2024 年秋季《人工智能导论》

实验二

A*算法求解迷宫寻路问题实验

实验报告

学院： 自动化学院

专业： 智能科学与技术

学号： _____

姓名： _____

2024 年 10 月

一、实验目的

熟悉和掌握 A*算法实现迷宫寻路功能，要求掌握启发式函数的编写以及各类启发式函数效果的比较。

二、实验内容

寻路问题常见于各类游戏中角色寻路、三维虚拟场景中运动目标的路径规划、机器人寻路等多个应用领域。迷宫寻路问题是在以方格表示的地图场景中,对于给定的起点、终点和障碍物(墙),如何找到一条从起点开始避开障碍物到达终点的最短路径。

假设在一个 n*m 的迷宫里,人口坐标和出口坐标分别为(1,1)和(5,5),每一个坐标点有两种可能:0 或 1,其中 **0 表示该位置允许通过,1 表示该位置不允许通过**。

如地图：

0	0	0	0	0
1	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	0	0	1	0

最短路径应该是：

A	B	0	0	0
1	C	1	0	1
E	D	1	1	1
F	1	J	K	L
G	H	I	1	M

即:(1,1)-(1,2)-(2,2)-(3,2)-(3,1)-(4,1)-(5,1)-(5,2)-(5,3)-(4,3)-(4,4)-(4,5)-(5,5)

以寻路问题为例实现 A*算法的求程序（编程语言不限）,要求设计两种不同的估价函数。

三、实验原理

A*算法是一种结合启发式搜索和代价估算的路径规划算法。通过维护两个列表（开放列表和关闭列表），逐步探索最优路径。

1. 基本概念

（1）$f(n)=g(n)+h(n)$

$g(n)$：从起点到当前节点的实际代价。

$h(n)$：从当前节点到终点的估计代价（启发式函数）。

（2）关闭列表 Close：存储已扩展结点，即所有寻找的节点。

（3）开放列表 Opens：存储已生成节点，即最终走过的节点。

2. 算法流程

（1）将起点加入开放列表。

（2）从开放列表中选取f值最小的节点作为当前节点。

（3）扩展当前节点的所有可到达的邻居节点。

（4）计算邻居节点的$f(n)$值，更新开放列表和关闭列表。

（5）如果目标节点进入关闭列表，算法结束，返回路径。

（6）若开放列表为空，表示无解。

3. 启发式函数

曼哈顿距离：适合网格场景，计算横纵距离之和。

$$h(n) = |x_1 - x_2| + |y_1 - y_2|$$

欧几里得距离：适合真实距离测算。

$$h(n) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

四、实验算法

1. 定义估价函数

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

其中 $g(n)$：起点到n状态的最短路径代价的估计值。

$h(n)$：n状态到目的状态的最短路径代价的估计值。

令$g(n)$为起点到n状态的**曼哈顿距离**，代码如下：

```
def gs(i, j):
    return abs(i - startx) + abs(j - starty)
```

2. 定义两种启发式函数：

$$h_1(n) = 10 \times \left(|n_x - endx| + |n_y - endy|\right)$$

$$h_2(n) = (n_x - endx)^2 + \left(n_y - endy\right)^2$$

代码如下：

```
def h1(i, j):
    return 10*(abs(i - endx) + abs(j - endy))
def h2(i, j):
    return pow(i - endx, 2) + pow(j - endy, 2)
```

五、实验分析与结果

（一）程序代码 [详见“Pathfinding.py”，以下仅展示关键代码。]

（1）主分支（判断起点是否“围在墙里”）

```
if a[startx - 1, starty - 1] == 1:
    print("起点%s 位置为墙，最短路径寻找失败！" % ([startx, starty]))
else: (详见核心程序)
```

（2）核心程序（实现迷宫寻路）：

```
Close = [[startx, starty]] # 存储已扩展结点，即所有寻找的节点
Opens = [[startx, starty]] # 存储已生成节点，即最终走过的节点
crossings = [] # 存储未走的分叉节点
road = 2 # 设置行走的路径值，初值为 2（便于与 0/1 区分）
start = time.time()
rows, cols = a.shape
while True:
    if Close[-1] != [endx, endy]:
        Open = []
        # 减 1 得到数组下标值
        i, j = Close[-1][0] - 1, Close[-1][1] - 1
        # 对当前节点上下左右四个节点进行判断
        for ni, nj in [(i - 1, j), (i + 1, j), (i, j - 1), (i, j + 1)]:
            if [ni + 1, nj + 1] not in Opens and 0 <= ni < rows and 0 <= nj < cols and a[ni, nj] == 0:
                Open.append([ni + 1, nj + 1])
        # 将已走过的节点值修改为路径值，并将路径值加 1
        a[i, j] = road
        road = road + 1
        if Open:
            # 对所有扩展到的节点排序，reverse=True 从大到小，尾部存储代价最小的节点
            Open = sorted(Open, key=lambda x: gs(x[0], x[1]) + h2(x[0], x[1]), reverse=True)
            Opens.extend(Open)
            # 将代价最小的节点存储到 Close 表
            Close.append(Open.pop())
            # 如果 pop 后 Open 表不为空，说明存在岔路，将岔路存储到 crossings 中
            if Open:
                crossings.extend(Open)
        # 判断是否存在未走过的分叉节点
        elif crossings:
            next_way = crossings.pop()
            # 实现路径回退，循环条件为回退节点与分叉节点不相邻
            while sum((np.array(Close[-1]) - np.array(next_way)) ** 2) != 1:
                # 当一条路径寻找失败后，是否将该路径岔路后的部分恢复为原地图
                # i, j = Close[-1]
                # a[i-1, j-1] = 0
                # 每次 while 循环路径值 road 减 1，并弹出 Close 表中的节点
                road -= 1
                Close.pop()
            # 将 crossings 中最后一个节点添加到 Close 表
            Close.append(next_way)
        else:
            print("最短路径寻找失败，失败位置为：%s，路径为：" % Close[-1])
            break
    else:
        a[endx-1, endy-1] = road
        print("最短路径寻找成功，路径为：")
        break
for r in range(rows):
    for c in range(cols):
        # 0 表示 format 中第一个元素，>表示右对齐输出，4 表示占四个字符
        print("{0: >4}".format(a[r, c]), end='')
    print('')
end = time.time()
print("\n 扩展节点数为：%d, 生成节点数为：%d, 用时为 %.8f" % (len(Opens), len(Close), float(end - start)))
print('Close 表为：%s' % Close)
print("点的移动轨迹为：")
for k in range(len(Close)):
    print(Close[k], end='')
    if k < len(Close) - 1:
        print("-->", end='')
```

（二）测试样例

【示例一】对于如图矩阵 a，先设定起点、终点位置：

startx, starty = 1,1

endx, endy = 20,20

以下是矩阵 a 的赋值：

```
a = np.asmatrix([[0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1],
                 [0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1],
                 [0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1],
                 [0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1],
                 [1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1],
                 [0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1],
                 [0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1],
                 [1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1],
                 [0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1],
                 [0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0],
                 [0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1],
                 [0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1],
                 [0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0],
                 [0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1],
                 [1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
                 [0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
                 [0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
                 [0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
                 [1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0],
                 [0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0]])
```

输出结果如下：

```
最短路径寻找成功，路径为：
2   1   0   1   0   1   1   0   1   1   0   0   1   1   1   0   1   1   1   1
3   1   0   0   0   0   1   0   0   0   1   0   1   0   1   0   0   0   1   1
4   1   8   9   10  0   0   1   1   1   1   0   0   0   0   1   1   0   1   1
5   6   7   1   11  0   1   0   0   0   1   0   1   0   0   0   1   0   1   1
1   1   1   1   12  1   0   0   1   1   1   0   0   1   1   1   0   0   1   1
0   0   1   0   13  14  15  0   1   0   1   1   1   1   0   1   1   0   0   1
0   1   0   1   0   0   16  17  1   0   1   0   0   1   0   0   1   0   1   1
1   0   0   0   0   1   0   18  19  1   1   0   0   1   1   1   1   0   0   1
0   1   1   0   0   0   0   0   20  21  22  1   26  27  28  1   1   1   0   1
0   1   0   0   1   1   1   1   1   1   23  24  25  1   29  1   1   0   1   0
0   0   1   0   0   1   1   1   1   1   1   1   1   1   30  1   0   0   0   1
0   1   1   0   1   0   43  42  41  38  37  36  35  34  31  1   0   1   1   1
0   1   1   1   0   1   44  1   40  39  1   1   1   33  32  1   0   0   0   0
0   0   0   0   0   1   45  1   1   1   0   1   1   1   1   1   0   0   0   1
1   1   0   1   1   1   46  1   0   0   0   0   0   0   0   0   0   0   0   1
0   0   1   50  49  48  47  1   0   0   0   0   0   0   0   0   0   0   0   1
0   1   1   51  1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   0
0   1   0   52  1   1   0   1   59  60  61  62  63  64  65  66  67  68  69  70
1   1   0   53  54  55  56  57  58  1   0   0   1   0   0   0   0   1   1   71
0   0   1   1   1   1   0   0   1   1   1   1   1   0   0   1   1   0   1   72
```

扩展节点数为：97，生成节点数为：71，用时为 0.00147247

Close 表为：[[1, 1], [2, 1], [3, 1], [4, 1], [4, 2], [4, 3], [3, 3], [3, 4], [3, 5], [4, 5], [5, 5], [6, 5], [6, 6], [6, 7], [7, 7], [7, 8], [8, 8], [8, 9], [9, 9], [9, 10], [9, 11], [10, 11], [10, 12], [10, 13], [9, 13], [9, 14], [9, 15], [10, 15], [11, 15], [12, 15], [13, 15], [13, 14], [12, 14], [12, 13], [12, 12], [12, 11], [12, 10], [13, 10], [13, 9], [12, 9], [12, 8], [12, 7], [13, 7], [14, 7], [15, 7], [16, 7], [16, 6], [16, 5], [16, 4], [17, 4], [18, 4], [19, 4], [19, 5], [19, 6], [19, 7], [19, 8], [19, 9], [18, 9], [18, 10], [18, 11], [18, 12], [18, 13], [18, 14], [18, 15], [18, 16], [18, 17], [18, 18], [18, 19], [18, 20], [19,

20], [20, 20]]

点的移动轨迹为：

[1, 1]-->[2, 1]-->[3, 1]-->[4, 1]-->[4, 2]-->[4, 3]-->[3, 3]-->[3, 4]-->[3, 5]-->[4, 5]-->[5, 5]-->[6, 5]-->[6, 6]-->[6, 7]-->[7, 7]-->[7, 8]-->[8, 8]-->[8, 9]-->[9, 9]-->[9, 10]-->[9, 11]-->[10, 11]-->[10, 12]-->[10, 13]-->[9, 13]-->[9, 14]-->[9, 15]-->[10, 15]-->[11, 15]-->[12, 15]-->[13, 15]-->[13, 14]-->[12, 14]-->[12, 13]-->[12, 12]-->[12, 11]-->[12, 10]-->[13, 10]-->[13, 9]-->[12, 9]-->[12, 8]-->[12, 7]-->[13, 7]-->[14, 7]-->[15, 7]-->[16, 7]-->[16, 6]-->[16, 5]-->[16, 4]-->[17, 4]-->[18, 4]-->[19, 4]-->[19, 5]-->[19, 6]-->[19, 7]-->[19, 8]-->[19, 9]-->[18, 9]-->[18, 10]-->[18, 11]-->[18, 12]-->[18, 13]-->[18, 14]-->[18, 15]-->[18, 16]-->[18, 17]-->[18, 18]-->[18, 19]-->[18, 20]-->[19, 20]-->[20, 20]

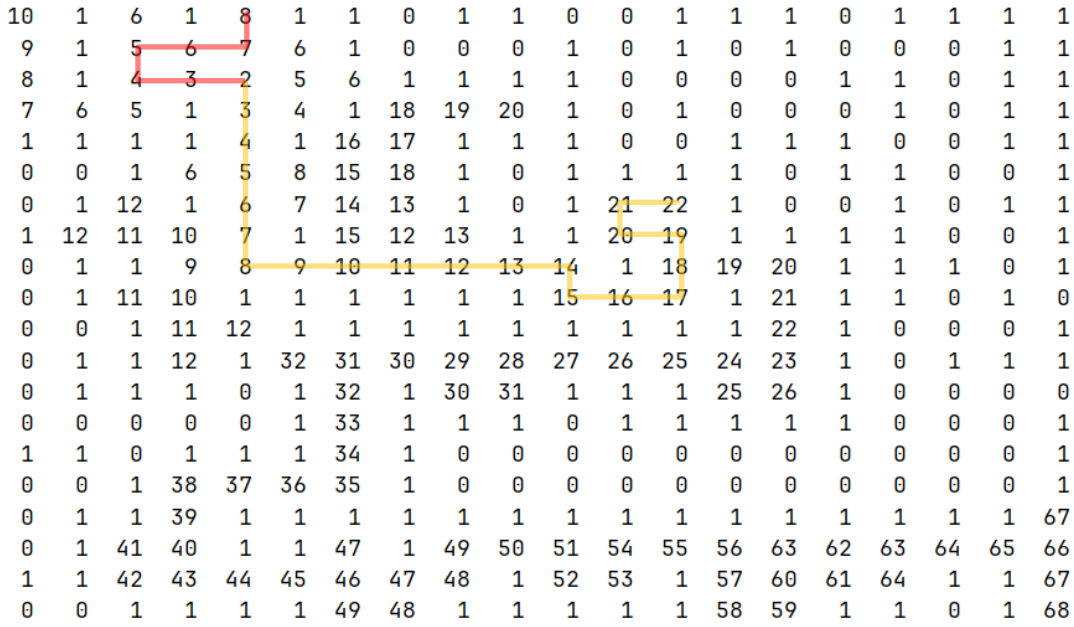
【示例二】仍然对于以上矩阵 a，设定起点、终点位置：

startx, starty = 3,5

endx, endy = 13,5

输出结果如下：

最短路径寻找失败，失败位置为：[1, 5]，路径为：



扩展节点数为：124, 生成节点数为：7, 用时为 0.00192022 ……（Close 表和移动轨迹省略）

（三）实验总结

A*算法在地图寻路问题中具有很好的优势，相比宽度优先搜索，效率更高，所耗时间更少，相比深度优先搜索，可以解决其不能找到最优解的不足，具有较高的应用价值。该算法的重点及难点就是启发式函数的选择，通过上述实验，可以发现启发式函数 h_2 相比 h_1 效果更好，但仍存在一些问题，需要继续找寻更优的启发式函数。