

电子科技大学
计算机科学与工程学院

标准实验报告

(实验) 课程名称 人工智能

电子科技大学

实验报告

学生姓名：朱若愚 学号：2022150501027 指导教师：段立新

实验地点：A2-412

实验时间：2024.5.19

一、实验室名称： 计算机学院实验中心

二、实验项目名称：A*算法实验

三、实验学时：5 学时

四、实验原理：

1. A*算法：A*（A-star）算法是一种在图中寻找从初始节点到目标节点最短路径的启发式搜索算法。它结合了 Dijkstra 算法的确保性（保证找到一条最短路径）和贪心算法的高效性（快速找到目标）。A* 算法通过评估函数 $f(n) = g(n) + h(n)$ 来工作，其中 $g(n)$ 是从起始点到任何顶点 n 的实际成本，而 $h(n)$ 是从顶点 n 到目标的估计最低成本，通常用启发式函数来计算，这个函数需要事先设计来反映实际的地形或环境特征。理想情况下， $h(n)$ 应该不会高估实际的成本，这种情况下，A* 算法保证找到一条最低成本路径。算法的性能和准确性高度依赖于启发式函数的选择。在实际应用中，A* 算法广泛应用于各类路径规划问题，如机器人导航、地图定位服务和游戏中的 AI 路径寻找等场景。通过适当选择和调整启发式函数，A* 算法能够在复杂的环境中有效地寻找最短路径，同时保持计算上的可行性和效率。

2. 八数码问题：八数码问题也叫九宫问题，是人工智能中状态搜索中的经典问题，其中，该问题描述为：在 3×3 的棋盘，摆有八个棋子，每个棋子上标有 1 至 8 的某一数字，不同棋子上标的数字不相同。棋盘上还有一个空格，与空格相邻的棋子可以移到空格中。要求解决的问题是：给出一个初始状态和一个目标状态，找出一种从初始转变成目标状态的移动棋子步数最少的移动步骤。

五、实验目的：

熟悉和掌握启发式搜索的定义、估价函数和算法过程，并利用 A*算法求解 N 数码难题，理解求解流程和搜索顺序。

六、实验内容：

1. 以 8 数码问题为例实现 A*算法的求解程序（编程语言不限），设计估价函数。

注：需在实验报告中说明估价函数，并附对应的代码。

2. 设置初始状态和目标状态，针对估价函数，求得问题的解，并输出移动过程。

要求：

(1) 提交源代码及可执行文件。

(2) 提交实验报告，内容包括：对代码的简单说明、运行结果截图及说明等。

七、实验器材（设备、元器件）：

PC 微机一台

八、实验步骤：

1. 定义节点 class Node:

```
2 class Node:
3     def __init__(self, matrix=None, x=0, y=0, g=0, h=0, father=None, flag=None):
4         if matrix is None:
5             matrix = []
6             for _ in range(3):
7                 row = []
8                 for _ in range(3):
9                     row.append(0)
10                matrix.append(row)
11            self.a = matrix
12            self.x = x
13            self.y = y
14            self.g = g
15            self.h = h
16            self.f = g + h
17            self.father = father
18            self.flag = flag
19
20        def __lt__(self, other):
21            return self.f < other.f
```

定义一个 Node 类用于存储包括矩阵，0 的位置，g、h 的值以及他们的和 f 等。

2. 矩阵输入函数 input_mat:

```
24 def input_mat(input_mat):
25     print(input_mat)
26     mat = []
27     zero_x = zero_y = -1
28     for i in range(3):
29         row = list(map(int, input().split()))
30         mat.append(row)
31         for j in range(len(row)):
32             if row[j] == 0:
33                 zero_x, zero_y = i, j
34     return mat, zero_x, zero_y
```

接收用户输入的八数码图存储入矩阵，并且检测 0 的位置，返回这个举证以及 0 的坐标信息。

3. 判断是否有解函数 judge:

```
37 def judge(start_mat, goal_mat):
38     start1 = []
39     for sublist in start_mat:
40         for item in sublist:
41             if item != 0:
42                 start1.append(item)
43
44     goal1 = []
45     for sublist in goal_mat:
46         for item in sublist:
47             if item != 0:
48                 goal1.append(item)
49
50     start2 = 0
51     for i in range(len(start1)):
52         for j in range(i):
53             if start1[j] > start1[i]:
54                 start2 += 1
55
56     goal2 = 0
57     for i in range(len(goal1)):
58         for j in range(i):
59             if goal1[j] > goal1[i]:
60                 goal2 += 1
61
62     return (start2 % 2) == (goal2 % 2)
```

八数码问题有解的数学原理：当且仅当初始状态和目标状态的逆序数的奇偶性相同时，可以通过若干移动由初始状态变化为目标状态。

- 1) 分别为初始状态和目标状态创建列表存储 0 以外的所有元素。
- 2) 分别计算两个列表的逆序数。
- 3) 判断计算后的两个逆序数的奇偶性，如果相同，返回 **True**，表示问题有解；否则，返回 **False**，表示问题无解。

4. 估价函数 h:

```
65 def h(node, goal):
66     h = 0
67     goal_pos1 = {}
68     for i in range(3):
69         for j in range(3):
70             goal_pos1[goal[i][j]] = (i, j)
71     for i in range(3):
72         for j in range(3):
73             if node.a[i][j] != 0:
74                 goal_pos2 = goal_pos1[node.a[i][j]]
75                 h = h + abs(goal_pos2[0] - i) + abs(goal_pos2[1] - j)
76     return h
```

计算从当前状态到目标状态的曼哈顿距离来实现计算最小代价。通过 for 循环遍历每个元素当前位置到目标位置的曼哈顿距离，并求和赋值给 h。

5. 打印步骤函数 print_step:

```
79 def print_step(node):
80     step = 0
81     path = ""
82     nodes = []
83     while node is not None:
84         nodes.append(node)
85         node = node.father
86     for node in reversed(nodes):
87         path += f"Step {step}: \n"
88         for row in node.a:
89             path += ' '.join(map(str, row)) + "\n"
90         path += "-----\n\n"
91         step += 1
92     return step, path
```

输出当前状态并计数打印步骤数。

6. 生成接下来所有可能移动结果子节点函数 `generate_children`:

```
95 def generate_children(now, goal):
96     dir = [(-1, 0), (0, 1), (1, 0), (0, -1)]
97     opposite_flags = [2, 3, 0, 1]
98     x = now.x
99     y = now.y
100     chil = []
101
102     for index, (dx, dy) in enumerate(dir):
103         nx = x + dx
104         ny = y + dy
105         if 0 <= nx < 3 and 0 <= ny < 3 and index != now.flag:
106             new_mat = [row[:] for row in now.a]
107             temp = new_mat[x][y]
108             new_mat[x][y] = new_mat[nx][ny]
109             new_mat[nx][ny] = temp
110             new_node = Node(new_mat, nx, ny, now.g + 1, h(Node(new_mat), goal), now, opposite_flags[index])
111             chil.append(new_node)
112     return chil
```

对于当前 0 所在位置的四个方向移动后的新位置，先检测其位置是否合法，并计算他的 g、h。最后将节点添加至 chil 列表返回当前节点的所有可能性子节点。

7. 寻找 0 位置函数 `find_zero`:

```
115 def find_zero(mat):
116     for i in range(3):
117         for j in range(3):
118             if mat[i][j] == 0:
119                 return i, j
```

遍历矩阵，寻找 0 的位置，返回坐标即行号和列号。

8. A*核心实现算法函数 `a_star`:

```
122 def a_star(start_mat, goal_mat):
123     start_node = Node(start_mat, *find_zero(start_mat), 0, h(Node(start_mat), goal_mat))
124     open_list = []
125     heappush(open_list, start_node)
126     visited = set()
127
128     while open_list:
129         current_node = heappop(open_list)
130         if current_node.a == goal_mat:
131             steps, path = print_step(current_node)
132             print(path)
133             print("\n共计 {} 步完成".format(steps - 1))
134             return
135         visited.add(tuple(tuple(row) for row in current_node.a))
136
137         for child in generate_children(current_node, goal_mat):
138             if tuple(tuple(row) for row in child.a) not in visited:
139                 heappush(open_list, child)
140
141     print("问题无解。")
```

- 1) 创建初始节点 `start_node` 代表起始状态、空优先队列 `open_list` 存储待访问节点，空集合 `visited` 存储已经访问过的状态节点。
- 2) 执行循环，当 `open_list` 不为空时，从中取出 f 最小的节点作为当前节点。
- 3) 检查当前节点是否为目标节点，若是打印步骤；若否将改节点添加到 `visited` 中并生成当前节点的所有可能性子节点并且存入 `open_list` 中。
- 4) 执行循环，直至 `open_list` 为空。

九、实验数据及结果分析：

测试样例一：

```
C:\Users\Albert\PycharmProjects\8_star\venv\Scripts\python.exe C:\Users\Albert\PycharmProjects\8_star\8_star.py
输入起始八数码图:
1 0 2
3 4 5
6 7 8
输入目标八数码图:
1 2 3
4 5 6
7 8 0
Step 0:
1 0 2
3 4 5
6 7 8
-----

Step 1:
1 4 2
3 0 5
6 7 8
-----

Step 2:
1 4 2
0 3 5
6 7 8
-----

Step 3:
1 4 2
6 3 5
0 7 8
-----

Step 4:

7 4 5
0 0 6
-----

Step 18:
1 2 3
0 4 5
7 8 6
-----

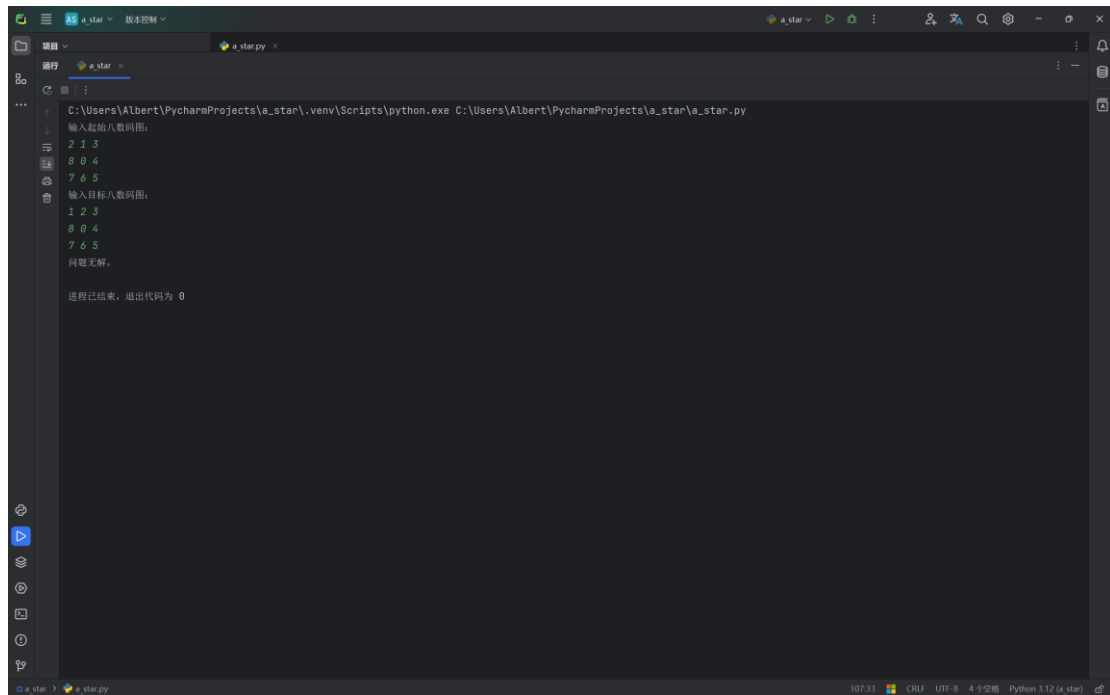
Step 19:
1 2 3
4 0 5
7 8 6
-----

Step 20:
1 2 3
4 5 0
7 8 6
-----

Step 21:
1 2 3
4 5 6
7 8 0
-----

共计 21 步完成
进程已结束，退出代码为 0
```

测试样例二：



```
C:\Users\Albert\PycharmProjects\A_star\.venv\Scripts\python.exe C:\Users\Albert\PycharmProjects\A_star\A_star.py
输入起始八数码图:
2 1 3
8 0 4
7 6 5
输入目标八数码图:
2 2 3
8 0 4
7 6 5
问题无解。

进程已结束，退出代码为 0
```

十、实验结论：

A* 算法是一种强大且灵活的路径搜索算法，适用于各种从点 A 到点 B 的最短路径搜索问题。通过合理选择启发式函数，A* 算法不仅能保证找到最短路径，还能在执行过程中保持高效。通过使用 A*算法在八数码问题中的应用，可以方便快捷求出最优解。

十一、总结及心得体会：

1. 合理选择启发函数，可以有效提高 A*算法的效率和效果，如果不能很好地选择启发函数，则有可能得不到最优解。
2. A*算法需要维护列表存储信息，在大型问题中可能会消耗大量内存。

十二、对本实验过程及方法、手段的改进建议：

1. 在计算逆序数的部分，我们可以使用归并排序来计算逆序数，以提高效率。
2. 在生成子节点时可检测改节点是都被访问，如果已经被访问可以不再需要生成。

报告评分：

指导教师签字：