# 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

电子科技大学教务处制表

电 子 科 技 大 学

# 电 子 科 技 大 学

# 实 验 报 告

**学生姓名：李天　　学 号：2020080904021**

**指导教师：段立新 张彦如 顾实**

**实验地点： 主楼 A2-413-1　　实验时间：2022.12.3**

## 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：

A*搜索算法利用启发性信息来引导搜索，动态地确定搜索节点的排序，每次选择最优的节点往下搜索，可以高效地减少搜索范围，减少求解的问题的时间。

A*算法用估价函数来衡量一个节点的优先度，优先度越小的节点越应该被优先访问。估价函数用如下公式表示：$f(n) = h(n) + d(n)$，$h(n)$是对于当前状态到目标状态的估计值，例如曼哈顿距离，$d(n)$是已经付出的搜索代价，例如节点的深度。A*算法和 A 搜索算法的不同之处在于：A*保证对于所有节点都有：$h(n) \leqslant h^*(n)$，其中 $h^*(n)$为当前状态到目的状态的最优路径的代价。

## 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.A*算法

本实验总体实现思路采用 A*算法实现，对于 8 数码问题，可以将其理解为：一个时刻八个数码放置的状态为一个 node，初始时刻八个数码的状态为起点，目标状态为终点（如图 1）。如此就可以将 8 数码问题转换成一个最短路径问题，方便采用 A*算法求解。

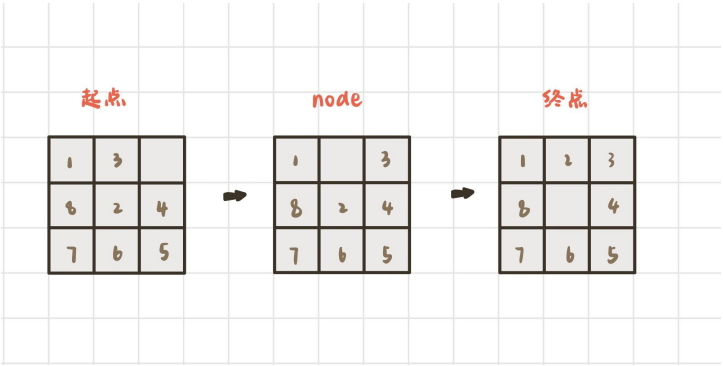


图 1

采用 A*算法的关键是要构造出估价函数，即 **f(n) = h(n) + d(n)**，其中 d(n)是已经付出的搜索代价，h(n)是对于当前状态到目标状态的估计值。对于这个问题 d(n)很好理解，就是从**初始状态到当前 node 所需要的步数**；而 h(n)我采用了曼哈顿距离来进行构造（如图 2），即对当前 node，**每个数码的当前位置与目标状态位置的曼哈顿距离之和作为 h(n)**，这一定满足 h(n)≤h*(n)。

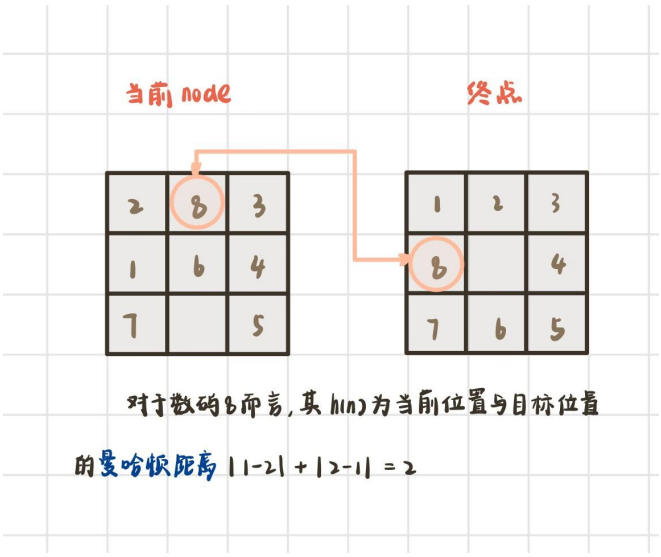


图 2

## 2.算法设计

**A*算法设计**

**Input：初始状态，目标状态；**

**Output：移动方案。**

1. 把起点加入 open list；
2. 遍历 open list，查找 **F 值最小的节点**，把它作为当前要处理的节点，并把这个节点移到 close list；
3. 对当前方格的 **4 个相邻方格**的每一个方格：
   - ✓ 如果它是**不可抵达**的或者它**在 close list 中**，则忽略它；
   - ✓ 否则，做如下操作：

   如果它不在 open list 中，把它加入 open list，并且把**当前方格设置为它的父亲**，记录该方格的 F，G 和 H 值；

   如果它已经在 open list 中，检查这条路径（即经由当前方格到达它那里）是否更好。如果是这样，把它的父亲设置为当前方格，并重新计算它的 G 和 F 值。
4. 当你把终点加入到了 open list 中（此时**路径已经找到**）或者 open list 是空的（**没有路径可选**），则进行下一步，否则**返回第（2）步重复过程**；
5. 保存路径，从终点开始沿着父节点**回溯直至起点**，得到移动方案。

## 3.具体实现

本次实现使用 python 语言进行编写，这里将对部分重点代码进行说明，全部代码见附件。

**（1）node 类的实现：**

```
# 定义节点类
class Node:
    G = 0  # g函数，已经消耗的步数
    H = 0  # h函数，预计到达目标还需要的步数
    F = 0  # f = g + h，总步数
    state = np.zeros((3, 3), dtype=int)
    parent = []  # 到达该状态的前一个状态

    # state是当前状态，prt是该状态的前一个状态
    def __init__(self, state, prt=[]):
        self.state = state
        if prt:
            self.parent = prt
            self.G = prt.G + 1
        for i in range(len(state)):
            for j in range(len(state[i])):
                x, y = self.find_pos(target_state, state[i][j])
                # 计算曼哈顿距离
                self.H = self.H + abs(x - i) + abs(y - j)
        self.F = self.G * 1 + self.H * 10

    # 找num在state状态中的位置x,y
    def find_pos(self, state, num):
        for i in range(len(state)):
            for j in range(len(state[i])):
                if state[i][j] == num:
                    return i, j

    # 移动该状态
    def moveto(self, x, y):
        x0, y0 = self.find_pos(self.state, 0)
        newstate = (self.state).copy()
        newstate[x0][y0] = newstate[x][y]
        newstate[x][y] = 0
        return newstate
```

node 类是用于表示 node 节点的类，它包含 **G**（已经消耗的步数，对应 d(n)）、**H**（预计到达终点还需要的步数，对应 h(n)）、**F**（总步数，对应 f(n)）、**state**（状态矩阵，存储 8 个数码的位置）以及 **parent**（该状态前一个状态）五个类成员变量。同时在构造方法中还有计算 F、G、H 的功能，以及两个类方法，一个

用于根据数码找到其当前位置，另一个用于移动数码。

**（2）逆序数的计算：**

```
# 得到逆序数，用于判断解的存在性
def get_reverse_num(state):
    ans = 0
    s = ""
    for i in range(len(state)):
        for j in range(len(state[i])):
            # 0即空格，不在考虑范围内
            if not state[i][j] == 0:
                s += str(state[i][j])

    for i in range(len(s)):
        for j in range(i):
            if s[j] > s[i]:
                ans += 1
    return ans
```

对于串中相邻的一对数，如果位置较前的数大于位置靠后的数，则成为一对逆序，一个串的逆序总数成为**逆序数**。在 N 数码中，并不是所有状态都是可解的：将一个状态表示为一维数组的形式，计算除了空位置（0）之外的所有数字的逆序数之和，如果初始状态和结束状态的逆序数的奇偶性相同，则相互可达，否则相互不可达。因此这里的函数用于计算某个 8 数码的逆序数。

**（3）回溯得到输出方案：**

```
# 输出状态及深度
def display(cur_node):
    alist = []
    tmp = cur_node
    while tmp:
        alist.append(tmp)
        tmp = tmp.parent
    alist.reverse()
    for node in alist:
        step = node.G
        if step == 0:
            print("原图：")
        elif step == 1:
            print("移动过程：")
            print()
            print("Step %d：" % node.G)
        else:
            print("Step %d：" % node.G)
        mat = node.state
        for i in range(len(mat)):
            for j in range(len(mat[i])):
                if mat[i][j] == 0:
                    print(" ", end=" ")
                else:
                    print(mat[i][j], end=" ")
            print()
        print()
    print("移动结束！")
```

最终能到达目标状态后，需要输出移动方案，而这需要从目标状态开始借助 node 类中的 parent 变量进行回溯来得到之前经过的所有状态，从而得到移动方案。

**（4）主程序：**

```
if __name__ == "__main__":
    # 初始状态和目标状态
    start_state = np.array([[1, 3, 0],
                            [8, 2, 4],
                            [7, 6, 5]])
    target_state = np.array([[1, 2, 3],
                             [8, 0, 4],
                             [7, 6, 5]])

    # 可行解判断
    if get_reverse_num(target_state) % 2 != get_reverse_num(start_state) % 2:
        print("找不到可行解！")
        exit(-1)

    # 可行解存在时，开始启发搜索
    open_list.append(Node(start_state))
    while open_list:
        current_node = open_list.pop(0)
        close_list.append(current_node)

        # 当open表中取出的恰好为目标状态时
        if (current_node.state == target_state).all():
            print("可行解已找到！")
            display(current_node)
            exit(0)

        # 否则对当前节点进行拓展
        x, y = current_node.find_pos(current_node.state, 0)
        for [x_, y_] in [[x + 1, y], [x - 1, y], [x, y + 1], [x, y - 1]]:
            if 0 <= x_ < len(start_state) and 0 <= y_ < len(start_state):
                new_state = current_node.moveto(x_, y_)
                # 判断新状态是否在close表
                if is_in_list(close_list, new_state) == -1:
                    # 如果不在close表
                    if is_in_list(open_list, new_state) == -1:
                        # 如果也不在open表
                        open_list.append(Node(new_state, current_node))
                    else:
                        # 如果open表中已存在这种状态，则选取G值较小的
                        sta = is_in_list(open_list, new_state)
                        if current_node.G + 1 < sta.G:
                            # 如果新路线更好，则放弃旧路线而选择新路线
                            open_list.remove(sta)
                            open_list.append(Node(new_state, current_node))

        # 对open表按F值从小到大进行排序
        open_list.sort(key=attrgetter("F"))
```

主程序主要利用前面总结的算法进行设计，首先根据逆序数奇偶性判断是否存在可行解，如果存在可行解就开始启发式搜索。在搜索过程中，若从 open_list 中取出的正好是目标节点，则表明已经到达目标状态，此时调用 display 函数展示移动方案。取新节点进行 open 表拓展时，应该先判断有没有越边界，如果没有，才能进行下一步拓展。

## 九、实验数据及结果分析：

这里仅做两组测试用例来对实验分析。

**（1）测试用例 1：**

8 数码初始状态为：

| | | |
|---|---|---|
| 1 | 3 | |
| 8 | 2 | 4 |
| 7 | 6 | 5 |

输出结果为：

```
可行解已找到！
原图：
1 3
8 2 4
7 6 5

移动过程：

Step 1：
1   3
8 2 4
7 6 5

Step 2：
1 2 3
8   4
7 6 5

移动结束！

Process finished with exit code 0
```

满足实验要求，并能正确输出移动步骤。

**（2）测试用例 2：**

8 数码初始状态为：

| | | |
|---|---|---|
| 2 | 8 | 3 |
| 1 | 6 | 4 |
| 7 | | 5 |

输出结果为：

```
可行解已找到！
原图：
2 8 3
1 6 4
7   5

移动过程：

Step 1：
2 8 3
1   4
7 6 5

Step 2：
2   3
1 8 4
7 6 5

Step 3：
  2 3
1 8 4
7 6 5

Step 4：
1 2 3
  8 4
7 6 5

Step 5：
1 2 3
8   4
7 6 5

移动结束！

Process finished with exit code 0
```

满足实验要求，并能正确输出移动方案。

## 十、实验结论：

本次实验圆满完成，成功实现了所有实验要求，能正确地按规格输出结果。

## 十一、总结及心得体会：

本次实验我熟悉和掌握了启发式搜索的定义、估价函数和算法过程，并利用A*算法求解了 N 数码难题，使我对 A*算法有了更深入的理解和思考，对该算法的应用有了一定的了解。

## 十二、对本实验过程及方法、手段的改进建议：

在实际编写代码时，可以先根据算法设计写出伪代码，这样在实现时能更清晰且更不容易出错。

## 十三、代码附件

**实验一、A*算法求解八数码问题**

```
from operator import attrgetter

import numpy as np

# 定义open表与close表
open_list = []
close_list = []
start_state = np.zeros((3, 3), dtype=int)  # 开始状态
target_state = np.zeros((3, 3), dtype=int)  # 目标状态


# 定义节点类
class Node:
    G = 0  # g函数，已经消耗的步数
    H = 0  # h函数，预计到达目标还需要的步数
    F = 0  # f = g + h，总步数
    state = np.zeros((3, 3), dtype=int)
    parent = []  # 到达该状态的前一个状态

    # state是当前状态，prt是该状态的前一个状态
```

```
21.         def __init__(self, state, prt=[]):
22.             self.state = state
23.             if prt:
24.                 self.parent = prt
25.                 self.G = prt.G + 1
26.             for i in range(len(state)):
27.                 for j in range(len(state[i])):
28.                     x, y = self.find_pos(target_state, state[i][j])
29.                     # 计算曼哈顿距离
30.                     self.H = self.H + abs(x - i) + abs(y - j)
31.             self.F = self.G * 1 + self.H * 10
32.
33.         # 找num在state状态中的位置x,y
34.         def find_pos(self, state, num):
35.             for i in range(len(state)):
36.                 for j in range(len(state[i])):
37.                     if state[i][j] == num:
38.                         return i, j
39.
40.         # 移动该状态
41.         def moveto(self, x, y):
42.             x0, y0 = self.find_pos(self.state, 0)
43.             newstate = (self.state).copy()
44.             newstate[x0][y0] = newstate[x][y]
45.             newstate[x][y] = 0
46.             return newstate
47.
48.
49.     # 得到逆序数，用于判断解的存在性
50.     def get_reverse_num(state):
51.         ans = 0
52.         s = ""
53.         for i in range(len(state)):
54.             for j in range(len(state[i])):
55.                 # 0即空格，不在考虑范围内
56.                 if not state[i][j] == 0:
57.                     s += str(state[i][j])
58.
59.         for i in range(len(s)):
60.             for j in range(i):
61.                 if s[j] > s[i]:
62.                     ans += 1
63.         return ans
64.
```

```
65.
66.     # 输出状态及深度
67.     def display(cur_node):
68.         alist = []
69.         tmp = cur_node
70.         while tmp:
71.             alist.append(tmp)
72.             tmp = tmp.parent
73.         alist.reverse()
74.         for node in alist:
75.             step = node.G
76.             if step == 0:
77.                 print("原图：")
78.             elif step == 1:
79.                 print("移动过程：")
80.                 print()
81.                 print("Step %d: " % node.G)
82.             else:
83.                 print("Step %d: " % node.G)
84.             mat = node.state
85.             for i in range(len(mat)):
86.                 for j in range(len(mat[i])):
87.                     if mat[i][j] == 0:
88.                         print(" ", end=" ")
89.                     else:
90.                         print(mat[i][j], end=" ")
91.                 print()
92.             print()
93.         print("移动结束！")
94.
95.
96.     # 检查state状态是否在list中（可能是open或close表）
97.     def is_in_list(alist, state):
98.         for stat in alist:
99.             if (stat.state == state).all():
100.                return stat
101.        return -1
102.
103.
104.    if __name__ == "__main__":
105.        # 初始状态和目标状态
106.        start_state = np.array([[2, 8, 3],
107.                                [1, 6, 4],
108.                                [7, 0, 5]])
```

```
    target_state = np.array([[1, 2, 3],
                             [8, 0, 4],
                             [7, 6, 5]])

    # 可行解判断
    if get_reverse_num(target_state) % 2 != get_reverse_num(start_state) % 2:
        print("找不到可行解！")
        exit(-1)

    # 可行解存在时，开始启发搜索
    open_list.append(Node(start_state))
    while open_list:
        current_node = open_list.pop(0)
        close_list.append(current_node)

        # 当 open 表中取出的恰好为目标状态时
        if (current_node.state == target_state).all():
            print("可行解已找到！")
            display(current_node)
            exit(0)

        # 否则对当前节点进行拓展
        x, y = current_node.find_pos(current_node.state, 0)
        for [x_, y_] in [[x + 1, y], [x - 1, y], [x, y + 1], [x, y - 1]]:
            if 0 <= x_ < len(start_state) and 0 <= y_ < len(start_state):
                new_state = current_node.moveto(x_, y_)
                # 判断新状态是否在 close 表
                if is_in_list(close_list, new_state) == -1:
                    # 如果不在 close 表
                    if is_in_list(open_list, new_state) == -1:
                        # 如果也不在 open 表
                        open_list.append(Node(new_state, current_node))
                    else:
                        # 如果 open 表中已存在这种状态，则选取 G 值较小的
                        sta = is_in_list(open_list, new_state)
                        if current_node.G + 1 < sta.G:
                            # 如果新路线更好，则放弃旧路线而选择新路线
                            open_list.remove(sta)
                            open_list.append(Node(new_state, current_node))

        # 对 open 表按 F 值从小到大进行排序
        open_list.sort(key=attrgetter("F"))
```

报告评分：

指导教师签字：