

DAY 22:

ASSIGNMENT 2:

Task 2: Rat in a Maze

Implement a function `bool SolveMaze(int[,] maze)` that uses backtracking to find a path from the top left corner to the bottom right corner of a maze. The maze is represented by a 2D array where 1s are paths and 0s are walls. Find a rat's path through the maze. The maze size is 6x6.

ANSWER:

```
public class RatInMaze {  
    private static final int N = 6;  
  
    // Function to print the solution matrix  
    private static void printSolution(int[][] solution) {  
        for (int i = 0; i < N; i++) {  
            for (int j = 0; j < N; j++) {  
                System.out.print(solution[i][j] + " ");  
            }  
            System.out.println();  
        }  
    }  
  
    // Function to check if x, y is a valid index for N*N maze  
    private static boolean isSafe(int[][] maze, int x, int y) {  
        return (x >= 0 && x < N && y >= 0 && y < N && maze[x][y] == 1);  
    }  
  
    // Function to solve the maze using backtracking  
    private static boolean solveMaze(int[][] maze) {  
        int[][] solution = new int[N][N];  
  
        if (!solveMazeUtil(maze, 0, 0, solution)) {
```

```

        System.out.println("No solution exists");
        return false;
    }

    printSolution(solution);
    return true;
}

// Utility function to solve the maze using backtracking
private static boolean solveMazeUtil(int[][] maze, int x, int y, int[][] solution) {
    // If x, y is the goal, mark it as part of the solution path
    if (x == N - 1 && y == N - 1 && maze[x][y] == 1) {
        solution[x][y] = 1;
        return true;
    }

    // Check if maze[x][y] is a valid move
    if (isSafe(maze, x, y)) {
        // Mark x, y as part of the solution path
        solution[x][y] = 1;

        // Move forward in x direction
        if (solveMazeUtil(maze, x + 1, y, solution)) {
            return true;
        }

        // If moving in x direction doesn't give solution then move down in y direction
        if (solveMazeUtil(maze, x, y + 1, solution)) {
            return true;
        }
    }
}

```

```

        // If none of the above movements work then backtrack
        solution[x][y] = 0;
        return false;
    }

    return false;
}

public static void main(String[] args) {
    int[][] maze = {
        {1, 0, 0, 0, 0, 0},
        {1, 1, 0, 1, 1, 0},
        {0, 1, 0, 1, 0, 0},
        {1, 1, 1, 1, 0, 1},
        {1, 0, 0, 1, 1, 1},
        {1, 1, 1, 0, 0, 1}
    };

    solveMaze(maze);
}
}

```

### In this implementation:

- The solveMaze method initializes a solution matrix and calls the utility method solveMazeUtil to solve the maze.
- The solveMazeUtil method uses backtracking to explore all possible paths from the start (0, 0) to the goal (5, 5).
- The isSafe method checks if a cell is valid and can be included in the path.
- The printSolution method prints the solution path.

If a path is found, it prints the solution matrix with the path marked by `1`s. If no path is found, it prints "No solution exists".