**A\*算法求解迷宫寻路问题**

1. **实验目的**

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

1. **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，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，入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。

以寻路问题为例实现A\*算法的求解程序，要求设计两种不同的估价函数。

1. **实验环境**

实验环境为Windows10系统及开源程序语言python。

1. **实验要求**
2. 画出用A\*算法求解迷宫最短路径的流程图。
3. 设置不同的地图，以及不同的初始状态和目标状态，记录A\*算法的求解结果，包括最短路径、扩展节点数、生成节点数和算法运行时间。
4. 对于相同的初始状态和目标状态，设计不同的启发式函数，比较不同启发式函数对迷宫寻路速度的提升效果，包括扩展节点数、生成节点数和算法运行时间。
5. **实验过程**
6. **绘制流程图**

A\*算法求解迷宫最短路径的流程图如下图5.1。

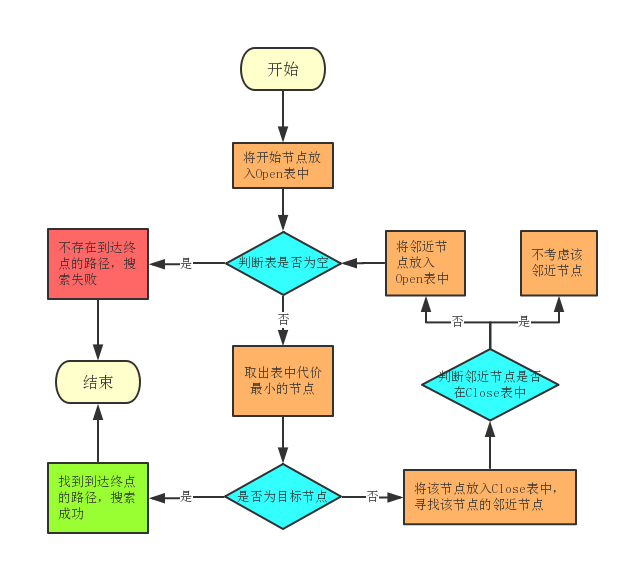


图 5.1

1. **设置相关数据**
2. **设置两种地图**

根据题意，用矩阵设置两个地图。

**地图1**：设置5行5列的迷宫，代码如下：

a = np.mat([[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]])

**地图2：**设置20行20列的迷宫，部分代码如下

a=np.mat([[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],

[………………………………………………… ],

[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]])

1. **设置两种启发式函数**

定义估价函数

式中，为起点到状态的最短路径代价的估计值，是状态到目的状态的最短路径代价的估计值。

令为起点到状态的曼哈顿距离，代码如下：

def gs(i, j):

return abs(i - startx) + abs(j - starty)

定义两种启发式函数。

**启发式函数：**，代码如下：

def h1(i, j):

return 10\*(abs(i - endx) + abs(j - endy))

**启发式函数**，代码如下：

def h2(i, j):

return pow(i - endx, 2) + pow(j - endy, 2)

1. **性能分析**
2. **分析不同起点终点**

采用地图2，具体如下图5.2，启发式函数进行分析，分别设置起点为（0，0）和（9，5），终点为（20，20）、（18，20）。

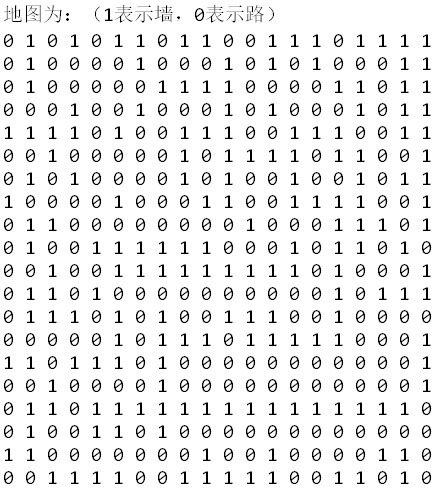


图 5.2

（0，0）🡪(20,20)的路径如图5.3，（9，5）🡪(18,20)的路径如图5.4。

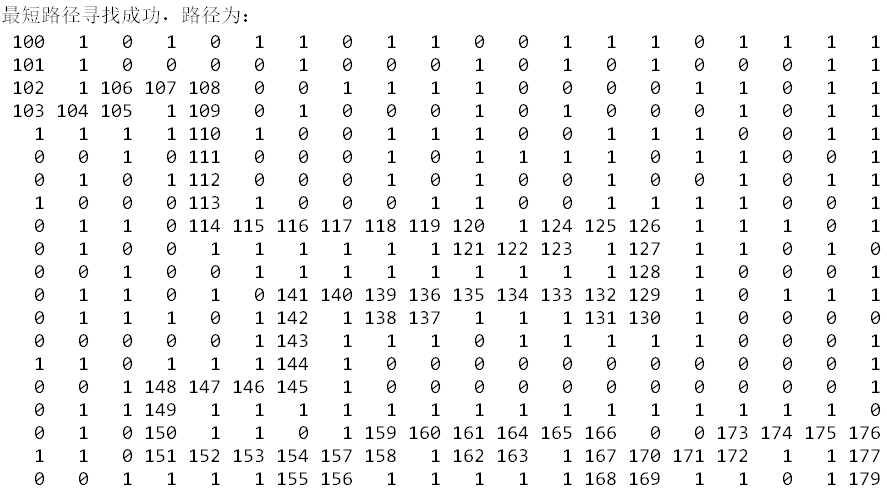


图 5.3

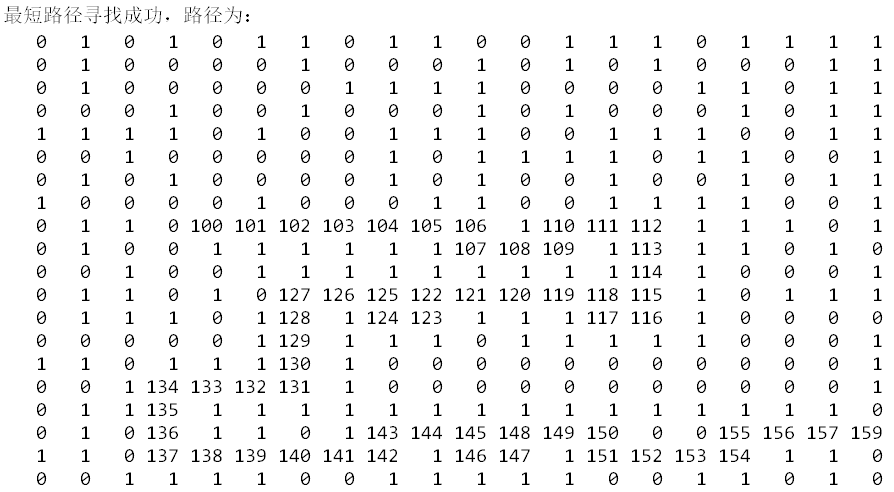


图 5.4

两次寻路过程的性能分析如下表5.1。

表 5.1

|  |  |  |
| --- | --- | --- |
| **路径** | （0，0）🡪(20,20) | （9，5）🡪(18,20) |
| **扩展节点数** | 107 | 80 |
| **生成节点数** | 79 | 59 |
| **运行时间** | 0.0016017052985262126 | 0.0008712525886949152 |

由上述图表可知路径延长后运行时间会相对变长，扩展节点及生成节点也会相应增多，且两次寻找的路程有重合的部分，所以寻找到的路径为较优路径。

1. **分析不同启发式函数**

采用地图2，启发式函数，进行分析，设置起点为（0，0），终点为（20，20），两次寻路过程的路径，扩展节点数、生成节点数和运行时间分别如下图5.5，5.6。

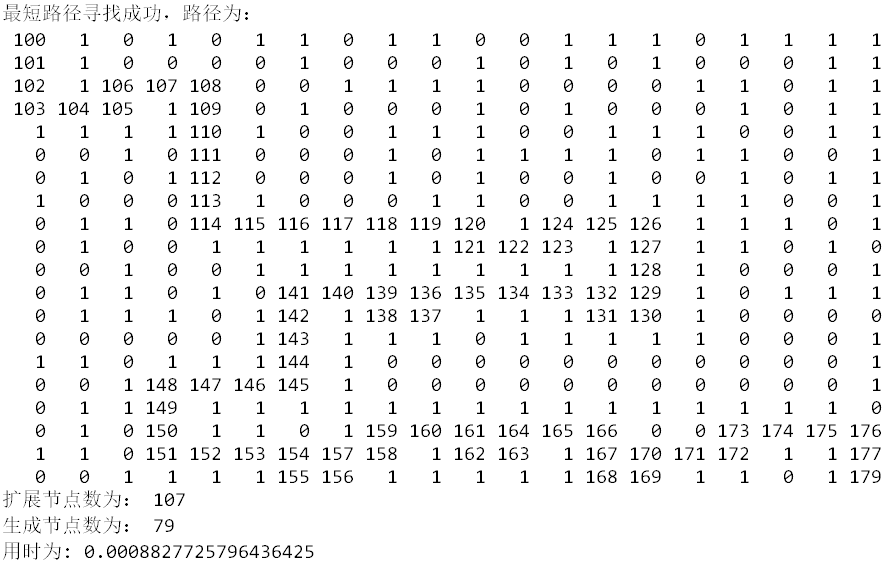


图 5.5

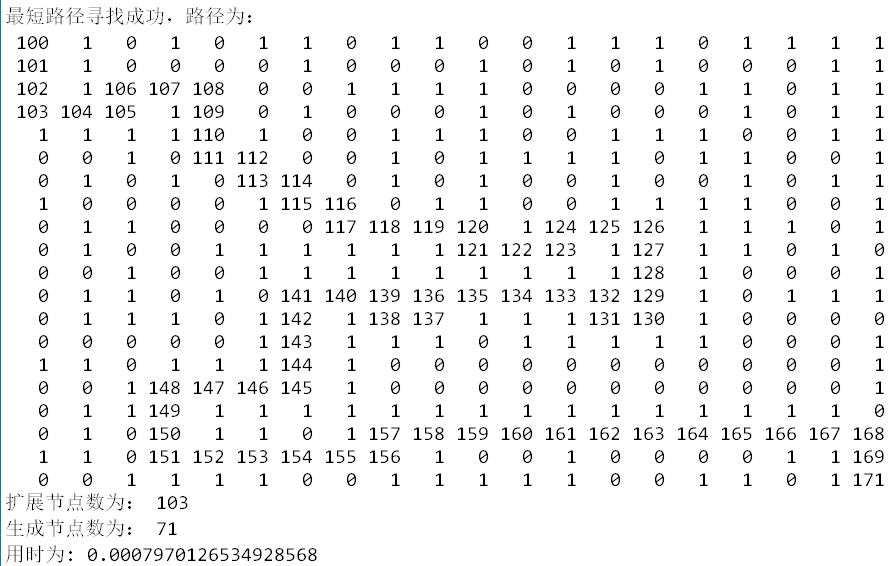


图 5.6

由图表可知，两种启发式函数所用时间相近，但相比所扩展的节点数和生成的节点数会减少，仔细分析可以发现启发式函数陷入的凹路径会少一些，更接近最优路径，但在远离目标点的路径行进时还是会陷进凹路径。

1. **分析不同地图**

上述过程一直采用地图2进行实验，为验证程序的普适性，现采用地图1进行实验。经过分析，选择上述较优的启发式函数进行分析。寻路问题初始节点为（0，0），目标节点为(5,5)，寻路路径如右图5.7，性能图如右图5.8。

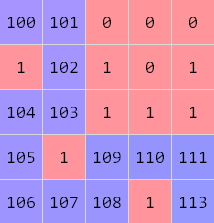


图 5.7

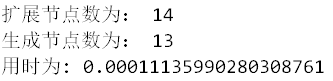


图 5.8

汇总3.2，3.3的性能报告如下表5.2。

表 5.2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **地图** | **启发式函数** | **扩展节点数** | **生成节点数** | **运行时间** |
| 地图1 |  | 14 | 13 | 0.00011135990280308761 |
| 地图2 |  | 107 | 79 | 0.0008827725796436425 |
| 地图2 |  | 103 | 71 | 0.0007970126534928568 |

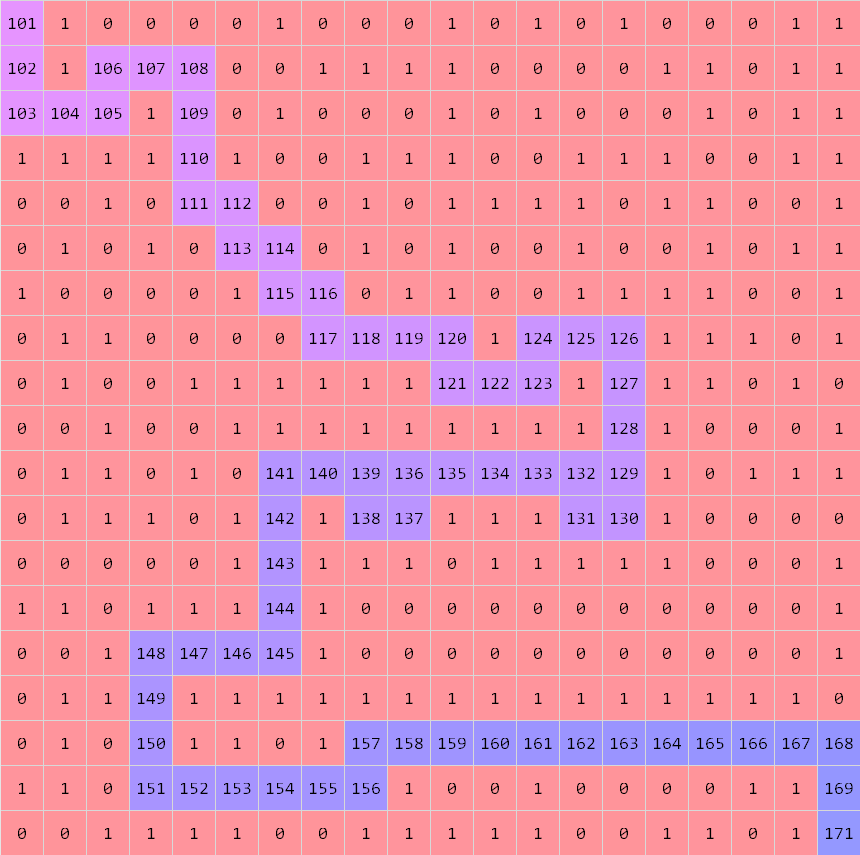
1. **总结**

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相比效果更好，但仍存在一些问题，需要继续找寻更优的启发式函数。

**附录**

**寻路路径图**

应用启发式函数对地图2的寻路路径彩绘图如下（因显示问题，第一行略去）。



**源程序**

# -\*- coding: utf-8 -\*-

"""

Created on Sat Nov 24 16:47:38 2018

@author: 魅梦

"""

import numpy as np

import time

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)

def gs(i, j):

return abs(i - startx) + abs(j - starty)

#a = np.mat(np.random.randint(0, 2, size = [10, 10]))

'''

a = np.mat([[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]])

'''

a = np.mat([[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, 0, 1, 0, 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]])

print("地图为：（1表示墙，0表示路）")

for l in range(len(a)):

for m in range(a[0].size):

print(a[l, m], end = ' ')

print('')

print('')

b = a

h, g = 0, 0

startx = 1

starty = 1

i = startx - 1

j = starty - 1

endx = 20

endy = 20

Open, Close = [[i + 1, j + 1]], [[i + 1, j + 1]]

Opens = [[i + 1, j + 1]]

road = 100

start = time.clock()

while(Open != []):

if(Open[-1] != [endx, endy]):

Open = []

minf = 10000

if(i + 1 < len(a) and a[i + 1, j] == 0):

f = gs((i + 1), j) + h2((i + 1), j)

if([i + 2, j + 1] not in Close):

Open.append([i + 2, j + 1])

if(f < minf):

minf = f

i1 = i + 1

j1 = j

if(i - 1 >= 0 and a[i - 1, j] == 0):

f = gs((i - 1), j) + h2((i - 1), j)

if([i, j + 1] not in Close):

Open.append([i, j + 1])

if(f < minf):

minf = f

i1 = i - 1

j1 = j

if(j + 1 < len(a) and a[i, j + 1] == 0):

f = gs(i, (j + 1)) + h2(i, (j + 1))

if([i + 1, j + 2] not in Close):

Open.append([i + 1, j + 2])

if(f < minf):

minf = f

i1 = i

j1 = j + 1

if(j - 1 >= 0 and a[i, j - 1] == 0):

f = gs(i, (j - 1)) + h2(i, (j - 1))

if([i + 1, j] not in Close):

Open.append([i + 1, j])

if(f < minf):

minf = f

i1 = i

j1 = j - 1

b[i, j] = road

road = road + 1

i = i1

j = j1

Close.append([i + 1, j + 1])

Opens.extend(Open)

else:

end = time.clock()

b[endx - 1, endy - 1] = road + 1

print("最短路径寻找成功，路径为：")

for l in range(len(b)):

for m in range(b[0].size):

#0表示format中第一个元素，>表示右对齐输出，3表示占三个字符

print("{0: >4}".format(b[l, m]), end = '')

print('')

print("扩展节点数为：", len(Opens))

print("生成节点数为：", len(Close))

print('用时为:', end - start)

'''

print('Close表为:', Close)

print("点的移动轨迹为：")

for k in range(len(Close)):

if(k < len(Close) - 1):

print(Close[k], "-->", end = '')

else:

print(Close[k])

'''

break

if(Open == []):

print("最短路径寻找失败，失败位置为：", Close[-1], "路径为：", Close)