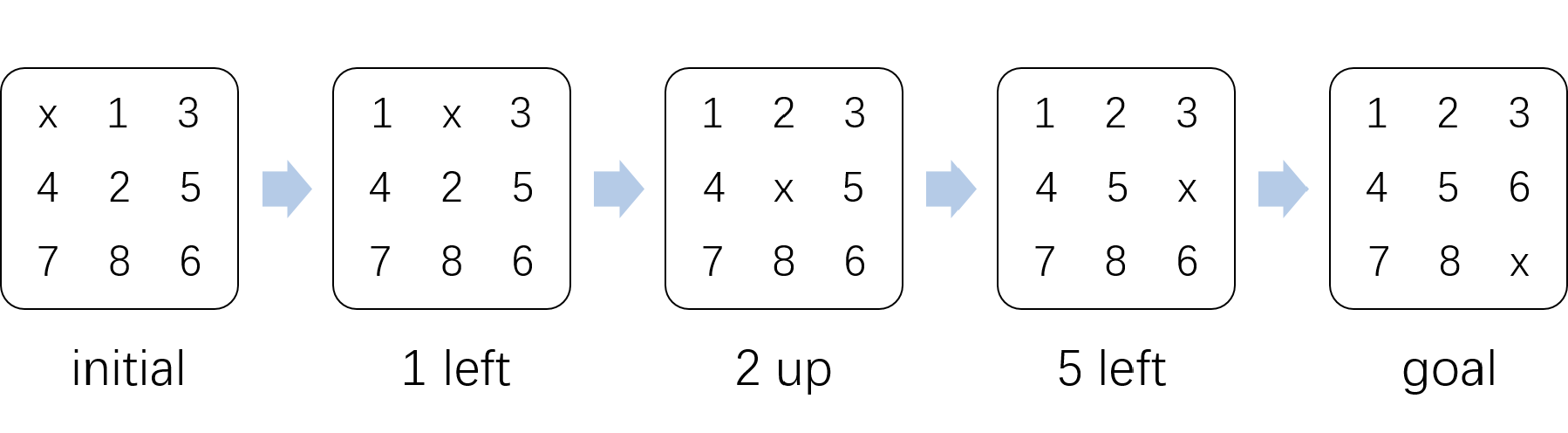
苏州大学实验报告

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 |

1. 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。

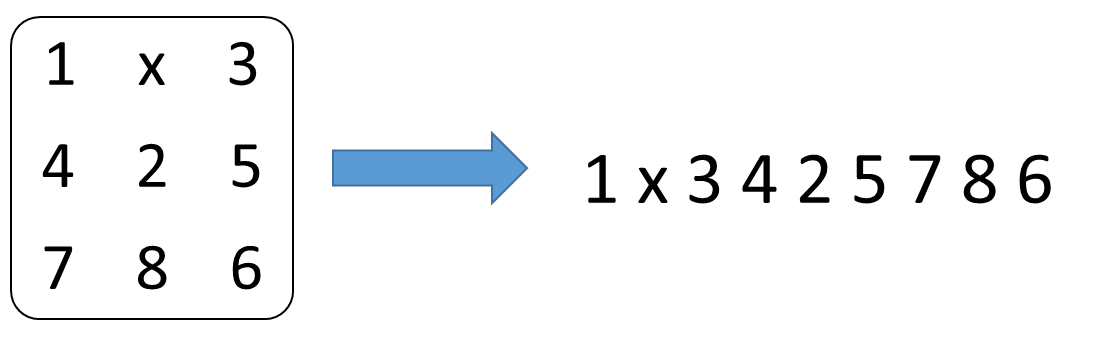


要求：支持连续输入、求解；使用A\*搜索算法解题。

**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）



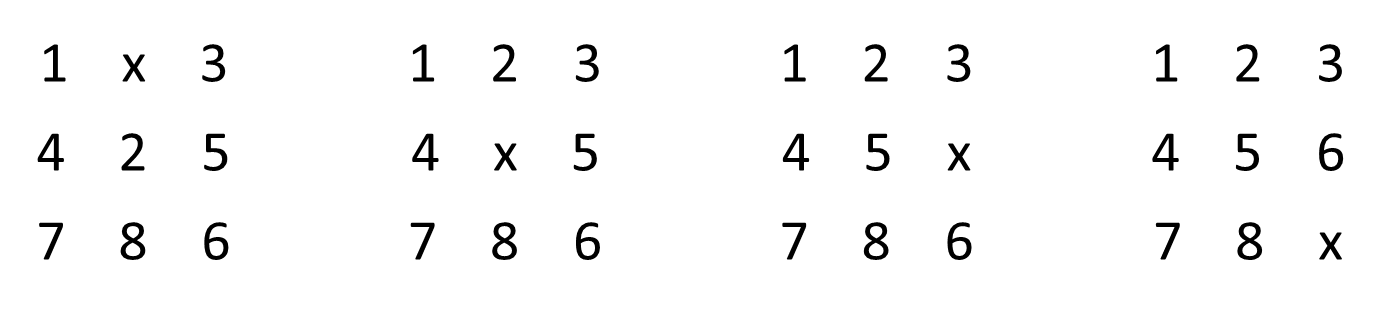
**输出**

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



1. 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   
 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:

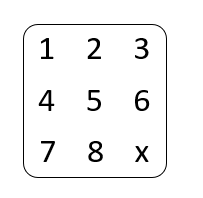
numbers = list(map(int, filter(lambda x: x != 'x', self.value\_list)))  
 reversed\_count = 0  
 for i, j in itertools.combinations(range(len(numbers)), 2):  
 if i < j and numbers[i] > numbers[j]:  
 reversed\_count += 1  
 return reversed\_count % 2 == 0

（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：



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：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

* 定义为节点到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。
* 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)

1. 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

1. 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。