实验一：启发式搜索解决八数码问题

一、实验目的

通过解决八数码问题，掌握启发式搜索（A * 算法）的定义、估价函数设计及算法流程，理解如何利用启发信息减少搜索范围，提高搜索效率。

二、实验内容

使用 A * 算法解决八数码问题。在 3×3 棋盘上，通过空格移动（左、上、右、下）将初始状态转换为目标状态。

初始状态（S0）：

```
2 8 3
1 6 4
7   5
```

目标状态（Sg）：

```
1 2 3
8   4
7 6 5
```

三、实验原理

1. **启发式搜索与 A * 算法**
   - 启发式搜索利用估价函数 (f(n) = g(n) + h(n)) 引导搜索，其中：
     - (g(n)) 为从初始状态到当前状态的实际代价（深度）；
     - (h(n)) 为当前状态到目标状态的启发式估计代价（曼哈顿距离之和）。
   - 当 (h(n) <= h^(n))（(h^(n)) 为最优路径代价）时，A * 算法保证找到最短路径。
2. **数据结构**
   - **Open 表**：优先队列，按 (f(n)) 从小到大存储待扩展节点。
   - **Closed 表**：集合，记录已扩展节点的状态，避免循环。
3. **估价函数设计**
   - 采用 (h(n) =) 各数字（除空格）到目标位置的曼哈顿距离之和，满足 (h(n) <= h^*(n))，确保 A * 算法正确性。

四、实验步骤

1. **定义节点类**：存储状态、父节点、g、h、f 值。
2. **初始化 Open 表**：将初始状态加入 Open 表。
3. **循环扩展节点**：
   - 从 Open 表中取出 f 最小的节点。
   - 若为目标状态，回溯路径并结束。
   - 生成子节点（空格左、上、右、下移动），计算 g 和 h，更新 Open 表和 Closed 表。

4. **路径回溯**：从目标节点反向追踪父节点，得到移动路径。

五、实验结果

1. **节点扩展顺序（状态图）**

```
S0 (f=5)
├─ 左移: 2 8 3 → 2 8 3    （非法，跳过）
├─ 上移: 2 8 3 → 2  3     （状态: 2 8 3, f=6）
|        1 6 4    1 6 4
|        7   5    7 8 5
├─ 右移: 2 8 3 → 2 8 3    （状态: 2 8 3, f=5）
|        1 6 4    1 6 5
|        7   5    7  4
└─ 下移: 2 8 3 → 2 8 3    （状态: 2 8 3, f=5）
         1 6 4    1 6 4
         7   5    7 5
```

六、实验总结

通过 A * 算法成功解决八数码问题，验证了启发式搜索的高效性。估价函数的设计直接影响搜索效率，曼哈顿距离作为启发式函数能有效减少搜索空间，确保找到最优解。实验中需注意状态去重和优先队列的正确实现。

完整代码如下：

```python
import heapq

# 目标状态（0代表空格）
GOAL = [
    [1, 2, 3],
    [8, 0, 4],
    [7, 6, 5]
]

class Node:
    def __init__(self, state, parent=None, g=0):
        self.state = state  # 当前状态（二维列表）
        self.parent = parent  # 父节点
        self.g = g  # 已走步数（g(n)）
        self.h = self.calculate_h()  # 启发式函数值（h(n)）
        self.f = self.g + self.h  # 估价函数值（f(n)）

    def calculate_h(self):
        """计算曼哈顿距离之和（h(n)）"""
        h = 0
        for i in range(3):
            for j in range(3):
                num = self.state[i][j]
                if num == 0:
                    continue
                # 目标位置
                goal_i, goal_j = divmod(num-1, 3)
                h += abs(i - goal_i) + abs(j - goal_j)
```

```
        return h

    def generate_children(self):
        """生成子节点（空格左、上、右、下移动）"""
        children = []
        # 找到空格位置
        for i in range(3):
            for j in range(3):
                if self.state[i][j] == 0:
                    si, sj = i, j
                    break
            else:
                continue
            break
        # 移动方向（左、上、右、下）
        directions = [(-1, 0), (0, -1), (1, 0), (0, 1)]
        for di, dj in directions:
            ni, nj = si + di, sj + dj
            if 0 <= ni < 3 and 0 <= nj < 3:
                # 复制状态并交换
                new_state = [row.copy() for row in self.state]
                new_state[si][sj], new_state[ni][nj] = new_state[ni][nj],
new_state[si][sj]
                children.append(Node(new_state, self, self.g + 1))
        return children

    def get_path(self):
        """回溯路径"""
        path = []
        current = self
        while current:
            path.append(current.state)
            current = current.parent
        return reversed(path)

def a_star(initial_state):
    """A*算法主函数"""
    initial_node = Node(initial_state)
    open_heap = []
    heapq.heappush(open_heap, (initial_node.f, id(initial_node), initial_node))
    closed = set()

    while open_heap:
        current_f, _, current_node = heapq.heappop(open_heap)
        state_tuple = tuple(tuple(row) for row in current_node.state)
        if state_tuple in closed:
            continue
        closed.add(state_tuple)
        # 检查是否为目标状态
        if current_node.state == GOAL:
            return current_node.get_path()
        # 生成子节点
        for child in current_node.generate_children():
            child_state = tuple(tuple(row) for row in child.state)
            if child_state not in closed:
                heapq.heappush(open_heap, (child.f, id(child), child))
```

```
    return None  # 无解

# 初始状态（注意空格用0表示）
INITIAL = [
    [2, 8, 3],
    [1, 6, 4],
    [7, 0, 5]
]

if __name__ == "__main__":
    path = a_star(INITIAL)
    if path:
        print("找到路径，共{}步：".format(len(list(path))-1))
        for i, state in enumerate(path):
            print(f"步骤{i}:")
            for row in state:
                print(row)
            print()
    else:
        print("无解")
```

八数码解过程节点状态图

1. 初始状态

```
2 8 3
1 6 4
7 0 5
f = 0（g） + 5（h） = 5
```

2. 扩展子节点
   - 下移：空格与下方 5 交换

     ```
     2 8 3
     1 6 5
     7 0 4
     f = 1 + 4 = 5
     ```

   - 右移：空格与右侧 4 交换（非法，边界）
   - 上移：空格与上方 6 交换

     ```
     2 8 3
     1 0 4
     7 6 5
     f = 1 + 4 = 5
     ```

   - 左移：空格左侧无元素（非法）
3. **选择 f 最小的节点（任意一个 f=5 的节点）继续扩展**，直至到达目标状态。