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

Maze Problem (Lab 1)

17341015 Hongzheng Chen

August 29, 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.

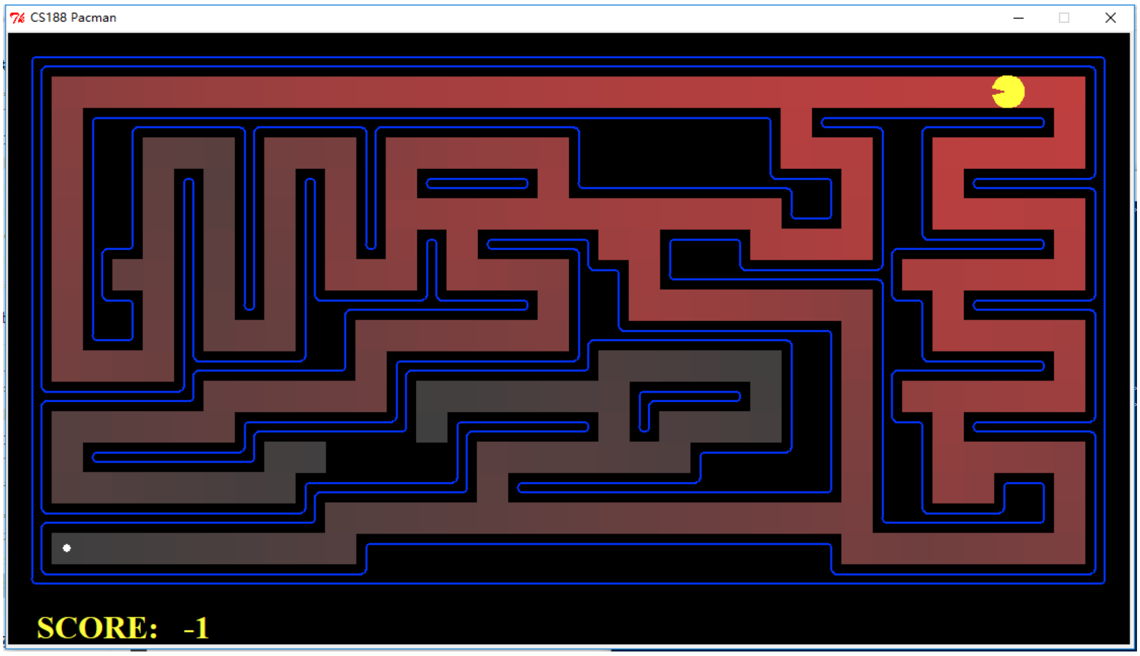

Figure 1: Searching by BFS or DFS

2 Codes

```
import sys
import time
import queue

if len(sys.argv) > 2:
    wallmark = '1'
    roadmark = '0'
    infile = "Maze1.txt"
else:
    wallmark = '%'
    roadmark = ' '
    infile = "Maze.txt"

# Initialization
```

```python
maze = []
s, e = (0,0), (0,0)
best = ""
shortestLength = 0x3f3f3f3f

# Read in maze from file
with open(infile,"r") as file:
    for (i,line) in enumerate(file):
        if not line[-1] in [wallmark,'S','E',roadmark]:
            line = line[:-1]
        maze.append(line)
        j = line.find('S')
        if j != -1:
            s = (i,j)
        j = line.find('E')
        if j != -1:
            e = (i,j)

def validQ(x,y):
    """
    Auxiliary function to test if (x,y) is a valid position to move to
    """
    if y < 0 or y >= len(maze[0]) or x < 0 or x >= len(maze) or maze[x][y] in [
        ↪ wallmark,'*']:
        return False
    else:
        return True

def explore(x,y,d,visited,q):
    """
    Auxiliary function for BFS

    (x,y): next position
    d: direction to the previous position, L, R, U, D
    visited: a bool array storing whether a position is visited
    """
    if validQ(x,y) and visited[x][y] == 0:
        visited[x][y] = d
        q.append((x,y))

def BFS(x,y):
    """
    Breath-First Search for the maze problem

    (x, y): the initial position
    """
    global best, shortestLength
    # initialization
    q = [(x,y)]
```

```
    visited = []
    for i in range(len(maze)):
        visited.append([0]*len(maze[0]))
    visited[x][y] = -1
    # traversal
    while len(q) > 0:
        (x,y), q = q[0], q[1:]
        if (x,y) == e:
            break
        explore(x-1,y,'D',visited,q)
        explore(x+1,y,'U',visited,q)
        explore(x,y-1,'R',visited,q)
        explore(x,y+1,'L',visited,q)
    # generate the shortest path
    (x, y) = e
    shortestLength = 0
    while not visited[x][y] == -1:
        shortestLength += 1
        if visited[x][y] == 'D':
            x = x+1
        elif visited[x][y] == 'U':
            x = x-1
        elif visited[x][y] == 'L':
            y = y-1
        elif visited[x][y] == 'R':
            y = y+1
        maze[x] = maze[x][:y] + '*' + maze[x][y+1:]
    maze[s[0]] = maze[s[0]][:s[1]] + 'S' + maze[s[0]][s[1]+1:]
    for i in range(len(maze)):
        best += maze[i] + "\n"

def DFS(x,y,depth):
    """
    Depth-First Search for the maze problem

    (x, y): the current position
    depth: current path length
    """
    global best, shortestLength
    if not validQ(x,y):
        return
    if x == e[0] and y == e[1] and depth < shortestLength:
        shortestLength = depth
        best = ""
        maze[s[0]] = maze[s[0]][:s[1]] + 'S' + maze[s[0]][s[1]+1:]
        maze[e[0]] = maze[e[0]][:e[1]] + 'E' + maze[e[0]][e[1]+1:]
        for i in range(len(maze)):
            best += maze[i] + "\n"
        maze[s[0]] = maze[s[0]][:s[1]] + '*' + maze[s[0]][s[1]+1:]
```

```
        maze[e[0]] = maze[e[0]][:e[1]] + ' ' + maze[e[0]][e[1]+1:]
        return
    maze[x] = maze[x][:y] + '*' + maze[x][y+1:]
    DFS(x,y-1,depth+1)
    DFS(x+1,y,depth+1)
    DFS(x-1,y,depth+1)
    DFS(x,y+1,depth+1)
    maze[x] = maze[x][:y] + roadmark + maze[x][y+1:]

def heuristic(des,curr):
    """
    Calculate the L1 distance
    """
    return abs(des[0] - curr[0]) + abs(des[0] - curr[0])

def AStar(x,y):
    """
    A* algorithm for the maze problem

    (x, y): the initial position
    """
    global best, shortestLength
    # initialization
    q = queue.PriorityQueue()
    q.put((0,(x,y)))
    prev = {}
    prevCost = {}
    prev[s] = None
    prevCost[s] = 0
    # traversal
    while not q.empty():
        _, (x,y) = q.get()
        if (x,y) == e:
            break
        for pos in [(x-1,y),(x+1,y),(x,y-1),(x,y+1)]:
            newCost = prevCost[(x,y)] + 1
            if validQ(pos[0],pos[1]) and ((pos not in prev) or newCost < prevCost[
                ↪ pos]):
                prevCost[pos] = newCost
                priority = newCost + heuristic(e,pos)
                q.put((priority,pos))
                prev[pos] = (x,y)
    # generate shortest path
    p = e
    shortestLength = 0
    while prev[p] != None:
        shortestLength += 1
        x, y = p[0], p[1]
        maze[x] = maze[x][:y] + '*' + maze[x][y+1:]
```

```
        p = prev[p]
    maze[e[0]] = maze[e[0]][:e[1]] + 'E' + maze[e[0]][e[1]+1:]
    maze[s[0]] = maze[s[0]][:s[1]] + 'S' + maze[s[0]][s[1]+1:]
    for i in range(len(maze)):
        best += maze[i] + "\n"

# Initial state
start = time.time()
if sys.argv[1] == "DFS":
    DFS(s[0],s[1],0)
    name = "DFS"
elif sys.argv[1] == "BFS":
    BFS(s[0],s[1])
    name = "BFS"
elif sys.argv[1] == "A*":
    AStar(s[0],s[1])
    name = "A*"
end = time.time()

# Output
print(name,"Shortest length: ",shortestLength)
print("Running time: {:.4f}s".format(end-start))
print(best)
```

3 Results

In this experiment, I use three methods to find the shortest path of the maze, including BFS, DFS, and A* algorithm. The results are shown in Fig. 2.

The first line of each output gives the length of the shortest path of the problem, which are **all** 68 obtained by the three algorithms.

The second line shows the running time. Surprisingly, **BFS runs the fastest, A* ranks the second, and DFS is extremely slow**. It may be caused by the relatively simple solution of the problem that A* cannot show all its power.

The rest of the outputs show the maze and the shortest path which is marked out in `*`. We can see that all the three algorithms give correct solutions.

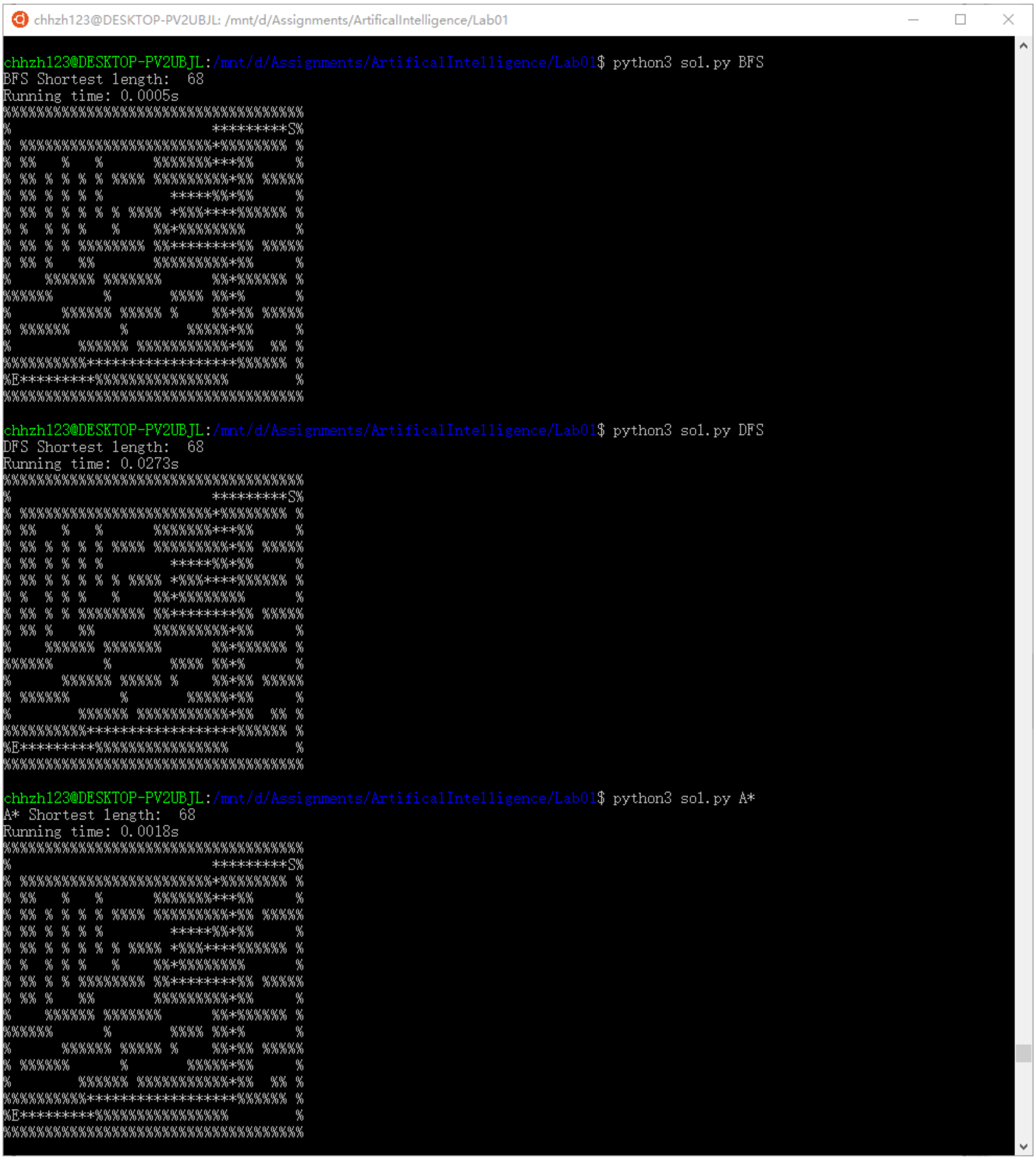

Figure 2: Experimental results of three search algorithms

References

[1] Red Blob Games, Introduction to the A⋆ Algorithm, https://www.redblobgames.com/pathfinding/a-star/introduction.html