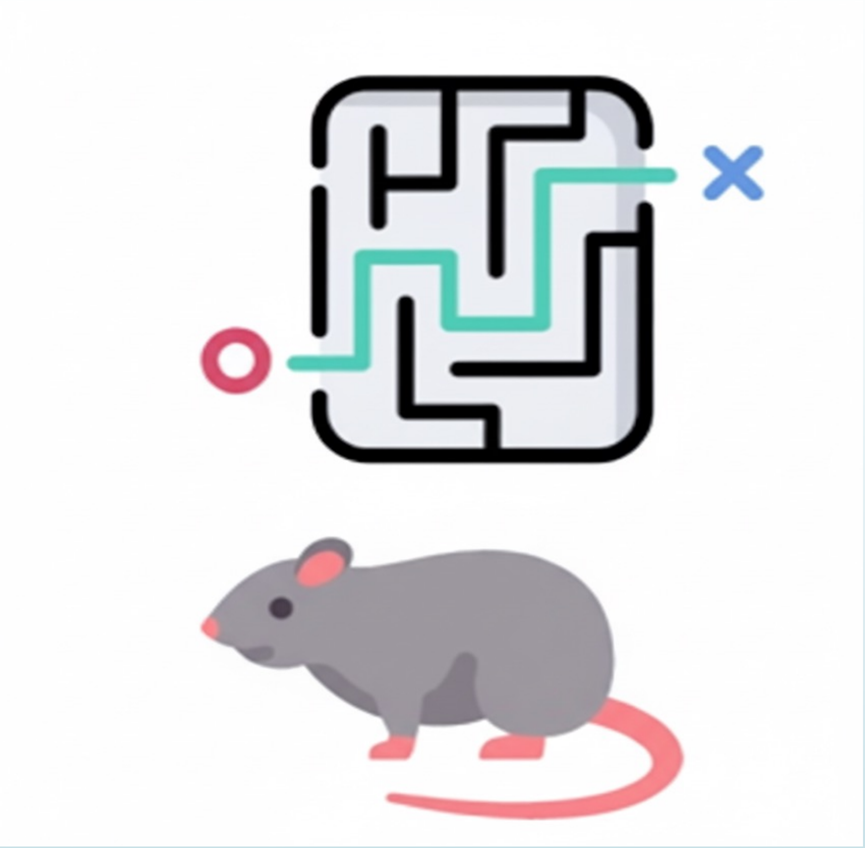
Power Of Algorithm

Rat In Maze



Presented By: Team 08

073: Harshal Vilas Tarmale

065: Gaurav Anil Sontakke

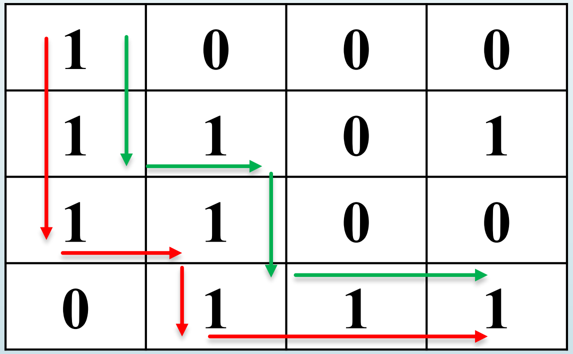
Problem Statement

Consider **a rat placed at position (0, 0) in an n x n square matrix maze[][].** The rat's goal is to reach the **destination at position (n-1, n-1).** The rat can move in four possible directions: **'U'(up), 'D'(down), 'L' (left), 'R' (right).**

1	0	0	0
1	1	0	1
1	1	0	0
0	1	1	1

The matrix contains only two possible values:
0: A blocked cell through which the rat cannot travel.
1: A free cell that the rat can pass through.

Your task is to find all possible paths the rat can take to reach the destination, starting from (0, 0) and ending at (n-1, n-1), under the condition that the rat cannot revisit any cell along the same path.



1	0	0	0
1	1	0	1
1	1	0	0
0	1	1	1

Algorithm Breakdown

1. Recursive Backtracking Approach

The solution uses Depth First Search with backtracking to explore all possible paths:

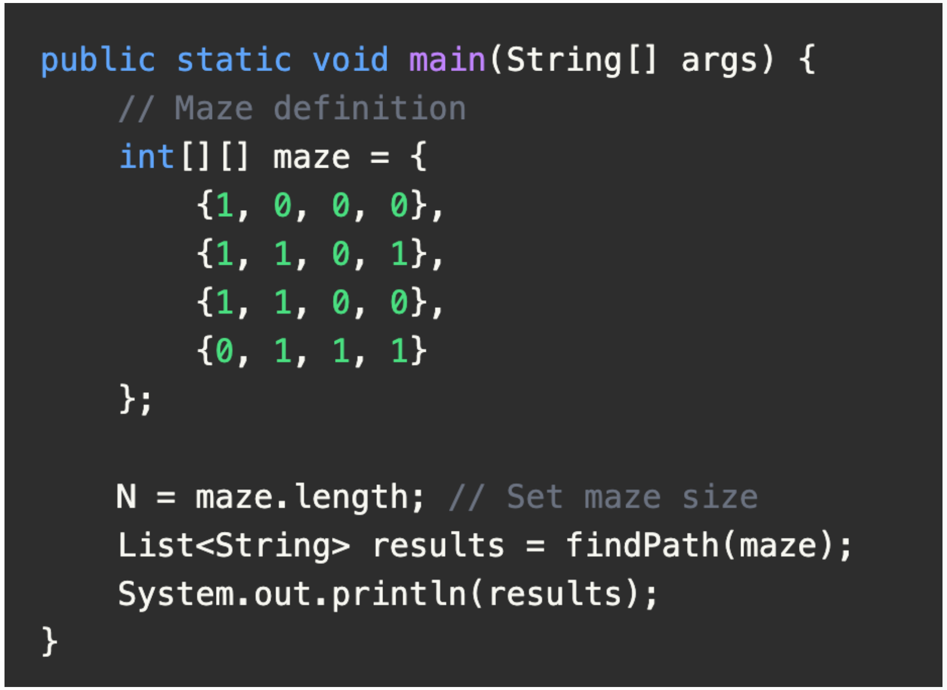
- Recursively explore all four directions (Down, Up, Right, Left)
- Mark current cell as visited (set to 0)
- If destination is reached, save the path
- Backtrack by unmarking the current cell (set back to 1)

2. Base Cases

The recursion stops when:

- Current cell is out of bounds (x or y is -1 or N)
- Current cell is blocked (value is 0)
- Destination cell is reached (x = N-1 and y = N-1)

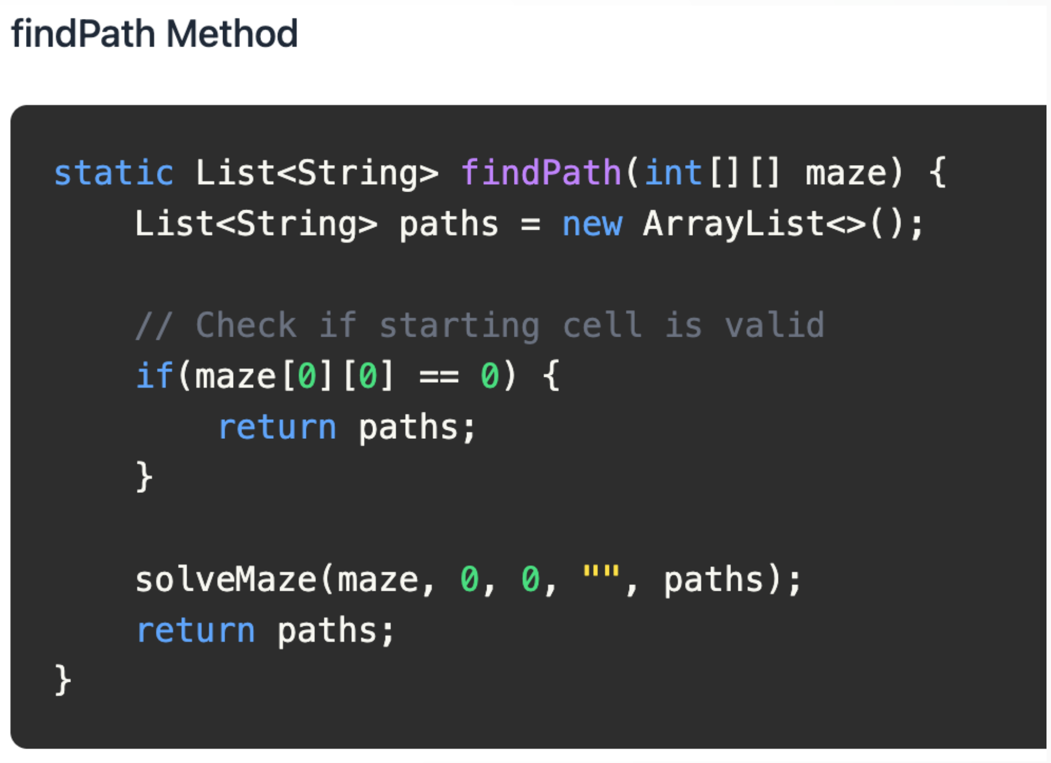
Code Walkthrough



```java
public static void main(String[] args) {
    // Maze definition
    int[][] maze = {
        {1, 0, 0, 0},
        {1, 1, 0, 1},
        {1, 1, 0, 0},
        {0, 1, 1, 1}
    };

    N = maze.length; // Set maze size
    List<String> results = findPath(maze);
    System.out.println(results);
}
```

Code Walkthrough



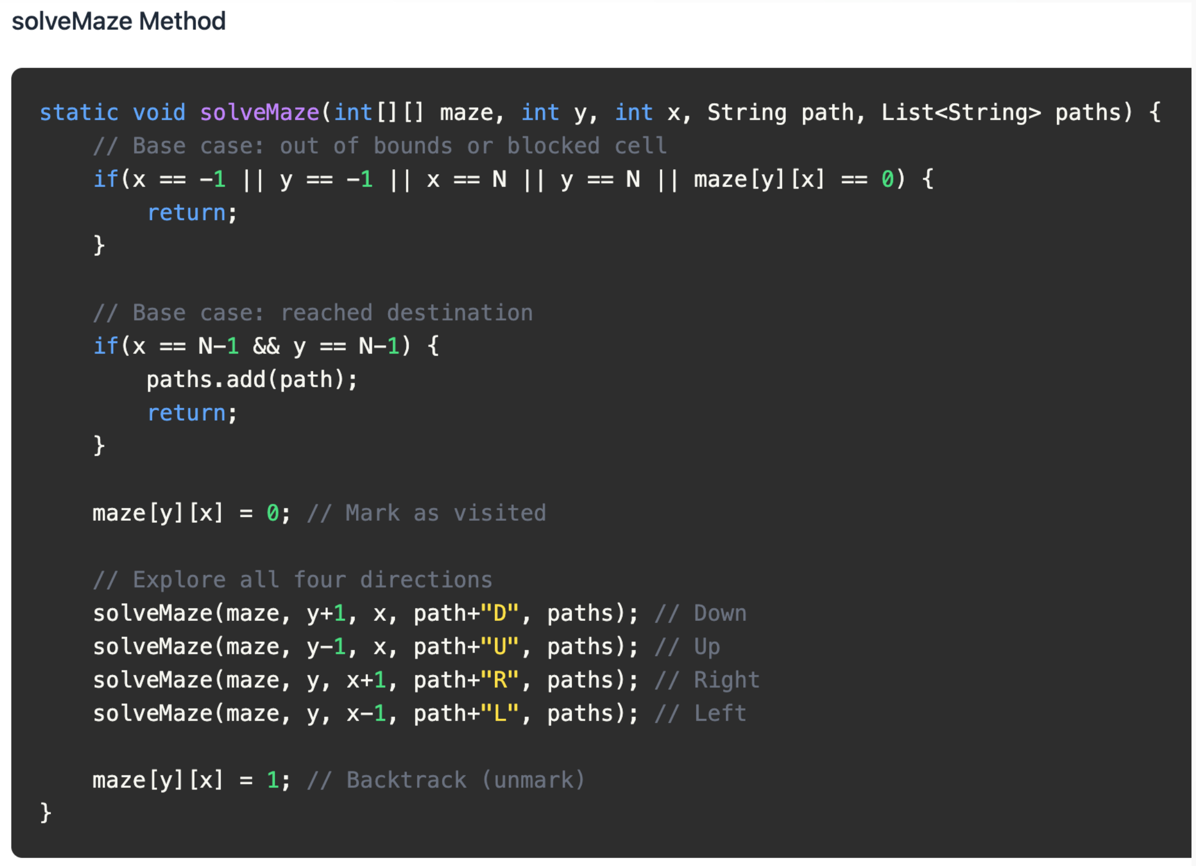
findPath Method

```java
static List<String> findPath(int[][] maze) {
    List<String> paths = new ArrayList<>();

    // Check if starting cell is valid
    if(maze[0][0] == 0) {
        return paths;
    }

    solveMaze(maze, 0, 0, "", paths);
    return paths;
}
```

Code Walkthrough



solveMaze Method

```
static void solveMaze(int[][] maze, int y, int x, String path, List<String> paths) {
    // Base case: out of bounds or blocked cell
    if(x == -1 || y == -1 || x == N || y == N || maze[y][x] == 0) {
        return;
    }

    // Base case: reached destination
    if(x == N-1 && y == N-1) {
        paths.add(path);
        return;
    }

    maze[y][x] = 0; // Mark as visited

    // Explore all four directions
    solveMaze(maze, y+1, x, path+"D", paths); // Down
    solveMaze(maze, y-1, x, path+"U", paths); // Up
    solveMaze(maze, y, x+1, path+"R", paths); // Right
    solveMaze(maze, y, x-1, path+"L", paths); // Left

    maze[y][x] = 1; // Backtrack (unmark)
}
```

Key Takeaways

- Backtracking

The algorithm marks cells as visited and unmarks them during backtracking to explore all possible paths.

- Recursion

DFS is implemented recursively to explore each path until it hits a base case.

- Path Construction

Paths are built incrementally by appending direction letters at each recursive step.

Thank You !