Problem Statement

This project solves maze puzzles using different algorithms. The maze is a grid with open spaces (0) and walls (1). The goal is to find a path from the top-left corner to the bottom-right corner. The user can choose between BFS, DFS, and Dijkstra's algorithms to find the path.

Data Structures Used

Maze: Stored as a 2D list of integers.

Visited: A 2D boolean list to track visited cells.

Queue (BFS): To explore neighbors level by level.

Stack (DFS): To explore paths deeply.

Priority Queue (Dijkstra): To select the next closest node by cost.

Parent Dictionary: To reconstruct the path after finding the end.

Code Walkthrough

Load Maze: Reads maze(s) from a file into 2D lists. Reads a text file and loads multiple mazes. Each maze is a 2D list where: 0 = walkable path and 1 = wall. Mazes are separated by blank lines in the file.

Check Valid Moves: Ensures moves stay inside the maze, are open, and not visited.

Path Build: Reconstructs the path using the parent dictionary.

BFS: Explores neighbors level-wise for shortest path in unweighted grid. In this, we use a queue (deque) to explore nodes level by level. It guarantees the shortest path in unweighted graphs.

DFS: Explores deep paths, may not find shortest path. Use stacks.

Dijkstra: Finds shortest path using a priority queue (handles weighted paths, here cost=1).

Display: Prints the maze showing start, end, walls, and path. Prints the maze showing:S for start ,E for end, # for walls, . for the found path

User Interaction: Lets user select algorithm per maze and shows results. For each maze, it sets the top-left corner as start and bottom-right as end and displays maze and path, and prints path cost.

Complexity Analysis

BFS and DFS:

Time: O(V), where V = number of cells

Space: O(V) for visited, queue/stack, and parents

Dijkstra:

Time: O(V log V) due to priority queue

Space: O(V) for distances, parents, and queue

All algorithms run efficiently for typical maze sizes.