华东师范大学数据科学与工程学院实验报告

课程名称：人工智能	**年级**：2019	**上机实践成绩**：
指导教师：李翔	**姓名**：张雯怡	
上机实践名称：A*算法	**学号**：10195501425	**上机实践日期**：
上机实践编号：实验 2	**组号**：	**上机实践时间**：2022.4.5

1、 实验目的

编程完成两道算法题，理解并掌握 A*算法。

2、 实验任务

Q1：小明玩球

题目描述：小明在一个九宫格中随机摆了八个球，每个球上标有 1-8 中的某一数字（球上数字不重复）。九宫格中留有一个空格，该空格用0 表示。空格周围的球可以移动到空格中。现在，给出一种初始布局（即初始状态）和目标布局（本题的目标布局设为 123804765），现在小明想找到一种最少步骤的移动方法，实现从初始布局到目标布局的转变，你能帮帮他吗？

Q2：爱跑步的小明

题目描述：众所周知，小明身材很好。但自从他博士毕业当老师后，他就自我感觉身体变差了，于是他就想锻炼了。为了不使自己太累，他提出一种从山顶跑步到山脚的锻炼方法。

千寻万觅，终于在郊区找到这样一座山。这座山有 N 个地标，有先行者在这些地标之间开辟了 M 条道路。并且这些地标按照海拔从低到高进行了编号，例如山脚是 1，山顶是 N。

小明这个人对跑步的方式很挑：

（1）只跑最短路径。但一条最短路径跑久了会烦，需要帮他设计 K 条最短路径。

（2）不想太累，每次选道路的时候只从(海拔的)高处到低处。

现在问题来了，给你一份这座山地标间道路的列表，每条道路用(Xi, Yi, Di)表示，表示地标 Xi 和地标 Yi 之间有一条长度为 Di 的下坡道路。你来计算下小明这 K 条路径的对应长度，看看小明的锻炼强度大不大？

Q1 和 Q2 均要求只能用A*算法。

3、 使用环境

Q1:vs code python　　Q2:Dev C++ C++

4、 实验过程

A*算法的理解：A*算法区别于 dfs、bfs 等盲目搜索的区别在于 A*使用了启发式搜索，就是说给定了一个估价函数 f(x) = g(x) + h(x)。其中，x 为待评价的节点，假设初始状态或位置为 s，目标状态或位置为 t，g(x)为从 s 到 x 的最佳路径的 cost；h(x)为从 x 到目标 t 的最佳路径 cost 的估计值，称为启发函数或是估价函数；而 f(n)为从初始节点 s 经过节点 n 到达目标节点 t 的最佳路径 cost 的估计值。如果启发函数满足 h(x) <= h*(x)，则可以证明当问题有解时，A*算法一定可以得到一个 cost 最小的结果。

然后用 A*算法来解决上面的两个问题。

Q1：小明玩球

启发 h(x)选择：

对于这个问题能想到两种启发，第一个就是初始状态的九宫格和目标状态九宫格中位置不匹配的数字个数，第二个就是不在位的数字归位所需的最短距离和，由于这里数字块的移动只能上、下、左、右四个方向，距离的计算一定是采用曼哈顿距离。

显然第一种启发过于简单，效果肯定没有第二种好。因此启发 h 选用参考曼哈顿距离。

但是这里 h(x)不是直接使用曼哈顿距离 |x1-x2| + |y1-y2| ，而是 |x1-x2+y1-y2| ，这个地方后面再作详细说明。

程序设计过程：

首先把九宫格及其状态抽象，每个状态节点有它的 g、h 和前继/父节点。

其中 g 值使用从初始到当前状态节点的 steps，启发 h 使用过一个类似曼哈顿距离的函数。因为 f = g + h ，就不在类内添加变量了直接写成函数。

```
"""
估价函数——类似曼哈顿距离
"""
def setH(self, endNode):
    for x in range(0, 3):
        for y in range(0, 3):
            for m in range(0, 3):
                for n in range(0, 3):
                    if self.matrix[x][y] == endNode.matrix[m][n]:
                        #self.h += abs(x * y - m * n)
                        if (self.matrix[x][y] != 0):
                            self.h += abs(x - m + y - n)

def setG(self, g):
    self.g = g

def setFather(self, node):
    self.father = node

def getG(self):
    return self.g

def getF(self):
    return self.h + self.g
```

```
class Node:
    def __init__(self, matrix, g=0, h=0):
        self.matrix = matrix  # 把九宫格抽象为矩阵
        self.father = None  # 父节点
        self.g = g  # g(x)
        self.h = h  # h(x)
    """
```

然后设置一个变量 openList 用于存放那些搜索图上的叶节点，就是已生成但为扩展的节点；变量 closeList 用于存放图中的非叶节点，也就是已经被生成和被扩展的节点；

```
class A:
    def __init__(self, startNode, endNode):
        self.openList = []
        self.closeList = []
        self.startNode = startNode
        self.endNode = endNode
        self.currentNode = startNode
        self.pathlist = []
        self.step = 0
        return
```

```
def getMinFNode(self):
    """
    获得openlist中F值最小的节点
    """
    nodeTemp = self.openList[0]
    for node in self.openList:
        if node.getF() < nodeTemp.getF():
            nodeTemp = node
    return nodeTemp
```

取 openList 中 f 最小的节点 node 进行扩展，如果 node 是目标节点，则找到一个解，算法结束，否则扩展 node；

对于 node 的子节点 m，如果 m 既不在 openList 也不在 closeList，则将 m 加入 openList；如果 m 在 closeList，说明从初始节点到 m 有两条路径，如果新路径的 cost 更小，则将 m（新找到的）从 closeList 中删除并取出放入 openList 中；

```python
def searchOneNode(self, node):
    """
    搜索一个节点
    """
    # closeList不用考虑
    if self.nodeInCloselist(node):
        return
    # G(x)计算
    gTemp = self.step

    # 如果不在openList中，就加入openlist
    if self.nodeInOpenlist(node) == False:
        node.setG(gTemp)
        # 计算启发h(x)
        node.setH(self.endNode)
        # 加入openList
        self.openList.append(node)
        # 当前节点置为父节点
        node.father = self.currentNode

    # 如果在openList中，判断currentNode到当前点的G是否更小
    # 如果更小，就重新计算g(x)，并且改变父节点
    else:
        nodeTmp = self.getNodeFromOpenList(node)
        if self.currentNode.g + gTemp < nodeTmp.g:
            nodeTmp.g = self.currentNode.g + gTemp
            nodeTmp.father = self.currentNode
    return
```

```python
def start(self):
    '''''
    开始寻路
    '''
    # 将初始节点加入openList
    self.startNode.setH(self.endNode)
    self.startNode.setG(self.step)
    self.openList.append(self.startNode)

    while True:
        # 获取openList里F值最小的节点
        # 并把它添加到closeList，从openList删除
        self.currentNode = self.getMinFNode()
        self.closeList.append(self.currentNode)
        self.openList.remove(self.currentNode)
        self.step = self.currentNode.getG()
        self.searchNear()
        # 检验是否结束
        if self.endNodeInOpenList():
            nodeTmp = self.getNodeFromOpenList(self.endNode)
            while True:
                self.pathlist.append(nodeTmp)
                if nodeTmp.father != None:
                    nodeTmp = nodeTmp.father
                else:
                    return True
        elif len(self.openList) == 0:
            return False
        elif self.step > 100:
            return False
    return True
```

重复前面所述步骤，直至找到一个解结束；或 openList 为空，即无解。

最终运行结果：

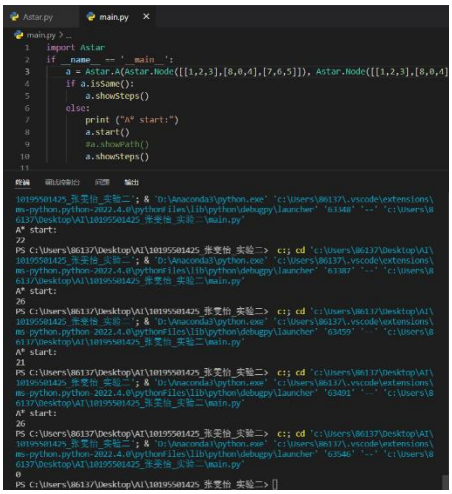
输入 1：024657318　　输出 1：22

输入 2：587346120　　输出 2：26（跑了比较久，大概 1min）

输入 3：375148206　　输出 3：21

输入 4：512768340　　输出 4：26（大概 30s）

输入 5：123804765　　输出 5：0



Q2：爱跑步的小明

这道题就是使用 A*算法解决 k 条最短路径问题。

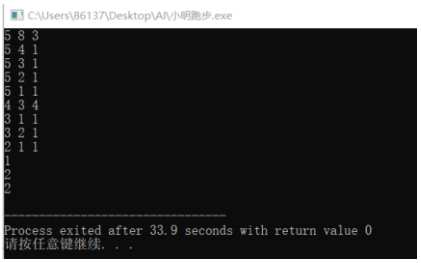
启发 h(x)选择：从 n 到当前点走过的距离 + 从当前点到终点的最短路径。

程序设计过程：

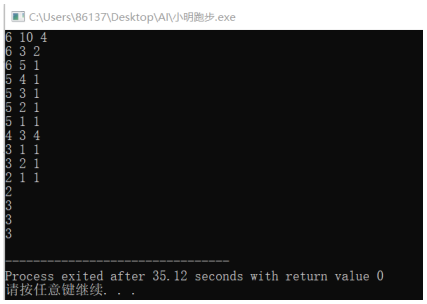
先建反向边求出每个点到山脚 1 的最短路径，对于每条路径的估价函数定义为这个点到 n 走过的距离加上这个点到终点的最短路，每次从堆中取出最小值，直至走到山脚 1 时给出路径答案。

程序运行结果：

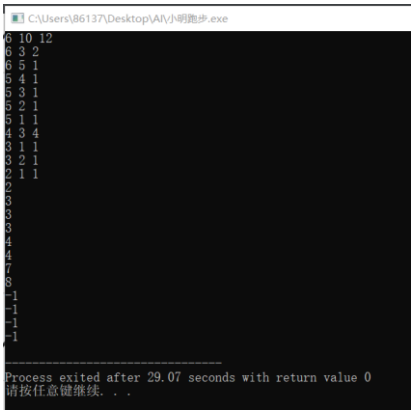
输出 1：1 2 2



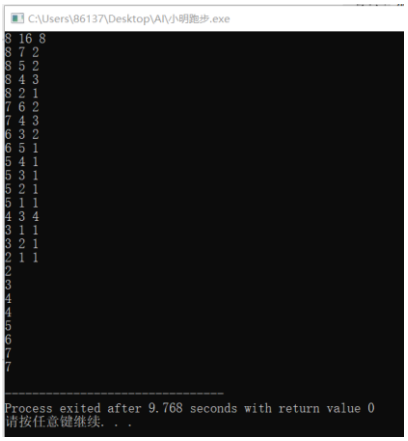
输出 2：2 3 3 3

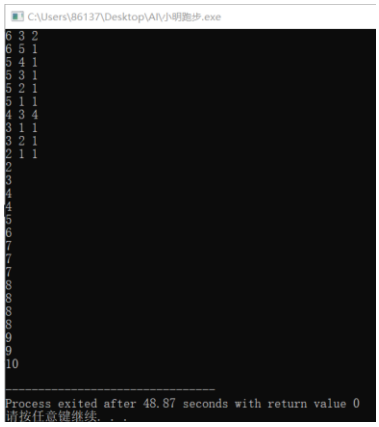


输出 3：2 3 3 3 4 4 7 8 -1 -1 -1 -1



输出 4：2 3 4 45 6 7 7





输出 5：2 3 4 4 5 6 7 7 7 8 8 8 8 9 9 10

5、 总结

A*算法以启发式搜索区别于与之有点类似的盲搜中的 dfs，其中启发函数 h(x)的选择对整个算法的效率有决定性的作用。

解决小明玩球的问题时我在启发函数的选择上踩了坑，一开始直接想当然地使用了曼哈顿距离作为启发，然而在测试用例中第二个输入跑出来的结果总是 28，而别的同学几乎都是 26，确定了后面代码逻辑没有问题之后嫌疑就到了启发函数 h(x)上。后面在和同学讨论中我们发现了这样一个问题：输出用例 2 的路径，发现在这一步

我的算法做出了不同的决策。是因为将矩阵拉伸成向量后，对于一个数字要移动到另一个位置上，使用曼哈顿距离其评分是一样的，但实际执行策略有顺时针和逆时针移动，两种决策之间正好相差 2steps。一种正确的类似曼哈顿距离的启发函数是|x1-x2+y1-y2|，可以避免两个策略估值相同。