

Practical - 7

Implementing a Maze Solver using AI Search Algorithms (BFS & DFS).

Objective: Solve AI search problems using Graph Search Algorithms..

Explanation:

Maze Representation:

- The maze is represented as a list of lists (a 2D grid).
- Each inner list represents a row.
- '#' indicates a wall.
- ' ' indicates an open path.
- 'S' is the starting point.
- 'E' is the ending point.

Example: Tree Expansion for Maze Search

Assume a simple maze grid (S = Start, E = End):

```
#####
```

```
#S  #
```

#E#

#####

BFS/DFS Exploration Tree (from 'S' at (1,1))

```
(1,1) S
|
+-- (1,2)
|   |
|   +-- (1,3)
|       |
|       +-- (2,3) E
|
+-- (2,1)
```

Explained:

- The root node $(1,1)$ is the start position 'S'.
- Possible moves:
 - Right to $(1,2)$, then right to $(1,3)$, then down to $(2,3)$ (which is 'E').
 - Down to $(2,1)$, but from there only wall or already visited positions are possible.

BFS Path:

- Explores all neighbors at each "layer"—would reach $(2, 3)$ through $(1, 2)$, $(1, 3)$ as shortest.

DFS Path:

- Could go deep along one path: e.g., from $(1, 1)$ to $(1, 2)$, then to $(1, 3)$, and finally to $(2, 3)$.
-

Visual Outline (Text Tree)

```

S (1,1)
|
+-- Right -> (1,2)
|   |
|   +-- Right -> (1,3)
|       |
|       +-- Down -> (2,3) [E]
|
+-- Down -> (2,1)

```

- BFS/DFS both expand nodes like a tree rooted at start, each branch showing a possible move.
- For BFS, all nodes on each level are explored before going deeper.
- For DFS, a full branch is explored as far as possible before backtracking.

This tree structure helps visualize how AI search algorithms traverse decision points in a maze.

```
Activities Terminal Mon 14:27
Open dfs.py Save
bfs.py x dfs.py x

1 def find_start_end(maze):
2     start = None
3     end = None
4     for r in range(len(maze)):
5         for c in range(len(maze[0])):
6             if maze[r][c] == 'S':
7                 start = (r, c)
8             elif maze[r][c] == 'E':
9                 end = (r, c)
10    return start, end
11
12 def is_valid(r, c, rows, cols, maze, visited):
13     return (0 <= r < rows and 0 <= c < cols and
14            maze[r][c] != '#' and (r, c) not in visited)
15
16 directions = [(0,1), (0,-1), (1,0), (-1,0)]
17
18 def dfs(maze):
19     rows, cols = len(maze), len(maze[0])
20     start, end = find_start_end(maze)
21     if not start or not end:
22         return None
23
24     stack = [(start, [start])]
25     visited = set([start])
26
27     while stack:
28         (r, c), path = stack.pop()
29         if (r, c) == end:
30             return path
31         for dr, dc in directions:
32             nr, nc = r + dr, c + dc
33             if is_valid(nr, nc, rows, cols, maze, visited):
34                 visited.add((nr, nc))
35                 stack.append(((nr, nc), path + [(nr, nc)]))
36
37     return None
```

```
s@28: ~
File Edit View Search Terminal Help
s@28:~$ python3 --version
Python 3.6.9
s@28:~$ python bfs.py
BFS path:
#####
#S*# #
# *# #
# *#*#
# *#*#
# #E#
#####
s@28:~$ python dfs.py
DFS path:
#####
#S*# #
# *# #
# *#*#
# *#*#
# #E#
#####
s@28:~$
```

Activities Terminal Mon 14:23 bfs.py Save

```
1 from collections import deque
2
3 def find_start_end(maze):
4     start = None
5     end = None
6     for r in range(len(maze)):
7         for c in range(len(maze[0])):
8             if maze[r][c] == 'S':
9                 start = (r, c)
10            elif maze[r][c] == 'E':
11                end = (r, c)
12    return start, end
13
14 def is_valid(r, c, rows, cols, maze, visited):
15     return (0 <= r < rows and 0 <= c < cols and
16            maze[r][c] != '#' and (r, c) not in visited)
17
18 directions = [(0,1), (0,-1), (1,0), (-1,0)]
19
20 def bfs(maze):
21     rows, cols = len(maze), len(maze[0])
22     start, end = find_start_end(maze)
23     if not start or not end:
24         return None
25
26     queue = deque([(start, [start])])
27     visited = set([start])
28
29     while queue:
30         (r, c), path = queue.popleft()
31         if (r, c) == end:
32             return path
33         for dr, dc in directions:
34             nr, nc = r + dr, c + dc
35             if is_valid(nr, nc, rows, cols, maze, visited):
36                 visited.add((nr, nc))
37                 queue.append((nr, nc), path + [(nr, nc)])
38     return None
```

s@28: ~

```
File Edit View Search Terminal Help
s@28:~$ python3 --version
Python 3.6.9
s@28:~$ python bfs.py
BFS path:
#####
#S***#
#    #
#    #
#    #
#    #
#    #
#E#
#####
s@28:~$
```

Python Tab Width: 8 Ln 48, Col 1 INS

