# 使用A\*算法解决N码问题

## 启发式算法[1]

### 1.1 汉明距离(Hamming Distance)

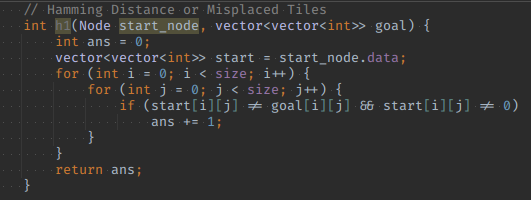
在本实验中，汉明距离表示错放的数字总数，在计算汉明距离时，我们不把数字‘0’考虑进去。

让我们考虑这样的一个输入：

|  |  |
| --- | --- |
| 3  1 2 5  3 0 6  7 4 8  1 2 3  4 5 6  7 8 0 | 起始状态：  1 2 5  3 0 6  7 4 8  目标状态：  1 2 3  4 5 6  7 8 0 |

对于起始状态，它的汉明距离为：3, 其中错放的数字为’5’, ‘3’, ‘4’。

代码实现如下：



程序运行分析：

|  |  |  |
| --- | --- | --- |
|  | 耗时 | 所需步数 |
| 汉明距离 | 1713.050059 s | 31641 |
| 曼哈顿距离 | 8.203404 s | 2141 |

总结：与其他启发式方法相比，汉明距离这种启发式方法是最慢的，需要探索大量的节点来达到目标状态。

### 1.2 曼哈顿距离（Manhattan Distance）

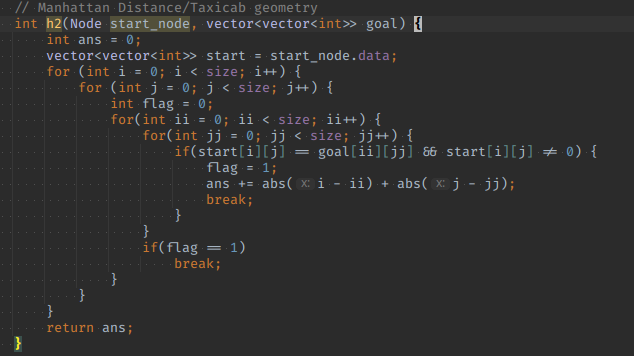
曼哈顿距离是同一数字在起始状态和目标状态距离。对于某种状态，曼哈顿距离将是除数字‘0’外的所有数字的曼哈顿距离的总和。

让我们考虑同样的输入：

|  |  |
| --- | --- |
| 3  1 2 5  3 0 6  7 4 8  1 2 3  4 5 6  7 8 0 | 起始状态：  1 2 5  3 0 6  7 4 8  目标状态：  1 2 3  4 5 6  7 8 0 |

对于起始状态，我们能够发现数字‘5’，‘3’，‘4’，‘8’被错放，计算它们的曼哈顿距离分别为2, 3, 2, 1。因此其实状态的曼哈顿距离为8。

代码实现如下：



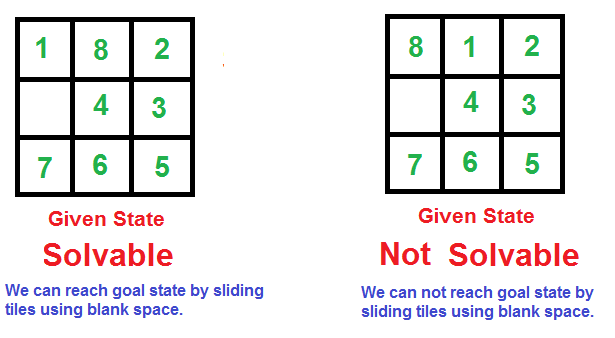
note: 代码通过四层循环实现曼哈顿距离，有很大的优化空间。

程序运行分析：

|  |  |  |
| --- | --- | --- |
|  | 耗时 | 所需步数 |
| 汉明距离 | 1713.050059 s | 31641 |
| 曼哈顿距离 | 8.203404 s | 2141 |

总结：曼哈顿距离比汉明距离速度显著提高，所需的步数显著减少。

## 可解性判断[2]



对于n码问题的可解性，前人已经做了很多研究。现在通用的方法是计算逆序对的数量。如果输入状态的逆序对数量是奇数，则该问题不可能解决。在上图中给出的例子中，第一个例子有10个逆序对，因此是可解的。第二个例子有11个逆序对，因此是不可解的。

上述方法能够解决标准的n码问题，即目标状态是顺序排列的。在本实验中，我们的目标状态不一定是循序排列的。因此，需要稍稍修改算法来解决我们的问题。

1. 将目标状态中的‘0’移动到右下角
2. 记录目标状态每个数字出现的位置，根据目标状态对起始状态重新编码
3. 计算重新编码后的起始状态的逆序对，判断可解性

|  |  |  |
| --- | --- | --- |
| 3  1 2 5  3 0 6  7 4 8  8 1 2  0 4 3  7 6 5 | 起始状态：  1 2 5  3 0 6  7 4 8  目标状态：  8 1 2  0 4 3  7 6 5 | 探索可解性：   1. 移动目标状态中0   8 1 2  7 4 3  6 5 0   1. 起始状态重新编码   1 2 7  5 8 6  3 4 0   1. 计算逆序对   11（不可解） |

## 参考文献：

[1] <https://algorithmsinsight.wordpress.com/graph-theory-2/a-star-in-general/implementing-a-star-to-solve-n-puzzle/>

[2] <https://www.geeksforgeeks.org/check-instance-8-puzzle-solvable/>