Astar八数码问题

问题描述：3\*3的棋盘，棋盘上摆着八个数字的棋子，留下一个空位，与空位相邻的棋子可挪到空位，最终达到一个状态

实验代码：

**import** copy  
**import** os  
**import** time  
  
  
*# 将一个八数码序列用3x3的阵列打印出来***def** prtNum(src):  
 **for** x **in** range(3):  
 **for** y **in** range(3):  
 print(str(src[x \* 3 + y] + **' '**), end=**''**)  
 print()  
  
  
*# 判断两个数码序列之间的差别，完全相同返回0，有一个字符不相同则返回值+1***def** diff(src, dst):  
 total = 0  
 **for** x **in** range(len(src)):  
 **if** src[x] != dst[x]:  
 total = total + 1  
 **return** total  
  
  
*# 返回一个序列中空‘’在序列中的位置，以及在3x3阵列中的行、列位置。***def** position(src):  
 flag = src.index(**' '**)  
 row = int(flag / 3)  
 col = int(flag % 3)  
 **return** [flag, row, col]  
  
  
*# 将一个3x3阵列中两个位置的数值调换，并返回调换后的序列***def** exchange(src, x, y, x2, y2):  
 flag = x \* 3 + y  
 flag2 = x2 \* 3 + y2  
 tmp1 = src[flag]  
 tmp2 = src[flag2]  
 dst = copy.copy(src)  
 dst[flag] = tmp2  
 dst[flag2] = tmp1  
 **return** dst  
  
  
*# 用于判定一个序列状态能否变换成另一个序列状态的依据，  
# 根据理论，转换前后的序列应具有这个特征：假设flag(n)等于  
# 该序列中数字n前面所有大于它的数字的和，则可以转换的两个序列  
# 其flag(1)+...+flag(8)之后的奇偶性应该相同。例如，原始状态  
# 1234 5678的flag()之和为:0, 其可以转换为12345 678，flag()之和  
# 也为0，因此互相可以转化。  
# 本函数初始化序列后，返回某个序列的flag（）之和。  
#***def** judge(number):  
 total = 0  
 data = [9, 9, 9, 9, 9, 9, 9, 9, 9]  
 **for** i **in** range(9):  
 **if** number[i] != **' '**:  
 data[i] = int(number[i])  
 **else**:  
 data[i] = 0  
 *# print('number is',number)  
 # print('data is',data)* **for** i **in** range(9):  
 **for** j **in** range(i):  
 **if** data[i] \* data[j] != 0:  
 **if** data[j] > data[i]:  
 total = total + 1  
 *# print(i,total)* **return** total  
  
  
*# 用于处理Open表  
# 方法是：如果Open表非空，则：  
# 1）按照顺序对open表的每个node中的空格进行所有方向的移动，  
# 将移动后的新状态节点添加进open表；如果过程中找到了满足条件  
# 的目的状态节点，则停止处理并返回打印结果；  
# 如果新获得的序列已存在与open、close表，则不再添加。  
# 2）将该节点加入close表；  
# 3）从open表中删除该节点；  
#***def** handleOpen():  
 **global** nodeid  
 **global** open  
 **while True**:  
 **if** len(open) == 0:  
 **break** *# x=0* **for** x **in** range(len(open)):  
 tmp = move(open[x][0], **''**)  
 *# print(tmp)  
 # print(open)  
 # print('tmp length is',len(tmp))* **for** y **in** range(len(tmp)):  
 flag = **False  
 for** jj **in** range(len(open)):  
 *# print('tmp[y][0]is',tmp[y][0])  
 # print('open[x][0]is',open[x][0])* **if** tmp[y][0] == open[jj][0]:  
 flag = **True** *# print('falg open set to True')* **for** kk **in** range(len(closed)):  
 *# print('tmp[',y,'][0]is',tmp[y][0])  
 # print('closed[',kk,'][0]is',closed[kk][0])* **if** tmp[y][0] == closed[kk][0]:  
 flag = **True** *# print('falg close set to True')* **if** flag == **False**:  
 open.append([tmp[y][0], tmp[y][1], tmp[y][2], tmp[y][3], open[x][3]])  
 *# print('add open node',open[-1])  
 # else:  
 # print('node',tmp[y][0], 'already exists in open or closed!')* **if** tmp[y][2] == 0:  
 closed.append(open[x])  
 closed.append(open[-1])  
 open.remove(open[x])  
 *# print('add close node',open[x])* print(**'Totally'**, nodeid, **'nodes ayalyzed,find the result.'**)  
 prtResult()  
 print(**'Success!'**)  
 exit(**"We find it!"**)  
 closed.append(open[x])  
 *# print('add close node',open[x])* open.remove(open[x])  
  
  
*# 基于输入的序列进行移动，并返回所有可能的移动后目的序列；  
# 每条数据：节点序列、前一节点序列、与目标序列偏差值、当前节点序列ID***def** move(src, side):  
 **global** crt  
 **global** nodeid  
 pos = position(src)  
 flag = pos[0]  
 x = pos[1]  
 y = pos[2]  
 leftDiff = 999  
 rightDiff = 999  
 upDiff = 999  
 downDiff = 999  
 *# print('Node being analyzed is:')  
 # prtNum(src)* rtResult = []  
 **if** side == **'left' or** side == **''**:  
 **if** y > 0:  
 crtLeft = exchange(src, x, y, x, y - 1)  
 *# print('Can move to LEFT,after move result is:')  
 # prtNum(crtLeft)* leftDiff = diff(numberFinal, crtLeft)  
 *# print('different factor is',leftDiff)  
 # addOpen(crtLeft,src,leftDiff)  
 # return [crtLeft,src,leftDiff]* nodeid = nodeid + 1  
 rtResult.append([crtLeft, src, leftDiff, nodeid])  
 *# else:  
 # print('Cannot move to LEFT!')* **if** side == **'right' or** side == **''**:  
 **if** y < 2:  
 crtRight = exchange(src, x, y, x, y + 1)  
 *# print('Can move to Right,after move result is:')  
 # prtNum(crtRight)* rightDiff = diff(numberFinal, crtRight)  
 *# print('different factor is',rightDiff)  
 # return(crtRight,src,rightDiff)* nodeid = nodeid + 1  
 rtResult.append([crtRight, src, rightDiff, nodeid])  
 *# else:  
 # print('Cannot move to RIGHT!')* **if** side == **'up' or** side == **''**:  
 **if** x > 0:  
 *# print('Can move to UP,after move result is:')* crtUp = exchange(src, x, y, x - 1, y)  
 *# prtNum(crtUp)* upDiff = diff(numberFinal, crtUp)  
 *# print('different factor is',upDiff)  
 # return(crtUp,src,upDiff)* nodeid = nodeid + 1  
 rtResult.append([crtUp, src, upDiff, nodeid])  
 *# else:  
 # print('Cannot move to UP!')* **if** side == **'down' or** side == **''**:  
 **if** x < 2:  
 *# print('Can move to DOWN,after move result is:')* crtDown = exchange(src, x, y, x + 1, y)  
 *# prtNum(crtDown)* downDiff = diff(numberFinal, crtDown)  
 *# print('different factor is',downDiff)  
 # return(crtDown,src,downDiff)* nodeid = nodeid + 1  
 rtResult.append([crtDown, src, downDiff, nodeid])  
 *# else:  
 # print('Cannot move to DOWN!')* **if** nodeid % 1000 >= 0 **and** nodeid % 1000 < 3:  
 print(int(nodeid / 1000) \* 1000, **'nodes analyzed!'**)  
 **return** rtResult  
  
  
*# 打印结果，方法是从close表最后一条开始，查找其前一个节点，  
# 直到前一节点为0，并将所有查到的序列写入step，打印出step  
# 即得到所有的变化过程。***def** prtResult():  
 step = [closed[-1]]  
 nodePrt = closed[-1][4]  
 **while True**:  
 **for** x **in** range(len(closed)):  
 **if** nodePrt == closed[x][3]:  
 step.insert(0, closed[x])  
 nodePrt = closed[x][4]  
 **if** nodePrt == 0:  
 **break  
 for** x **in** range(len(step)):  
 print(**'Step'**, x, **':'**)  
 prtNum(step[x][0])  
 print(**'Finished!'**)  
 time.sleep(10)  
  
  
*# numberOrig=['1','2','3','4','7',' ','6','5','8']  
  
# numberOrig=[' ','7','2','5','1','6','8','3','4']  
  
# numberOrig=['4','1','6','7','2','8','5',' ','3']  
  
# numberFinal=['1','2','3','4','5','6','7','8',' ']*open = []  
  
closed = []  
  
nodeid = 1  
  
*# 主程序  
# 输入初始和目标序列，并打印出来供确认，如不正确可重新输入***while True**:  
 print(**'Please input Original state:'**, end=**'\t'**)  
 tmp = input()  
 numberOrig = [tmp[0], tmp[1], tmp[2], tmp[3], tmp[4], tmp[5], tmp[6], tmp[7], tmp[8]]  
 print(**'Please input Final state:'**, end=**'\t'**)  
 tmp = input()  
 numberFinal = [tmp[0], tmp[1], tmp[2], tmp[3], tmp[4], tmp[5], tmp[6], tmp[7], tmp[8]]  
 print(**'Orig is'**)  
 prtNum(numberOrig)  
 *# print('Orig judge is',judge(numberOrig))* print(**'Final is'**)  
 prtNum(numberFinal)  
 *# print('Final judge is',judge(numberFinal))* print(**'Is it correct?'**, end=**'\t'**)  
 confirm = input()  
 **if** confirm == **'y'**:  
 **break***# 如果初始和目标序列的判定值奇偶性一致，则存在解，开始处理***if** (judge(numberOrig) + judge(numberFinal)) % 2 == 0:  
 print(**'Have answer! Orig is '**, judge(numberOrig), **', Final is'**, judge(numberFinal))  
 *# 处理方式：将初始节点加入open表，开始处理。* open.append([numberOrig, **'NULL'**, diff(numberOrig, numberFinal), 1, 0])  
 handleOpen()  
*# 否则，不存在解，直接退出。***else**:  
 print(**'No answer! Orig is '**, judge(numberOrig), **', Final is'**, judge(numberFinal))