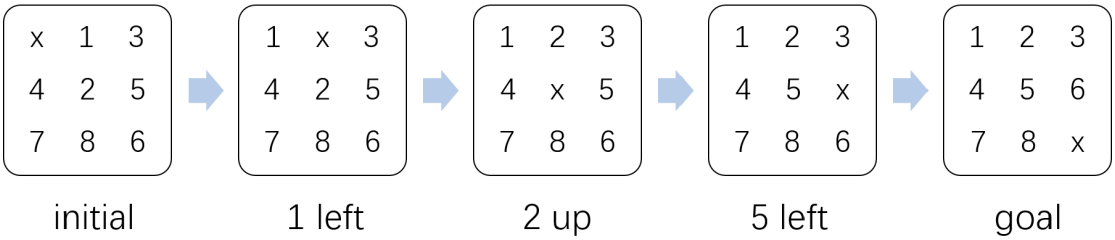
苏 州 大 学 实 验 报 告

院、系	计算机学院	年级专业	19 计科图灵	姓名	张昊	学号	1927405160
课程名称	人工智能与知识工程实验					成绩	
指导教师	陈文亮	同组实验者	无		实验日期	2021/12/9	

实 验 名 称 8 数码问题

一. 实验题目

8 数码问题。一块 3*3 的拼图，由 8 个正方形滑块和一个空档组成。每个滑块上标有[1, 15]的一个整数，假设空档为 x。你的目标是使用尽量少次数的移动，来重新组织这些滑块，使拼图还原。仅允许水平地或竖直地将滑块移到空档内。下面展示一个还原的合法移动序列。

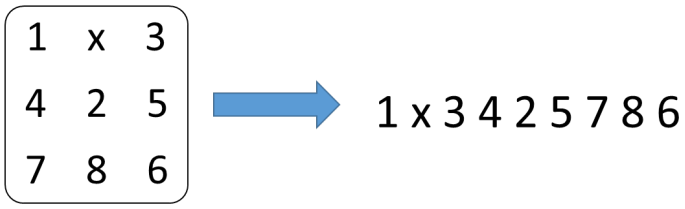


要求：支持连续输入、求解；使用 A*搜索算法解题。

输入

3*3 拼图：一个以空格隔开的字符串表示

（相当于序列展开）



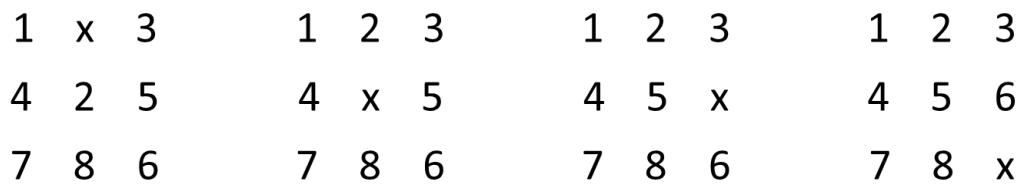
输出

若不可解，输出“无法搜到有效解”

若可解，输出

1、最少移动次数：3 次

2、每一步的移动过程



二. 实验过程

1. 拼图的实现

（1）初始化

根据题目要求，一个 3*3 拼图用一个以空格隔开的字符串来表示。所以只需将其分割并重新展开到二维即可。特别地，使用一个 5*5 的二维矩阵来保存，行和列的下标索引从 1 开始（相当于在 3*3 的拼图外加一圈围栏），可以有效简化访问拼图边界时的特殊判断。

```
def __init__(self, initialize: str, level=0):
    graph = initialize.strip().split()
    assert len(graph) == 9 and graph.count('x') == 1
    self._graph = [[None for _ in range(5)] for _ in range(5)]
    self._level = level  # 保存是第几步对应的结点，即为函数 g(x)
    self.x_positon = [None, None]
    for i, v in enumerate(graph):
        self._graph[i // 3 + 1][i % 3 + 1] = v
        if v == 'x':
            self.x_positon = [i // 3 + 1, i % 3 + 1]
```

（2）有效解存在性的判断

对于任意给定的 3*3 拼图，可以证明逆序数为奇数的 3*3 拼图不可解。因此，只需对任一对滑块 `graph[i]`和 `graph[j]`，判断是否有 `i < j` 但 `graph[i] > graph[j]`，即可计算得到逆序数，再判断奇偶性即可。具体实现中，将滑块重新展开为一维数组，并去掉 x 字符，两两枚举即可。

```
def has_efficient_solution(self) -> bool:
    numbers = list(map(int, filter(lambda x: x != 'x', self.value_list)))
    reversed_count = 0
    for i, j in itertools.combinations(range(len(numbers)), 2):
        if i < j and numbers[i] > numbers[j]:
            reversed_count += 1
    return reversed_count % 2 == 0
```

（3）节点到目标结点的曼哈顿距离

节点到目标结点的曼哈顿距离 = 每个滑块到目标位置的曼哈顿距离之和

每个滑块到目标位置的曼哈顿距离 = 滑块当前位置与目标位置的横纵坐标对应之差的绝对值之和

目标结点一般为：

1	2	3
4	5	6
7	8	x

因此，使用了一个静态变量来保存每个目标结点中滑块的位置。在对任一个拼图节点中的滑块计算曼哈顿距离时，直接查表得到目标结点中滑块的位置即可。（如此计算得到的是$h(x)$：节点 x 到目标结点的一条最佳路径的代价。）具体实现如下：

```
@property
def h(self) -> int:
    manhattan = 0
    for i in range(1, 4):
```

```
        for j in range(1, 4):
            best_i, best_j = self.best_value_map[self._graph[i][j]]
            if best_i is None:
                continue
            manhattan += abs(i - best_i) + abs(j - best_j)
    return manhattan
```

（4）获得后继结点

定义了方向数组，保存了允许的四个方向（上下左右），对当前 x 字符所在的位置（已经预先保存）分别加上该方向，得到新的 x 字符位置。若新的 x 字符位置到了边界则说明此路不通，否则产生后继结点：首先产生一个同本结点一样的子代结点（但步数加一），之后交换两个 x 的位置，最后保存新的 x 字符位置。具体实现如下：

```
def move(self):
    childern = []
    directions = ((0, 1), (1, 0), (0, -1), (-1, 0))
    for direction in directions:
        new_x_position = [self.x_positon[0] + direction[0],
                          self.x_positon[1] + direction[1]]
        if self._graph[new_x_position[0]][new_x_position[1]] is None:  # 达到边界
            continue
        new_graph = self.clone()
        new_graph._graph[self.x_positon[0]][self.x_positon[1]], \
            new_graph._graph[new_x_position[0]][new_x_position[1]] = \
            new_graph._graph[new_x_position[0]][new_x_position[1]], \
            new_graph._graph[self.x_positon[0]][self.x_positon[1]]
        new_graph.x_positon = new_x_position
        childern.append(new_graph)
    return childern
```

（5）估价函数

$$f(x) = g(x) + h(x)$$

- $g(x)$定义为节点S_0到 x 的一条最佳路径的实际代价。在这里为已经移动的次数，使用属性 level 来维护。
- $h(x)$定义为节点 x 到目标结点的一条最佳路径的代价。在这里为节点 x 和目标结点的曼哈顿距离，使用上述（3）节的方法来实现。

2. A*算法流程控制的实现

为支持连续输入、求解，控制函数在一个死循环中不断等待用户输入并求解，直到用户输入了字符 q，程序才会结束运行。

对于用户输入，算法首先会利用该字符串建立初始结点，并验证该结点是否为可解的。可以证明，任何合法的移动不会改变拼图的逆序数的奇偶性，故判断初始结点的奇偶性即能够提前判断这一输入是否存在解，从而避免陷入无解的死循环。若无解则直接输出“无法搜到有效解”；否则进行迭代，在迭代过程中按上一小节定义的估价函数$f(x)$选取预估代价最小的结点作为路线，保存到历史记录中，并维护一个迭代轮次的变量，直到得到目标结点（即$h(x) = 0$时）。

```
def a_star():
    while True:
        inputs = input('请输入一个以空格隔开的字符串表示的 3*3 拼图（输入 q 退出）：')
```

```
        if inputs.strip() == 'q':
            return
        init_graph = Graph(inputs)
        if not init_graph.has_efficient_solution():
            print('无法搜到有效解')
            continue
        history = [init_graph]
        move_count = 0
        graph = init_graph
        while not graph.is_best_solution():
            next_graphs = graph.move()
            assert len(next_graphs) != 0
            graph = min(next_graphs, key=lambda g: g.f)
            move_count += 1
            history.append(graph)
        print('最少移动次数:', move_count)
        print('每一步的移动过程:')
        for i, g in enumerate(history):
            print('Step', i)
            print(g)
            print('-' * 6)
```

三. 实验结果

- 运行环境：Python 3.8
- 运行方法：运行 main.py 即可

运行结果：

```
请输入一个以空格隔开的字符串表示的 3*3 拼图（输入 q 退出）：1 x 3 4 2 5 7 8 6
最少移动次数: 3
每一步的移动过程:
Step 0
1 x 3
4 2 5
7 8 6
------
Step 1
1 2 3
4 x 5
7 8 6
------
Step 2
1 2 3
4 5 x
7 8 6
```

```
------
Step 3
1 2 3
4 5 6
7 8 x
------
请输入一个以空格隔开的字符串表示的 3*3 拼图（输入 q 退出）：1 2 3 5 4 6 7 8 x
无法搜到有效解
请输入一个以空格隔开的字符串表示的 3*3 拼图（输入 q 退出）：1 3 x 4 2 5 7 8 6
最少移动次数： 4
每一步的移动过程：
Step 0
1 3 x
4 2 5
7 8 6
------
Step 1
1 x 3
4 2 5
7 8 6
------
Step 2
1 2 3
4 x 5
7 8 6
------
Step 3
1 2 3
4 5 x
7 8 6
------
Step 4
1 2 3
4 5 6
7 8 x
------
请输入一个以空格隔开的字符串表示的 3*3 拼图（输入 q 退出）：q
```

四. 实验总结和反思

本次实验加深了我对 A*搜索算法的理解，掌握了利用 A*搜索算法解决实际问题的能力，进一步加深了我对 8 数码问题的认识。