**电子科技大学**

**计算机科学与工程学院**

**标 准 实 验 报 告**

**（实验）课程名称 人工智能**

**电子科技大学教务处制表**

**电 子 科 技 大 学**

**实 验 报 告**

**学生姓名：王宇豪 学 号：2023110801008 指导教师：顾舒航**

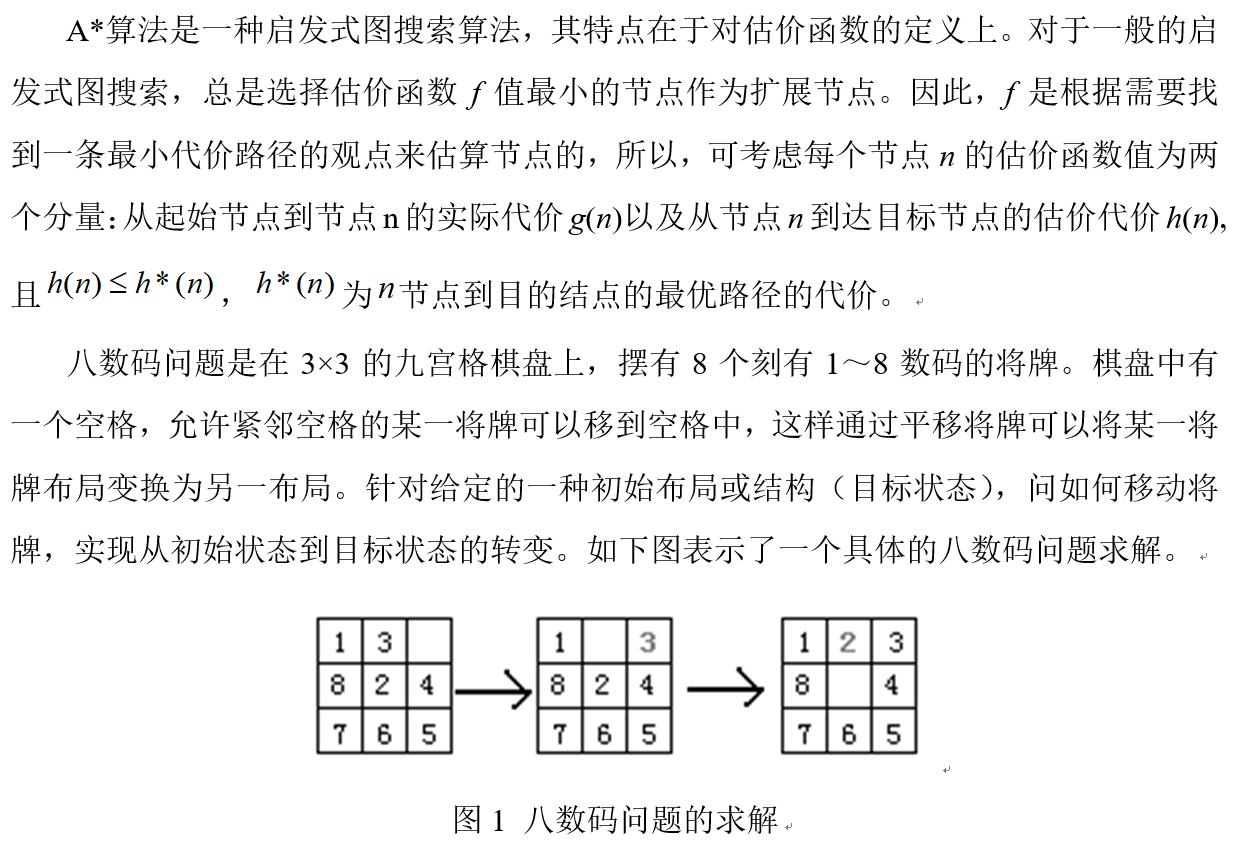
**实验地点：主楼A2-413-1 实验时间：周日下午**

**一、实验室名称：计算机学院实验中心**

**二、实验项目名称：A\*算法实验**

**三、实验学时：5学时**

**四、实验原理：**



**五、实验目的：**

熟悉和掌握启发式搜索的定义、估价函数和算法过程，并利用A\*算法求解N数码难题，理解求解流程和搜索顺序。

**六、实验内容：**

1. 以8数码问题为例实现A\*算法的求解程序（编程语言不限），设计估价函数。

注：需在实验报告中说明估价函数，并附对应的代码。

2. 设置初始状态和目标状态，针对估价函数，求得问题的解，并输出移动过程。

要求：

（1）提交源代码及可执行文件。

（2）提交实验报告，内容包括：对代码的简单说明、运行结果截图及说明等。

**七、实验器材（设备、元器件）：**

PC微机一台

**八、实验步骤：**

1. 确定估价函数

由于八数码问题牵动的是整个整体，所以我们再确定估价函数时应该将每一个位置全部都考虑进来，比如说不在各自位置的位数、每一个个体到目标位置的曼哈顿距离等等。在这里我们选择曼哈顿距离，现在我们先证明它会小于实际应该走的步数，由于每次只会移动一块，并且只能上下左右移动，因此显然成立。

代码如下(python，IDE为Pycharm)：

def heuristic(self, s):  
 h = 0  
 goal\_str = self.list2str(self.goal)  
 for idx in range(9):  
 if s[idx] == '0':  
 continue #空格不计入  
 x1, y1 = idx // 3, idx % 3  
 x2, y2 = goal\_str.find(s[idx]) // 3, goal\_str.find(s[idx]) % 3  
 h += abs(x1 - x2) + abs(y1 - y2)  
 return h

1. 确定

我们很容易想到把它设定为当前已经走过的步数。

1. 如何存储一个状态

我们应该将这个状态存储为一个列表或者一个字符串（python）。然后还要额外存储为一个字典，值为这个状态相对应的总代价，这里注意，我们对于相同的代价只存储它的最小代价，这是显然的。

1. 具体方法

确定完核心步骤之后我们就开始正式设定我们的算法，由于空位只有一个，那么我们就根据这个空位的位置将它与上下左右的块进行交换（如果可以），然后将这个状态和值压入我们的堆（也就是优先队列），直到我们得到我们的目标状态。但是我们每一次拓展操作只在我们目前总代价最小的哪个状态进行，并且进行完之后就派出去，或者将代价置为一个非常大的数。

1. 关于无解

八数码问题可能是无解的，当所有可能的移动路径都被探索完毕（堆为空），且未找到目标状态时，说明问题无解。八数码问题的有解性可通过逆序数奇偶性判断：

* 初始状态与目标状态的逆序数奇偶性相同 → 有解。
* 奇偶性不同 → 无解。

代码中未显式判断逆序数，但通过A\*算法自然覆盖：若问题无解，堆终将耗尽。

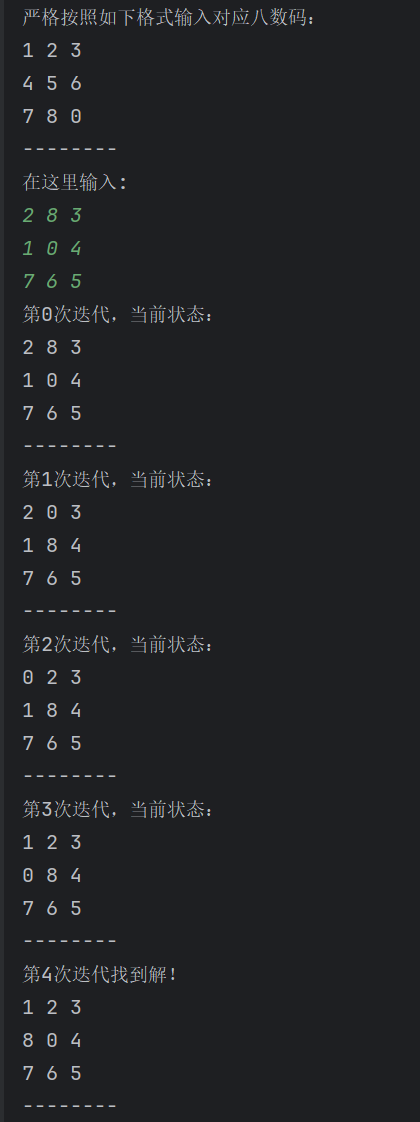
**九、实验数据及结果分析：**

在我的代码中提供了输入选择，但是格式要一定，这里我们选择两个典型的例子，一个简单有解，一个无解。

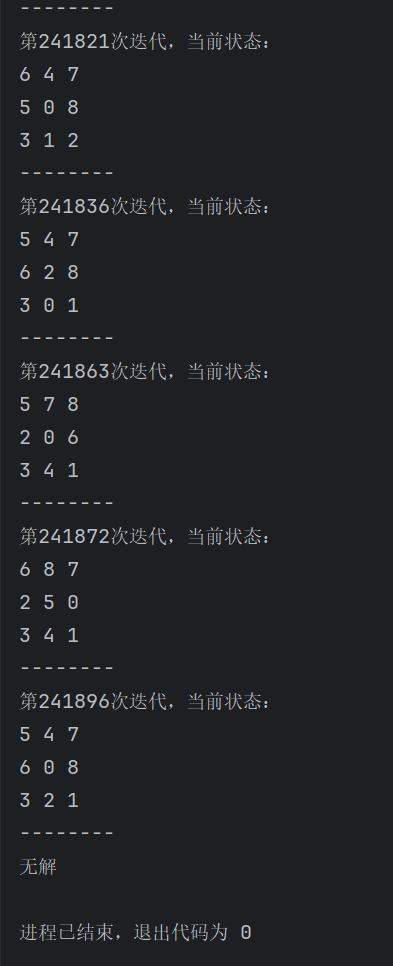
分别是 和

我们看一下结果

1. 结果一（有解）：



1. 结果二（无解）：



就像前面实验步骤所讨论的一样，根据逆序数奇偶性我们可以判断我们第一个输入是有解的，第二个输入是无解的。

**十、实验结论：**

我们所撰写的A\*算法是正确的，也就是说八数码问题可以通过A\*算法来解决。

**十一、总结及心得体会：**

本次实验逻辑上是比较简单的，但是实现上还得一步一步去模块化地实现，总之还是挺满意的。

**十二、对本实验过程及方法、手段的改进建议：**

建议在八数码问题前面加一个简单的距离搜索问题，循序渐进，这样可以让学生少一些突兀感。

**十三、附录（代码）：**

import copy  
  
class A\_star:  
 #初始化目标状态  
 def \_\_init\_\_(self, start, goal):  
 self.start = start  
 self.goal = goal  
 self.heap = [['top', 0, 0]] #堆从索引1开始，索引0占位  
 self.closed = set() #已扩展节点集合  
 self.cnt = 0 #迭代计数器  
  
 #堆下降操作  
 def heap\_down(self, index):  
 t = index  
 if index \* 2 <= len(self.heap)-1 and self.heap[index\*2][2] < self.heap[t][2]:  
 t = index \* 2  
 if index \* 2 + 1 <= len(self.heap)-1 and self.heap[index\*2+1][2] < self.heap[t][2]:  
 t = index \* 2 + 1  
 if t != index:  
 self.heap[t], self.heap[index] = self.heap[index], self.heap[t]  
 self.heap\_down(t)  
  
 #堆上升  
 def heap\_up(self, index):  
 while index > 1 and self.heap[index][2] < self.heap[index//2][2]:  
 self.heap[index], self.heap[index//2] = self.heap[index//2], self.heap[index]  
 index = index // 2  
  
 #堆插入  
 def heap\_insert(self, state\_str, g, f):  
 self.heap.append([state\_str, g, f])  
 self.heap\_up(len(self.heap)-1)  
  
 #堆pop  
 def heap\_pop(self):  
 if len(self.heap) <= 1:  
 return None  
 res = self.heap[1]  
 if len(self.heap) > 2:  
 self.heap[1] = self.heap[-1]  
 self.heap.pop()  
 self.heap\_down(1)  
 else:  
 self.heap.pop()  
 return res  
  
 #转换列表为字符串  
 def list2str(self, list1):  
 return ''.join(str(num) for row in list1 for num in row)  
  
 #转换字符串为列表  
 def str2list(self, s):  
 return [[int(s[3\*i + j]) for j in range(3)] for i in range(3)]  
  
 #计算曼哈顿距离  
 def heuristic(self, s):  
 h = 0  
 goal\_str = self.list2str(self.goal)  
 for idx in range(9):  
 if s[idx] == '0':  
 continue #空格不计入  
 x1, y1 = idx // 3, idx % 3  
 x2, y2 = goal\_str.find(s[idx]) // 3, goal\_str.find(s[idx]) % 3  
 h += abs(x1 - x2) + abs(y1 - y2)  
 return h  
  
 #拓展当前节点  
 def a\_expand(self, node):  
 current\_str, g = node[0], node[1]  
 current\_list = self.str2list(current\_str)  
 empty\_pos = current\_str.find('0')  
 x, y = empty\_pos // 3, empty\_pos % 3  
  
 #四个移动方向  
 directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]  
 for dx, dy in directions:  
 nx, ny = x + dx, y + dy  
 if 0 <= nx < 3 and 0 <= ny < 3:  
 new\_list = copy.deepcopy(current\_list)  
 new\_list[x][y], new\_list[nx][ny] = new\_list[nx][ny], new\_list[x][y]  
 new\_str = self.list2str(new\_list)  
  
 #检查是否为目标  
 if new\_str == self.list2str(self.goal):  
 print(f"第{self.cnt}次迭代找到解！")  
 self.print\_state(new\_str)  
 return True  
  
 #未处理过的加入堆  
 if new\_str not in self.closed:  
 h = self.heuristic(new\_str)  
 self.heap\_insert(new\_str, g + 1, g + 1 + h)  
 return False  
  
 #打印当前状态  
 def print\_state(self, s):  
 for i in range(3):  
 print(' '.join(s[3\*i:3\*i+3]))  
 print("--------")  
  
 #主算法入口  
 def a\_star(self):  
 start\_str = self.list2str(self.start)  
 h\_start = self.heuristic(start\_str)  
 self.heap\_insert(start\_str, 0, h\_start)  
  
 while len(self.heap) > 1:  
 self.cnt += 1  
 current\_node = self.heap\_pop()  
 if current\_node is None:  
 break  
 current\_str = current\_node[0]  
  
 #跳过已处理状态  
 if current\_str in self.closed:  
 continue  
 self.closed.add(current\_str)  
  
 print(f"第{self.cnt - 1}次迭代，当前状态：")  
 self.print\_state(current\_str)  
  
 if self.a\_expand(current\_node):  
 return  
  
 print("无解")  
  
#测试案例  
#注意我这个示例是无解的  
print('严格按照如下格式输入对应八数码：')  
print('''1 2 3  
4 5 6  
7 8 0  
--------''')  
print('在这里输入:')  
start = []  
for i in range(3):  
 st = []  
 a, b, c = map(int, input().split())  
 st.append(a), st.append(b), st.append(c)  
 start.append(st)  
goal = [[1, 2, 3], [8, 0, 4], [7, 6, 5]]  
solver = A\_star(start, goal)  
solver.a\_star()

**报告评分：**

**指导教师签字：**