实　验：八　数　码　问　题

一. 实验目的

1. 以八数码问题作为对象，利用A*算法求解并在屏幕上动态显示OPEN表的结点和评估函数最小的结点

二. 实验内容

1. 使用两种启发式函数并比较两者的不同之处。

三. 实验器材

语言：Python、编译器：PyCharm

四. 实验过程与结果

- 这是一个八数码问题，也就是一个3×3的九宫格打乱之后的恢复问题，其中空格我们可以用数字0补全，这样方便与自己编程。首先的第一个问题也就是能否到达目标状态的，可以通过资料查阅得到的条件是：**初始状态与目标状态的逆序和的奇偶性一样也就可以恢复初始状态**（对于所有的奇数的数码问题都是如此）

所以这一部分的代码如下：

```
# 计算逆序和
def calculat_inverse(arr):
    inverse_sum = 0
    n = len(arr)
    m = n ** 2 - 1
    for i in range(n ** 2):
        temp = [k + 1 for k in range(i, m - 1)]
        for j in temp:
            if arr[int(j / n), j % n] == 0:
                continue
            elif arr[int(i / n), i % n] > arr[int(j / n), j % n]:
                inverse_sum = inverse_sum + 1
    return inverse_sum
# 判断是否能够到达目的状态
# 用于判断是否有解
def judge(start, target_state):
    return (calculat_inverse(start) % 2) == (calculat_inverse(target_state) % 2)
```

- 两种启发函数（因为一个是八数码一个是九数码问题，其中九数码问题在于将0也算入评估函数之中，所以就会有4个，尽管八数码与九数码只是细微上的区别）

```
# 九数码
# 评估函数2
# 放错元素与正确位置的曼哈顿距离
# 返回一个数字
def evaluate2(current, target_state):
    distances = 0
```

```
    for i in range(len(current)):
        for j in range(len(current)):
            for k in range(len(current)):
                for m in range(len(current)):
                    if current[i][j] == target_state[k][m]:
                        distances = distances + (abs(i - k) + abs(j - m))
    return distances + evaluate(current, target_state)
# 九数码
# 评估函数1
# 放错位置的元素的个数
# 返回一个数字
def evaluate(current, target_state):
    temp = current - target_state
    return sum(temp != 0)
# 八数码
# 评估函数3 八数码问题
# 放错位置的元素的个数
# 返回一个数字
def evaluate3(current, target_state):
    temp = current - target_state
    result = sum(temp != 0)
    if result > len(current)**2 - 1:
        result = len(current)**2 - 1
    return result
 # 八数码
# 评估函数4
# 放错元素与正确位置的曼哈顿距离
# 返回一个数字
def evaluate4(current, target_state):
    distances = 0
    for i in range(len(current)):
        for j in range(len(current)):
            for k in range(len(current)):
                for m in range(len(current)):
                    if current[i][j] == target_state[k][m] and current[i][j] != 0 and
target_state[k][m] != 0:
                        distances = distances + (abs(i - k) + abs(j - m))
    return distances + evaluate(current, target_state)
```

- **核心代码，也就是A*算法：**

```
# 算法核心
def a_start(states, current, target_state, tag):
    # 根据标签选择对应的评估函数
    if tag == 1:
        method = evaluate
    elif tag == 2:
        method = evaluate2
    elif tag == 3:
        method = evaluate3
    elif tag == 4:
        method = evaluate4
    path = zeros([maxsize], dtype=uint)
    # 记录是否访问过
    visited = array([False for i in range(maxsize)])
    # 估计函数值
    fitness = array([0 for i in range(maxsize)])
    # 路径的长度
    passlen = array([0 for i in range(maxsize)])
```

```
    # 记录状态的变换
    curpos = copy.deepcopy(current)
    id, curid = 0, 0
    fitness[id] = method(curpos, target_state)
    states[id][:][:] = current
    id = id + 1
    # 访问的节点的个数
    count = 0
    # 判断状态是否相同
    while not str(curpos) == str(target_state):
        for i in range(1, 5):
            tmp = move(curpos, i)
            # 判断是否移动成功
            if tmp[1]:
                state = checkadd(states, tmp[0], id)
                # 不在表中，存入表中
                if state == -1:
                    count = count + 1
                    path[id] = curid
                    passlen[id] = passlen[curid] + 1
                    fitness[id] = method(tmp[0], target_state) + passlen[id]
                    states[id][:][:] = tmp[0]
                    id = id + 1
                # 在表中，进行更新
                else:
                    l = passlen[curid] + 1
                    fit = method(tmp[0], target_state) + l
                    if fit < fitness[state]:
                        path[state] = curid
                        passlen[state] = l
                        fitness[state] = fit
        visited[curid] = True
        minCur = -1
        # 取出最小的节点
        open_count = 0
        for i in range(id):
            if (visited[i] == False and (minCur == -1 or fitness[i] < fitness[minCur])):
                open_count += 1
                minCur = i
        curid = minCur
        curpos = states[curid][:][:]
        print("当前 open 表的数目是：", open_count)
        print("当前总的结点数目是：", count)
        print("当前最小的节点评估值是：", fitness[curid], "当前最小的节点是")
        print(states[curid])
        print("+"*100)
        if id == maxsize:
            return -1
    # curid 表示最后的状态的 id 值，count 表示总的结点的数目，path 表示变换的路径，fitness 表示每一个结点的评估值
    return curid, count, path, fitness
```

- 输出界面动态显示OPEN表的结点数、总扩展的结点数和评估函数最小的结点

示例如下：

```
--------------------------------------------------
原状态
[[0 1 2]
 [4 5 3]
 [7 8 6]]
方法： 1
当前open表的数目是： 2
当前总的结点数目是： 2
当前最小的节点评估值是： 5 当前最小的节点是
[[1 0 2]
 [4 5 3]
 [7 8 6]]
++++++++++++++++++++++++++++++++++++++++++++++++++
当前open表的数目是： 2
当前总的结点数目是： 4
当前最小的节点评估值是： 5 当前最小的节点是
[[1 2 0]
 [4 5 3]
 [7 8 6]]
++++++++++++++++++++++++++++++++++++++++++++++++++
当前open表的数目是： 2
当前总的结点数目是： 5
当前最小的节点评估值是： 5 当前最小的节点是
[[1 2 3]
 [4 5 0]
 [7 8 6]]
++++++++++++++++++++++++++++++++++++++++++++++++++
当前open表的数目是： 2
当前总的结点数目是： 7
当前最小的节点评估值是： 4 当前最小的节点是
[[1 2 3]
 [4 5 6]
 [7 8 0]]
++++++++++++++++++++++++++++++++++++++++++++++++++
```

- 输出open表中在最佳路径上的结点与评估函数值并验证A*算法所挑选出来的后继节点都必定满足： f(n) ≤f*(S0)

```
--------------------------------------------------
5
第1次移动的结果
[[1 0 2]
 [4 5 3]
 [7 8 6]]
--------------------------------------------------
5
第2次移动的结果
[[1 2 0]
 [4 5 3]
 [7 8 6]]
--------------------------------------------------
5
第3次移动的结果
[[1 2 3]
 [4 5 0]
 [7 8 6]]
--------------------------------------------------
4
第4次移动的结果
[[1 2 3]
 [4 5 6]
 [7 8 0]]
--------------------------------------------------
原状态
```

以上的结果可以说明上面的条件了f(n) ≤f*(S0)

- 验证h1(n)的单调性，显示凡A*算法挑选出来求后继的点ni扩展的一个子结点nj，检查是否满足: h(ni) ≤1+h(nj)

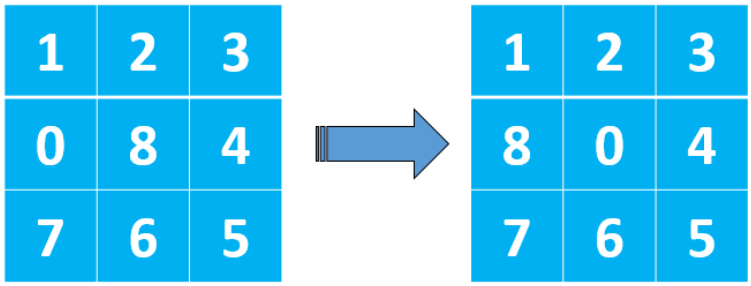
```
方法:  1
当前层数 0
前一代的评估值 4
当前的评估值 5
当前层数 0
前一代的评估值 4
当前的评估值 5
当前层数 0
前一代的评估值 4
当前的评估值 3
当前open表的数目是:  2
当前总的结点数目是:  3
当前最小的节点评估值是:  4 当前最小的节点是
[[1 2 3]
 [4 0 5]
 [7 8 6]]
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
当前层数 1
前一代的评估值 3
当前的评估值 4
当前层数 1
前一代的评估值 3
当前的评估值 4
当前层数 1
前一代的评估值 3
当前的评估值 4
当前层数 1
前一代的评估值 3
当前的评估值 2
当前open表的数目是:  2
当前总的结点数目是:  6
当前最小的节点评估值是:  4 当前最小的节点是
[[1 2 3]
 [4 5 0]
 [7 8 6]]
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
当前层数 2
前一代的评估值 2
当前的评估值 3
当前层数 2
前一代的评估值 2
当前的评估值 0
当前层数 2
前一代的评估值 2
当前的评估值 3
当前open表的数目是:  2
当前总的结点数目是:  8
当前最小的节点评估值是:  3 当前最小的节点是
[[1 2 3]
 [4 5 6]
 [7 8 0]]
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

- 如果将空格看作0，即九数码问题，利用相似的启发函数h1(n)和h2(n)，求解相同的问题的搜索图是否相同?

求解相同的问题的搜索是不相同的，因为在进行评估的时候会评估的结果是不一样的，因为前面默认为空的时候不会将其作为数字然后计入放错位置元素的个数以及没有放错的曼哈顿距离。而作为数字0的时候就会增加这一部分的数值。而这就导致了搜索的空间是不一样的。同时九数码问题还不符合A*条件。

如下：

用相似的估计函数时，九数码的启发式算法在这里不满足A*算法的必要条件$h(n) \leqslant h^*(n)$，即没有A*算法的性质，所以不一定能得到最优解



八数码：$1 = h1(n) \leqslant h^*(n) = 1$
九数码：$2 = h1(n) > h^*(n) = 1$

五. 实验心得

在上课的时候没有很认真的听讲，只知道上学期的实训的时候做过，所以就只记住了两个评估函数，也就是放错的位置元素的个数和放错位置的元素与目标位置的曼哈顿距离。做着一个实验的时候才知道还有open集与closed集，其中close集存放的是每一次评估的最小的状态，而每一次产生新的状态都需要从open集中取出上面的最小的状态。产生的新的状态有两种处理方法：

1. 当存在与前面产生过的状态集的时候就更改该节点的指针与费用。（如果更小的话）
2. 如果不存在，则存入状态集中并加入open集中，录入费用。

另外每一个状态的费用不只是评估函数的值，还与当前的层数有关，所以要加上层数。