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

Maze Problem

17341203 Yixin Zhang

August 30, 2019

Contents

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.

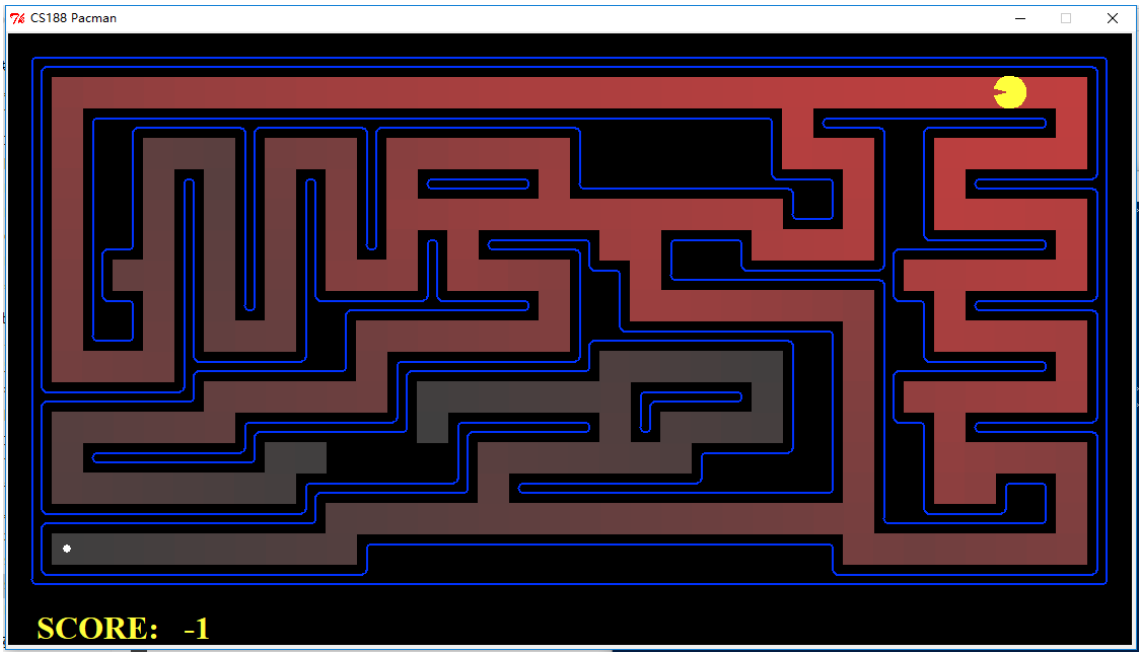

Figure 1: Searching by BFS or DFS

2 Code Framework

There are several code fragments that can be shared among different search algorithms such as BFS, DFS and A*, e.g., code of file-reading, setting characters, printing results and so on.

settings.py

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

filename = 'MazeData.txt'
wall_char = '%'
space_char = ' '
start_char = 'S'
end_char = 'E'
```

main.py

```
# -*- coding: utf-8 -*-
import numpy as np
```

```
3  from queue import Queue
4  from bfs import bfs
5  import settings
6
7
8  def makePath(maze_origin, prev, start, end):
9      '''
10     Make a path presented by list from the prev matrix given.
11     '''
12     path = []
13     current = end
14     while current != start:
15         path.insert(0, current) # insert at the head of the path
16         current = prev[current[0]][current[1]]
17     return path
18
19
20 def makeMazeWithPath(maze_origin, path):
21     '''
22     Draw the path into the maze.
23     '''
24     maze = maze_origin.copy()
25
26     for i in range(len(path)-1):
27         if path[i][0] + 1 == path[i+1][0]:
28             maze[path[i]] = ' '
29         elif path[i][0] - 1 == path[i+1][0]:
30             maze[path[i]] = ' '
31         elif path[i][1] - 1 == path[i+1][1]:
32             maze[path[i]] = ' '
33         elif path[i][1] + 1 == path[i+1][1]:
34             maze[path[i]] = ' '
35         else:
36             exit('[-] Path Error!')
37
38     return maze
39
40
41 if __name__ == '__main__':
42     with open(settings.filename) as file:
43         maze = [] # list of list
44         for i, line in enumerate(file):
45             line = line.strip() # delete EOL
46             start_col = line.find(settings.start_char)
47             end_col = line.find(settings.end_char)
48             if start_col != -1:
49                 start = (i, start_col)
50             if end_col != -1:
51                 end = (i, end_col)
52             maze.append(list(line)) # append current row
53     maze = np.array(maze) # convert to numpy 2D-array
54
55     prev = bfs(maze, start, end) # use bfs() or astar()
56     path = makePath(maze, prev, start, end)
57
58     # Print the length of the path
```

```
    print('[+] Steps:', len(path))
    # Print the path in a list of coordinates
    print('[+] Path (in coordinates):\n', path)
    # Print the figure of the maze and the path
    print('[+] Path (in figure):')
    maze_with_path = makeMazeWithPath(maze, path)
    for line in maze_with_path:
        print(''.join(line))
```

3 Searching by BFS

3.1 Overview

BFS, a.k.a Breadth-First Search, is the simplest of the graph search algorithms. It is a search algorithm categoried as uninformed search. Breadth First Search explores equally in all directions. This is an incredibly useful algorithm, not only for regular path finding, but also for procedural map generation, flow field pathfinding, distance maps, and other types of map analysis. [1]

I use numpy arrays to store the maze (which is read from a text file), and use a list of coordinates to store the path. After finding the path successfully, the length of the path, the list of those coordinates and a figure with path presented by arrows inside the maze are printed.

3.2 Code

I didn't use a "visited" matrix. Instead, I change the corresponding point into wall_char to show that the point has been visited.

bfs.py

```
# -*- coding: utf-8 -*-
from queue import Queue
import settings


def bfs(maze_origin, start, end):
    '''
    Find a path in the maze given using BFS.
    '''
    maze = maze_origin.copy()

    rows = len(maze)   # num of rows of the maze
    cols = len(maze[0]) # num of columns of the maze
    prev = [[(-1, -1) for j in range(cols)] for i in range(rows)] # record the path

    frontier = Queue()
    frontier.put(start)

    while not frontier.empty():
        current = frontier.get()
        if current == end: # have explored to the end
            break

        row = current[0]
        col = current[1]

```

```
        # up
        if row > 0 and maze[row-1][col] != settings.wall_char:
            frontier.put((row-1, col))
            maze[row-1][col] = settings.wall_char
            prev[row-1][col] = (row, col)

        # down
        if row < rows - 1 and maze[row+1][col] != settings.wall_char:
            frontier.put((row+1, col))
            maze[row+1][col] = settings.wall_char
            prev[row+1][col] = (row, col)

        # left
        if col > 0 and maze[row][col-1] != settings.wall_char:
            frontier.put((row, col-1))
            maze[row][col-1] = settings.wall_char
            prev[row][col-1] = (row, col)

        # right
        if col < cols - 1 and maze[row][col+1] != settings.wall_char:
            frontier.put((row, col+1))
            maze[row][col+1] = settings.wall_char
            prev[row][col+1] = (row, col)

    return prev
```

3.3 Result

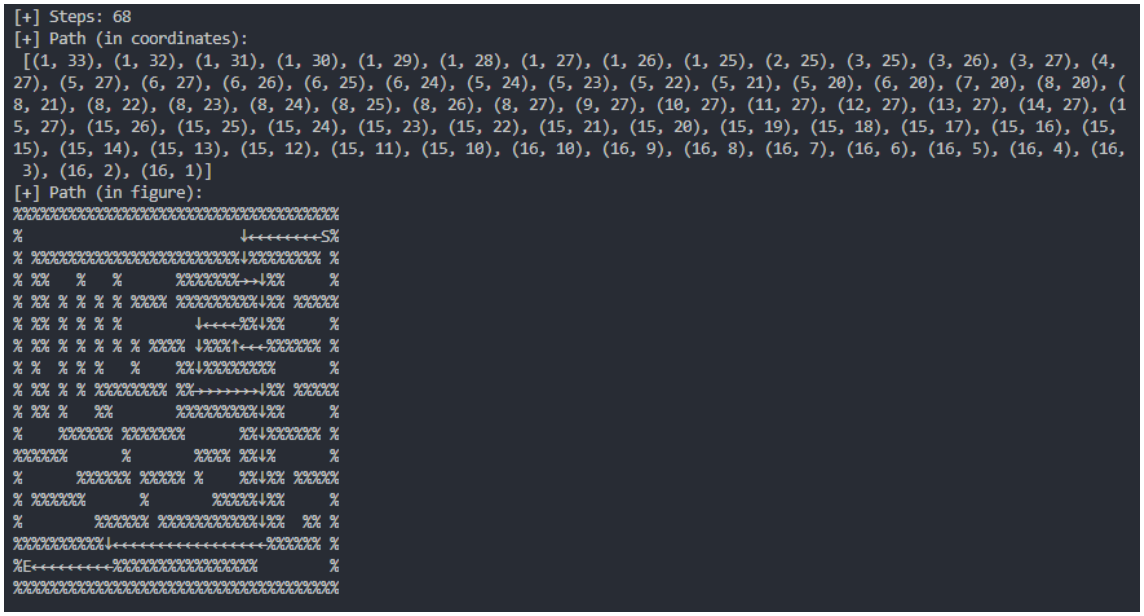

Figure 2: Result of BFS

As shown in the figure above, the algorithm find a path of length 68. The path is printed as well.

4 Discussion

In this task, I use BFS to find a path from start point to end point in a given maze. BFS is simple and easy to understand, because we have learnt it in previous lessons. BFS satisfies optimality because of three facts:

- All shorter paths are expanded before any longer path;
- There are finitely many paths of a certain length;
- Eventually we must examine all paths of length d, and thus find the shortest solution.

However, there exist more powerful algorithms, such as A*, which I will study in the near future.

This is the first time that I have written a report in LaTeX, and I am happy to learn a new tool that will be very useful in future.

References

[1] Introduction to the A* Algorithm,
`https://www.redblobgames.com/pathfinding/a-star/introduction.html`