成 绩

北京航空航天大學
BEIHANG UNIVERSITY

A*算法机器人路径规划问题

实验报告

院（系）名称	自动化科学与电气工程学院
专 业 名 称	自动化
学 生 学 号	16711094
学 生 姓 名	李翰韬

2020 年 6 月

一、实验目的

1、使学生加深对图搜索技术的理解

2、掌握图搜索基本编程方法

3、运用图搜索技术解决一些应用问题

二、实验要求

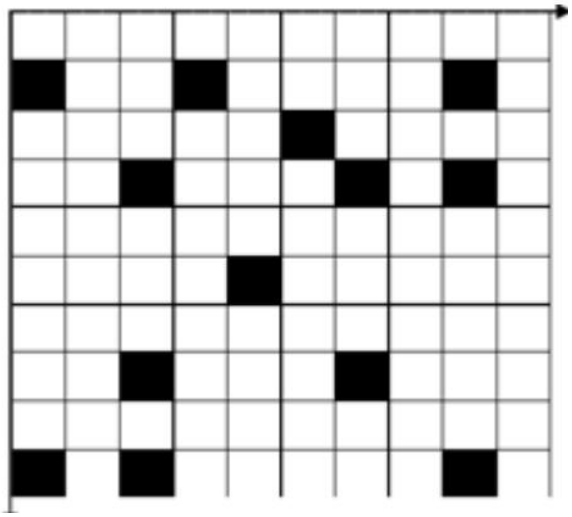
1、用启发式搜索算法实现路径规划问题。

2、有明确的状态空间表达，规则集以及估计函数。

3、程序运行时，应能清晰直观演示搜索过程。

三、实验内容

机器人路径规划问题:左上角为坐标原点,水平向右为 x 轴方向,竖直向下 为 y 轴方向。白色为自由栅格，黑色为障碍栅格，机器人只能在自由栅格中运动， 并躲避障碍。每个栅格由唯一的坐标（x,y）表示。机器人一般有八个可移动方向。给出由初始位置（3，3）到目标位置（9，9）的最佳路线。



四、实验步骤

1. 设计问题的状态表示方法

在本次(10*10)地图的机器人寻路中，设计的状态表示方法即为机器人所在格子的坐标。在实验要求的地图中，S_0=(3, 3)；S_g=(9, 9)

对于 S_i=(x, y)，状态表示规则为：

$$r_1(x \neq 1) \wedge (map[x-1][y] \neq block) \rightarrow x \Leftarrow x-1$$
$$r_2(x \neq 10) \wedge (map[x+1][y] \neq block) \rightarrow x \Leftarrow x+1$$
$$r_3(y \neq 1) \wedge (map[x][y-1] \neq block) \rightarrow y \Leftarrow y-1$$
$$r_4(y \neq 10) \wedge (map[x][y+1] \neq block) \rightarrow y \Leftarrow y+1$$
$$r_5(x \neq 1) \wedge (y \neq 1) \wedge (map[x-1][y-1] \neq block) \rightarrow x \Leftarrow x-1 \wedge y \Leftarrow y-1$$
$$r_6(x \neq 1) \wedge (y \neq 10) \wedge (map[x-1][y+1] \neq block) \rightarrow x \Leftarrow x-1 \wedge y \Leftarrow y+1$$
$$r_7(x \neq 10) \wedge (y \neq 1) \wedge (map[x+1][y-1] \neq block) \rightarrow x \Leftarrow x+1 \wedge y \Leftarrow y-1$$
$$r_8(x \neq 10) \wedge (y \neq 10) \wedge (map[x+1][y+1] \neq block) \rightarrow x \Leftarrow x+1 \wedge y \Leftarrow y+1$$

八项转移规则分别表示机器人向[上、下、左、右、左上、右上、左下、右下]八个方向运动。map 为迷宫地图，block 表示此点为墙壁。

2. 定义启发式函数

设计启发式函数，*i* 为搜索得到的单元格。

*g(i)*为机器人从起点坐标前往 *i* 每行动一步所耗费的距离代价和。

$$g^* = g = \sum_{j=0}^{i} \sqrt{\left(x_{j+1} - x_j\right)^2 + \left(y_{j+1} - y_j\right)^2}$$

h(i)*为 *i* 与终点坐标的欧氏距离。

$$h^* = \sqrt{\left(x_{des} - x_j\right)^2 + \left(y_{des} - y_i\right)^2}$$

可以明显发现，在最理想情况下，即 *i* 与终点横坐标差等于纵坐标差，且路径上没有障碍物，此时 *h(n)=h*(n)*。其余情况下，*h(n)<h*(n)*，说明符合 A*算法条件。

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

3. 实现搜索过程

首先编写函数描述机器人的运动

```
# 机器人运动的八种方式
orders = ['u', 'd', 'l', 'r','ul','ur','dl','dr']

# 搜索机器人当前位置能够做出的运动
def valid_actions(loc):
    loc_actions = []
    for order in orders:
        if is_move_valid(loc, order):
            loc_actions.append(order)
    return loc_actions

# 判断能够进行某个运动
def is_move_valid(loc, act):
    x = loc[0] - 1
    y = loc[1] - 1
    if act not in orders:
        return False
    else:
        if act == orders[0]:#u
            return x != 0 and env_data[x-1][y] != 2
        elif act == orders[1]:#d
            return x != len(env_data)-1 and env_data[x+1][y] != 2
        elif act == orders[2]:#l
            return y != 0 and env_data[x][y-1] != 2
        elif act == orders[3]:#r
            return y != len(env_data[0])-1 and env_data[x][y+1] != 2
        elif act == orders[4]:#ul
            return x != 0 and y != 0 and env_data[x-1][y-1] != 2
        elif act == orders[5]:#ur
            return x != 0 and y != len(env_data[0])-1 and env_data[x-1][y+1] != 2
        elif act == orders[6]:#dl
            return x != len(env_data)-1 and y != 0 and env_data[x+1][y-1] != 2
        else:                       #dr
            return x != len(env_data)-1 and y != len(env_data[0])-1 and env_data[x+1][y+1] != 2

# 搜索机器人当前位置能够前往的位置
def get_all_valid_loc(loc):
    all_valid_data = []
    cur_acts = valid_actions(loc)
    for act in cur_acts:
        all_valid_data.append(move_robot(loc, act))
    if loc in all_valid_data:
        all_valid_data.remove(loc)
    return all_valid_data

# 得到某个运动后坐标
def move_robot(loc, act):
    if is_move_valid(loc, act):
        if act == orders[0]:#u
            return loc[0] - 1, loc[1]
        elif act == orders[1]:#d
            return loc[0] + 1, loc[1]
        elif act == orders[2]:#l
            return loc[0], loc[1] - 1
        elif act == orders[3]:#r
            return loc[0], loc[1] + 1
        elif act == orders[4]:#ul
            return loc[0] - 1, loc[1] - 1
        elif act == orders[5]:#ur
            return loc[0] - 1, loc[1] + 1
        elif act == orders[6]:#dl
            return loc[0] + 1, loc[1] - 1
        else:                       #dr
            return loc[0] + 1, loc[1] + 1
    else:
        return loc
```

其中，['u', 'd', 'l', 'r','ul','ur','dl','dr']分别表示机器人向[上、下、左、右、左上、右上、左下、右下]八个方向运动。两个函数可以通过地图数组，判断机器人能否进行某个方向的运动。

值得注意的是，本程序编写时，机器人运动可以穿越斜向薄墙。若需要使机器人不能穿越斜向墙壁，只需简单更改 is_move_valid 函数即可，在此不做赘述。

编写函数描述启发式函数：

```
def compute_g(loc,cur_node):
    g = cur_node[3] + math.sqrt((loc[0]-cur_node[0][0])**2 + (loc[1]-cur_node[0][1])**2);
    return round(g,2)
```

```
def compute_h(loc):
    h = math.sqrt((loc[0]-des_loc[0])**2 + (loc[1]-des_loc[1])**2)
    return round(h,2)
```

A*算法搜索过程可按以下步骤说明：

(1) 把起始节点 S 放到 OPEN 表中，计算 f(S)，并把其值与节点 S 联系起来.

(2) 如果 OPEN 表是个空表，则失败退出，无解.

(3) 从 OPEN 表中选择一个 f 值最小的节点 i。 结果有几个节点合格，当其中有一个为目标节点时，则选择此目标节点，否则就选择其中任一个节点作为节点 i.

(4) 把节点 i 从 OPEN 表中移出，并把它放入 CLOSED 的扩展节点表中.

(5) 如果 i 是个目标节点，则成功退出，求得一个解.

(6) 扩展节点 i，生成其全部后继节点. 对于 i 的每一个后继节点 j:

a) 计算 f(j).

b) 如果 j 既不在 OPEN 表中，也不在 CLOSED 表中，则用估价函数 f 把它添入 OPEN 表. 从 j 加一指向父辈节点 i 的指针。

c) 如果 j 已在 OPEN 表或 CLOSED 表上，则比较刚刚对 j 计算过的 f 值和前面计算过的该节点在表中的 f 值. 如果新的 f 值较小，则

I. 以此新值取代旧值.

II. 从 j 指向 i，而不是指向它的父辈节点

III. 如果节点 j 在 CLOSED 表中，则把它移回 OPEN 表

(7) 转向(2),即 GOTO(2)；

可以用代码直接描述这些步骤，其中标示出的编号对应上述步骤中编号。

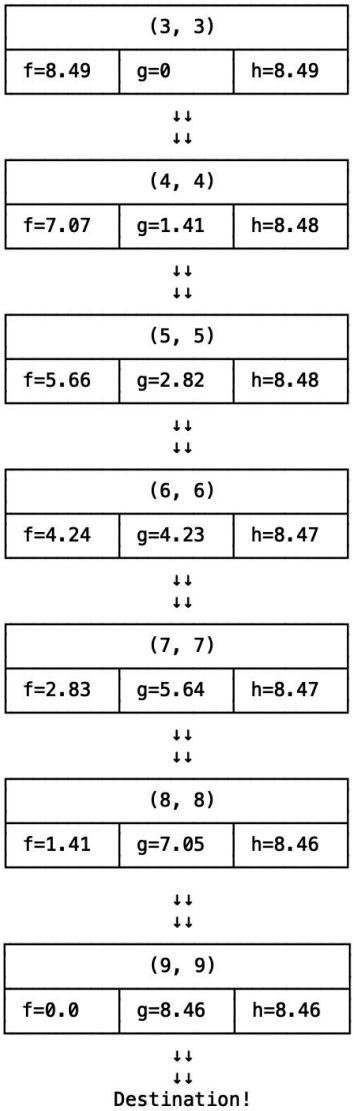
```
#  (1)
    Openlist = [[start_loc,start_loc,compute_f(start_loc),0,compute_f(start_loc)]]
    Closelist = []

    while True:
#  (2)
        if len(Openlist) == 0:
            print("失败，无解")
            break
#  (3)
        Openlist.sort(key=takeh)

#  (4)
        cur_node = Openlist[0]
        cur_node_loc = Openlist[0][0]
        Closelist.append(Openlist.pop(0))


#  (5)
        if cur_node_loc == des_loc:
            print("以上为搜索过程")
            print('-'*30)
            print('以下为最终路径单支树状图')
            Routelist.append(cur_node)
            while Routelist[-1][0] != start_loc:
                for close_node in Closelist:
                    if Routelist[-1][1] == close_node[0]:

Routelist.append(close_node)
            Routelist.reverse()
            break

#  (6)
        Curlist = []
        valid_loc_now= get_all_valid_loc(cur_node_loc)
        #(a)
        for i in range(0,len(valid_loc_now)):
            h = compute_h(valid_loc_now[i])
            g = compute_g(valid_loc_now[i],cur_node)
            f = round((h+g),2)

Curlist.append([valid_loc_now[i],cur_node_loc,h,g,f])



        Checklist = Openlist + Closelist
        Checklist = [x[0] for x in Checklist]

        for i_Cur in range(0,len(Curlist)):
            #(b)
            if Curlist[i_Cur][0] not in Checklist:

Openlist.append(Curlist[i_Cur])
            #(c)
            else:
                for i_Open in range(0,len(Openlist)):
                    if Curlist[i_Cur][0] == Openlist[i_Open][0]:
                        if Curlist[i_Cur][4] - Openlist[i_Open][4] < 0:

Openlist[i_Open] = Curlist[i_Cur]

                for i_Close in range(0,len(Closelist)):
                    if Curlist[i_Cur][0] == Closelist[i_Close][0]:
                        if Curlist[i_Cur][4] - Closelist[i_Close][4] < 0:

Closelist[i_Close] = Curlist[i_Cur]

Openlist.append(Closelist[i_Close])
                Closelist = [i for i in Closelist if i not in Openlist]
```

代码中具体函数功能在代码文件注释中描述。

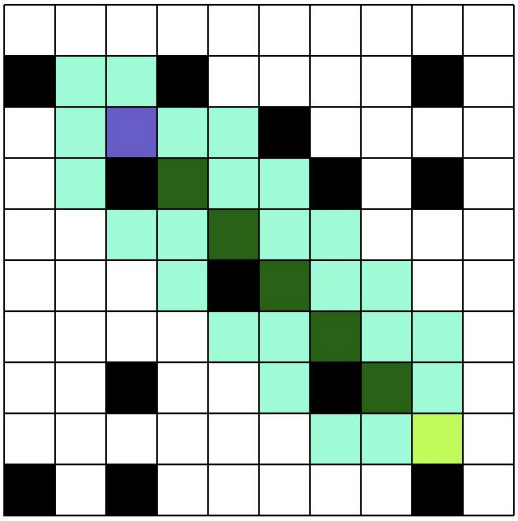
编写完成 A*算法具体代码后，将整个迷宫地图可视化。使用颜色为 Black 的方块表示迷宫障碍；使用颜色为 SlateBlue 的方块表示起点；使用颜色为 GreenYellow 的方块表示终点；使用颜色为 Aquamarine 的方块表示搜索过的格子；使用颜色为 DarkGreen 的方块表示当前的最佳路径。

程序可自动输出搜索过程以及最终路径节点的单支树状图：

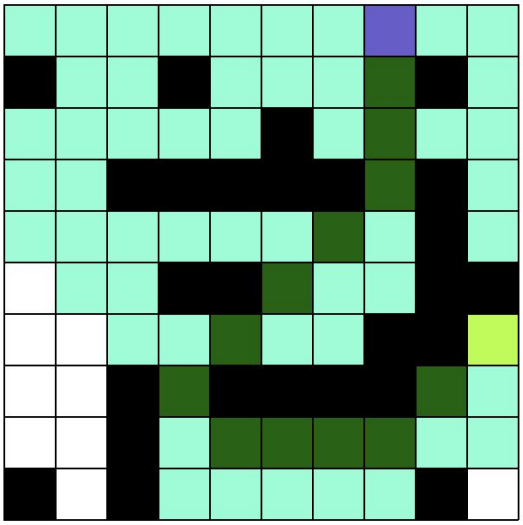
```
[(3, 3)]
[(3, 3), (4, 4)]
[(3, 3), (4, 4), (5, 5)]
[(3, 3), (4, 4), (5, 5), (6, 6)]
[(3, 3), (4, 4), (5, 5), (6, 6), (7, 7)]
[(3, 3), (4, 4), (5, 5), (6, 6), (7, 7), (8, 8)]
以上为搜索过程
```



每一步搜索延迟 0.5 秒，程序可以实现搜索的动态展示过程，由于文档中无法展示动态过程，报告中只展示最终图像：



由于题目所给迷宫过于简单，为验证程序准确性，实验另一个较为复杂的迷宫：



由此可知，本程序能够很好地进行机器人路径规划。

五、总结分析

A*算法是一个从概念上很好理解的算法，但是在编程中，却需要严格遵守 A*算法的步骤进行编程，才能得到正确的程序。如果在复现 A*算法时省略了某个步骤，比如没有将 Close 表中更新的节点重新载入 Open 表，在某些简单的例子中可能能够得到正确结果，但在复杂的例子中，就可能出现错误。因此，虽然 A*算法是人工智能中理解起来较为简单的算法，但是编程中仍需要谨慎。