图搜索实验报告

张栋玮 19373703　在Github中打开

1. 定义位置与目标数码

可以在测试部分修改目标数码。

In [1]:
```
# 位置定义：
#     0 1 2
#     3 4 5
#     6 7 8
# 目标数码：
#     1 2 3
#     8 0 4
#     7 6 5
```

2. 定义规则与交换算法

格式如下：{position:[exchangeable position]}，比如位置0可以和[1,3]交换。在本实验中，采用字符串记录各个节点，如目标数码为“123804765”。

In [2]:
```
chart = {0:[1,3],
         1:[2,4,0],
         2:[1,5],
         3:[0,4,6],
         4:[1,3,5,7],
         5:[2,4,8],
         6:[3,7],
         7:[4,6,8],
         8:[5,7]}
```

In [3]:
```
# 交换两个方块，a为当前数码阵，i、j为要交换的两个位置
def swap(a, i, j):
    if i > j:
        #将i,j变换顺序，便于后面拷贝字符串
        i, j = j, i
    b = a[:i] + a[j] + a[i+1:j] + a[i] + a[j+1:]
    return b
```

3. 定义启发函数（A*）

在A*算法中，定义启发函数total_cost=future_cost+deep，其中future_cost为当前状态和目标状态的曼哈顿距离总和，deep为当前状态的探索深度。

In [4]:
```
# 计算当前状态和目标状态的曼哈顿距离
def future_cost(src,dst):
    sum = 0
    a = src.index("0")
```

```
    for i in range(0,9):
        if i!=a:
            sum=sum+abs(i-dst.index(src[i]))
    return sum

def total_cost(deep,new,dst):
    val = future_cost(new, dst)+deep
    return val
```

4. 判断是否可解

对于八数码问题，如果两个状态的逆序数奇偶性相同，那么这两个状态可以相互到达，否则不能相互到达。参见八数码问题。

In [5]:
```
# 计算逆序数，判断是否可解
def inv_num(s):
    count = 0
    for i in range(1,9):
        inv=0
        for j in range(0,i):
            if(s[j]>s[i] and s[i]!='0'):
                inv+=1
        count+=inv
    return count
```

5. 算法实现

使用solve(src, dst, method)函数同时实现深度优先、广度优先、A*算法，分别对应method=1,2,3。三者的主要区别在于从Open表取出一个节点的方法，深度优先采用的是后进先出，栈型数据结构；广度优先采用的是先进先出，队列型数据结构；*A*算法是根据Open表中的启发函数值，每次给出最小的节点。当扩展至出现目标节点时，就完成了搜索。最后通过steps将步骤返回。

In [6]:
```
def solve(src, dst, method=2):

    # Open表，以整个字符串为一个表中元素
    # 后续通过查找“0”所在位置来确定可移动方向
    openBox = []
    openBox.append(src)#当前状态存入列表

    # Closed表，以字典记录
    # 便于查找是否已扩展，并且记录父节点
    closedBox = {}
    closedBox[src] = -1

    deep = {}
    deep[src]= 1

    # Open表中各节点的总距离
    openDeep = {}
    openDeep[src] = 1 + future_cost(src, dst)

    count=0

    while len(openBox) > 0:
        # count+=1
        # print("No.",count)
```

```
        if(method==1):
            cur = openBox.pop() # 后进先出，深度优先
        elif(method==2):
            cur = openBox.pop(0) # 先进先出，广度优先
        else:
            #找到当前Open表中总距离最小的节点
            cur = min(openDeep, key=openDeep.get)
            # print("cur:",cur)
            del openDeep[cur]
            openBox.remove(cur)


        if cur == dst: #判断当前状态是否为目标状态
            break

        # 寻找0的位置并进行扩展
        init = cur.index("0")
        Available = chart[init]#当前可进行交换的位置集合
        for move in Available:
            new = swap(cur, move, init) #交换后的状态
            if closedBox.get(new) == None:#判断交换后的状态是否已经查询过
                closedBox[new] = cur      #当前扩展节点加入Closed表
                openBox.append(new)       #新节点加入Open表
                if(method==3):
                    val = total_cost(deep[cur] + 1, new, dst)
                    deep[new] = deep[cur] + 1#存入距离
                    openDeep[new] = val#存入val


    # 输出
    steps = []
    steps.append(cur)
    # 回溯至根节点src
    while closedBox[cur] != -1:
        cur = closedBox[cur]
        steps.append(cur)
    steps.reverse()
    return steps
```

6. 测试

src为初始状态，dst为目标状态，method为选用方法。

In [7]:
```
if __name__ == "__main__":

    # 测试数据，可在此进行修改：
    src = "541203786"
    dst = "123804765"
    method = 3
    # 方法1：深度优先
    # 方法2：广度优先
    # 方法3：A*算法

    assert method in [1,2,3],"方法错误，请重新输入！"

    # 判断是否可解
    inv1 = inv_num(src)
    inv2 = inv_num(dst)
    assert inv1%2==inv2%2,"不可解！请重新开始！"

    # 获取解步骤
```

```
    steps = solve(src, dst, method)

    # 输出打印
    for i in range(len(steps)):
        print(f"--Steps:{i+1}--")
        print(steps[i][:3])
        print(steps[i][3:6])
        print(steps[i][6:])
```

--Steps:1--
541
203
786
--Steps:2--
501
243
786
--Steps:3--
510
243
786
--Steps:4--
513
240
786
--Steps:5--
513
204
786
--Steps:6--
513
024
786
--Steps:7--
013
524
786
--Steps:8--
103
524
786
--Steps:9--
123
504
786
--Steps:10--
123
584
706
--Steps:11--
123
584
760
--Steps:12--
123
580
764
--Steps:13--
123
508
764
--Steps:14--
123
058
764
--Steps:15--
123
758
064
--Steps:16--
123
758
604

--Steps:17--
123
708
654
--Steps:18--
123
780
654
--Steps:19--
123
784
650
--Steps:20--
123
784
605
--Steps:21--
123
784
065
--Steps:22--
123
084
765
--Steps:23--
123
804
765