MARIA KOILALOU *JUSTINE LE BOURG*

Artificial Intelligence

Laboratory 1

Variant 1

This code implements two search algorithms (Breadth-First Search and Depth-First Search) to solve a maze loaded from a file. The maze is represented as a two-dimensional list of characters, where 'S' represents the start position, 'E' represents the end position, and '# represents a wall that cannot be traversed. The search algorithms are implemented as functions inside the solve_maze function, and the algorithm to use is specified by the user at runtime.

- The visualize function takes the maze and the path found by the algorithm as input, and prints a visualization of the maze with the path represented by asterisks ('*') when the position belongs to the path and by question marks (‘?’) when the position is visited by the algorithm.

- The load_maze function reads the maze from a file named 'maze.txt', which should be in the same directory as the script. The maze is represented as a list of strings, with each string representing a row of the maze.

  maze.txt example used in code:

```
##########S#
#         #
# #####   #
# #   #   #
# # ### # #
# # #   # #
#   # # # #
### # # # #
#   #     #
# # # # # #
#     #   #
#####E#####
```

  The maze.txt represents a maze using (‘#’) for walls and (‘ ‘) for path space and (‘S’) for start point and (‘E’) for end point. The code will work for every file with the same representation of a maze.

- The find_start_end function searches the maze to find the positions of the start and end points.

- The is_valid_position function checks if a given position is valid (i.e., not outside the bounds of the maze or a wall).

- The get_neighbors function returns a list of valid neighboring positions for a given position.

- The bfs and dfs functions implement Breadth-First Search and Depth-First Search, respectively. They use a queue and a stack to keep track of the positions to visit next, and a set to keep track of the visited positions. They also use a dictionary (parent) to keep track of the parent position of each visited position, so that the path can be reconstructed once the end position is found

BFS explores the search space in a breadth-first manner, meaning it visits all the neighbors of the current node before moving on to the next level of nodes. This guarantees that the shortest path to the goal node is found, as the goal node is visited only when it is at the shallowest level of the search tree.

DFS, on the other hand, explores the search space in a depth-first manner, meaning it visits one neighbor of the current node before moving on to the next node. This can lead to a situation where a longer path to the goal node is found before the shortest path.

Additionally, the order in which the neighbors are explored can also affect the path found. For example, if BFS and DFS explore the same set of neighbors in a different order, they might end up finding different paths.

In summary, BFS and DFS can give different paths due to their different exploration strategies and the order in which they explore the search space. If you want to ensure that you always find the shortest path, you should use BFS.

INPUT/OUTPUT

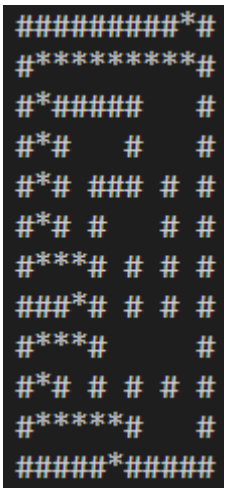
When running the program,you will see the maze and you will be prompted to choose between two search algorithms: BFS or DFS :

```
Maze:
#########S#
#         #
# #####   #
# #   #   #
# # ### # #
# # #   # #
#   # # # #
### # # # #
#   #     #
# # # # # #
#     #   #
#####E#####
Enter 'bfs' to use breadth-first search or 'dfs' to use depth-first search: d
```

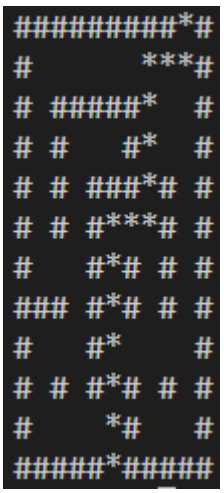
After you choose which algorithm you want to display the program will start to visualize the algorithm's logic step by step by printing ('?') each time it visits a position in the maze until it reaches the end and prints the final solution with the path.

```
Step 15:
#########?#
#??????????#
#?#####   #
#?#   #   #
#?# ### # #
#?# #   # #
#?  # # # #
### # # # #
#   #     #
# # # # # #
#     #   #
#####E#####
  Visited: (6, 1)
  Added to queue: (6, 2)
```

If you choose the BFS option, the code will provide you with the results of that algorithm method.



If you choose the DFS option, the code will provide you with the results of that algorithm method.



As you can see the BFS and DFS algorithm gives a different path for the same maze. This is a result of the different logic and algorithms use when they explore the same set of neighbors.