School of Data and Computer Science, Sun Yat-sen University

Maze Problem

17341189 姚森舰

2019 年 8 月 31 日

目录

1 Task

- Please solve the maze problem (i.e., find the shortest path from the start point to the finish point) by using BFS or DFS (Python or C++)
- The maze layout can be modeled as an array, and you can use the data file `MazeData.txt` if necessary.
- Please send `E01_YourNumber.pdf` to `ai_201901@foxmail.com`, you can certainly use `E01_Maze.tex` as the LaTeXtemplate.

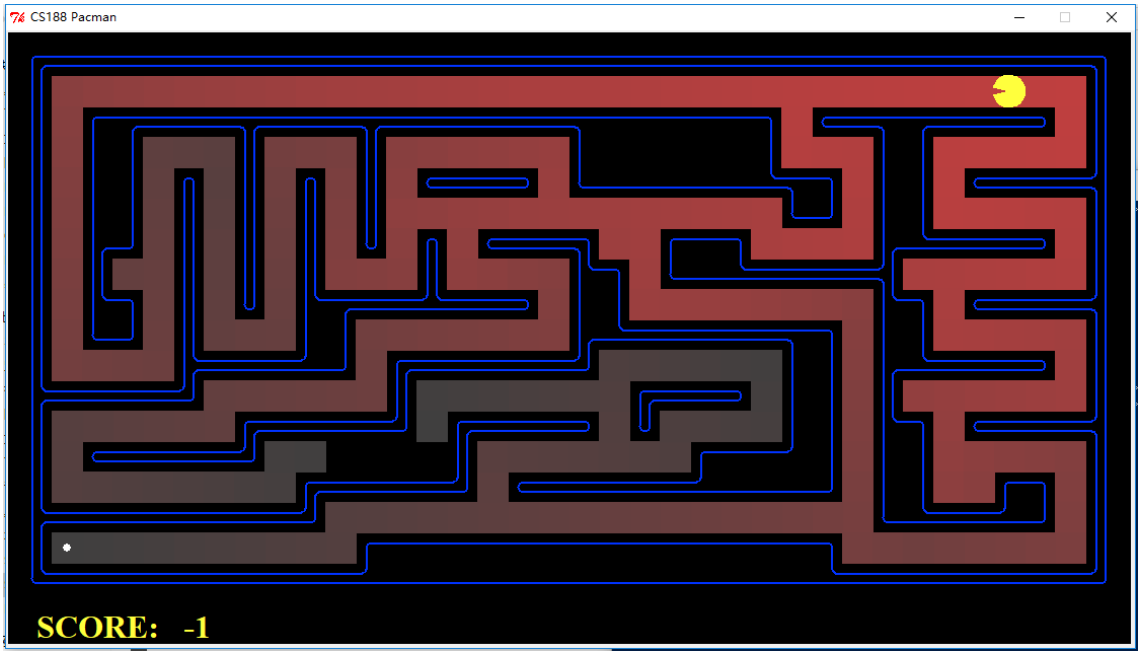


图 1: Searching by BFS or DFS

2 Codes

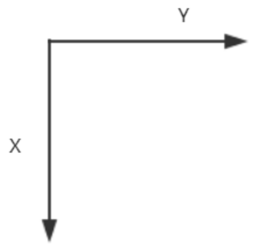
```python
# -*- coding: utf-8 -*-


def bfs(maze_map, S, E, row, col):
    '''利用广搜在迷宫图中寻找两点间最短路径

    :param maze_map: 二维列表, 迷宫图
    :param S: [int, int] 起点坐标
    :param E: [int, int 终点坐标
    :param row: int, 迷宫图的行数
    :param col: int, 迷宫图的列数
```

```
    :return: 有路则返回记录路线在各点的方向，否则返回None]
    '''
    move = [[-1, 0], [0, 1], [1, 0], [0, -1]]  # 上右下左
    visited = [[0] * col for _ in range(row)]  # 记录点是否被访问过
    # visited = [[0] * col * row]
    visited[S[0]][S[1]] = 1                    # 标记起点
    directions = []                            # 走过的路线，在各点处的方向
    for r in range(row):
        a_row = [0] * col
        directions.append(a_row)

    nodes = [S]                                # BFS将要访问的点的队列

    while len(nodes) != 0:
        node = nodes[0]                        # 取第一个点
        # print(node)
        for i in range(1, 5):                  # 1，2，3，4分别代表上右下左
            x = node[0] + move[i-1][0]         # 探索临近的点
            y = node[1] + move[i-1][1]

            if x == E[0] and y == E[1]:        # 到达终点
                directions[x][y] = i
                return directions
            # 如果坐标合法且未被访问过且有路可走，
            # 这里python可以使用链式比较，为了不搞混还是使用C的写法
            if x >= 0 and y >= 0 and x < row and y < col \
                    and visited[x][y] == 0 and maze_map[x][y] == ' ':
                nodes.append([x, y])           # 将合理的点加入待访问队列
                visited[x][y] = 1              # 标记为已访问
                directions[x][y] = i           # 记录行走的方向
            # print(nodes)
        del (nodes[0])                         # 删除已访问过的点

    return None


filename = 'MazeData.txt'
maze = []
S = []
E = []
row = 0
col = 0
with open(filename, 'r') as data:
    for line in data:
        line = line.strip()
        print(line)
        s_col = line.find('S')
```

```
        e_col = line.find('E')
        if s_col is not -1:
            S = [row, s_col]
            col = len(line)

        if e_col is not -1:
            E = [row, e_col]

        row += 1
        maze.append(list(line))

ans = bfs(maze, S, E, row, col)

if ans is None:
    print('No way to the end point.')
else:
    direction = ans[E[0]][E[1]]
    cur_x = E[0]
    cur_y = E[1]
    # for r in ans:
    #     print(r)
    # move = [[-1, 0], [0, 1], [1, 0], [0, -1]]  # 上右下左
    path = [E]
    while direction != 0:
        if direction == 1:
            cur_x += 1

        if direction == 2:
            cur_y -= 1

        if direction == 3:
            cur_x -= 1

        if direction == 4:
            cur_y += 1

        path.append([cur_x, cur_y])
        direction = ans[cur_x][cur_y]
        maze[cur_x][cur_y] = '*'

    maze[S[0]][S[1]] = 'S'
    path = path[::-1]
    print('The path is: ', path)
    print('Total length of the path: ', len(path)-1)
    for l in maze:
        print(''.join(l))
```

3 Results

需要注意的是，本次采用的坐标系表示如下：



输出结果如下：

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                 S%
% %%%%%%%%%%%%%%%%%%%%%%% %%%%%%%% %
% %%   %   %      %%%%%%%   %%     %
% %% % % % % %%%% %%%%%%%%% %% %%%%%
% %% % % % %             %% %%     %
% %% % % % % % %%%%  %%%    %%%%%% %
% %  % % %   %    %% %%%%%%%%      %
% %% % % %%%%%%%% %%        %% %%%%%
% %% %   %%       %%%%%%%%% %%     %
%    %%%%%% %%%%%%%      %% %%%%%% %
%%%%%%      %       %%%% %% %      %
%      %%%%%% %%%%% %    %% %% %%%%%
% %%%%%%      %       %%%%% %%     %
%       %%%%%% %%%%%%%%%%% %%  %% %
%%%%%%%%%%                 %%%%%% %
%E         %%%%%%%%%%%%%%%%       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


The path is:  [[1, 34], [1, 33], [1, 32], [1, 31], [1, 30], [1, 29], [1, 28], [1, 27], [1, 26], [1, 25], [2, 25], [3, 25], [3, 26],
[3, 27], [4, 27], [5, 27], [6, 27], [6, 26], [6, 25], [6, 24], [5, 24], [5, 23], [5, 22], [5, 21], [5, 20], [6, 20], [7, 20], [8, 20],
[8, 21], [8, 22], [8, 23], [8, 24], [8, 25], [8, 26], [8, 27], [9, 27], [10, 27], [11, 27], [12, 27], [13, 27], [14, 27], [15, 27],
[15, 26], [15, 25], [15, 24], [15, 23], [15, 22], [15, 21], [15, 20], [15, 19], [15, 18], [15, 17], [15, 16], [15, 15], [15, 14], [15, 13],
[15, 12], [15, 11], [15, 10], [16, 10], [16, 9], [16, 8], [16, 7], [16, 6], [16, 5], [16, 4], [16, 3], [16, 2], [16, 1]]
Total length of the path:  68
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                        *********S%
% %%%%%%%%%%%%%%%%%%%%%%%*%%%%%%%% %
% %%   %   %      %%%%%%%***%%     %
% %% % % % % %%%% %%%%%%%%%*%% %%%%%
% %% % % % %        *****%%*%%     %
% %% % % % % % %%%% *%%%****%%%%%% %
% %  % % %   %    %%*%%%%%%%%      %
% %% % % %%%%%%%% %%********%% %%%%%
% %% %   %%       %%%%%%%%%*%%     %
%    %%%%%% %%%%%%%      %%*%%%%%% %
%%%%%%      %       %%%% %%*%      %
%      %%%%%% %%%%% %    %%*%% %%%%%
% %%%%%%      %       %%%%%*%%     %
%       %%%%%% %%%%%%%%%%%*%%  %% %
%%%%%%%%%%******************%%%%%% %
%E*********%%%%%%%%%%%%%%%%       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

输出的列表就是路径上所有的点，包含了起点和终点，路径上有 69 个点，真正的路径长度应该是点的个数-1，也就是 68.

BFS 算法不算陌生，但是在实现时，在记录路径这个地方有点难度，我是记录下了路径中到每个点的方向来记录路径。实际上如果要输出路径的话，可能 DFS 更容易实现一些。另外，在初始化 visited 列表时遇到了一个有点坑的问题，如果像下面这样写：

```
visited = [[0] * 3] * 3
```

```
visited[0][0] = 1
print(visited)
```

输出会是：[[1, 0, 0], [1, 0, 0], [1, 0, 0]]，而不是想象中的：[[1, 0, 0], [0, 0, 0], [0, 0, 0]]. 这是 Python 中一个需要让人注意的地方。所以最终还是用来了一个循环来初始化。