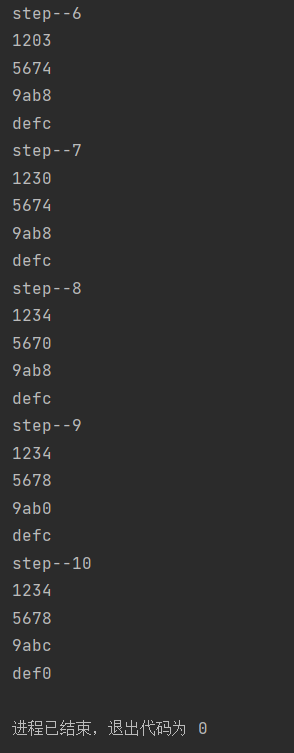
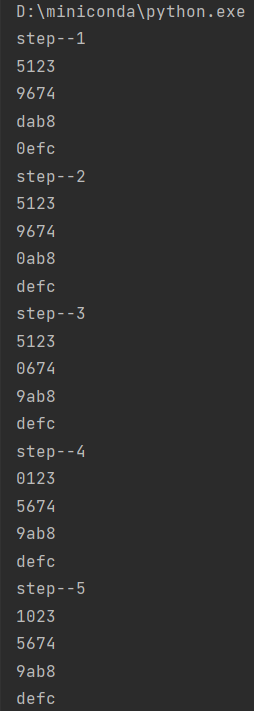
**3.4 系统中间及最终输出结果**

将初始状态，中间状态，目标状态都格式化为易读的矩阵形式输出，以

start = '51239674dab80efc'

goal = '123456789abcdef0'

为例，输出显示其每一步的结果



可见其不但满足了正确性的要求，每一步也都符合人类对于这个15数码问题的常规思考求解过程