Maze Solver using Iterative Deepening Search (IDS) with Binary Search Optimization

1. Introduction

This project presents a professional C++ implementation of a maze-solving algorithm based on Iterative Deepening Search (IDS), enhanced with a Binary Search strategy to optimize the depth selection process. The objective is to efficiently find a valid path from a fixed starting cell (0,0) to a user-defined goal position in a two-dimensional maze.

2. Maze Representation

The maze is modeled as a two-dimensional grid of size $N \times M$, where each cell represents either a traversable path or an obstacle:

- **0**: Empty cell (traversable)
- **1**: Wall or obstacle (non-traversable)

3. Algorithm Overview

The solution combines Depth-Limited Search (DLS) with Binary Search to efficiently determine the minimum depth required to reach the goal.

3.1 Depth-Limited Search (DLS)

Depth-Limited Search is a constrained variant of Depth-First Search (DFS). It explores the maze recursively while enforcing a strict upper bound on the maximum search depth. To ensure correctness and prevent infinite loops:

- A *visited* matrix is used to avoid revisiting cells.
- Backtracking is applied to construct the final path once the goal is reached.
- Recursive calls terminate immediately when the depth limit is exceeded.

3.2 Binary Search over Depth Limits

Traditional IDS increases the depth limit incrementally (1, 2, 3, ...). In contrast, this implementation applies Binary Search over the depth range $[1, N \times M]$ to significantly reduce the number of DLS executions.

4. Monotonic Function Justification

The use of Binary Search is mathematically justified by the monotonic nature of the reachability condition in maze traversal.

If a path to the goal exists within a depth limit D, then the same path (or another valid path) must also exist for any depth limit greater than D. Conversely, if no path exists within D steps, it is impossible for any smaller limit.

This behavior forms a step-like boolean function that transitions from False to True exactly once, making it suitable for Binary Search.

5. Complexity Analysis

Time Complexity

A single Depth-Limited Search visits each cell at most once, resulting in a time complexity of $O(N \times M)$. Binary Search performs $\log(N \times M)$ iterations. Therefore, the total time complexity is:

$$O((N \times M) \cdot \log(N \times M))$$

Space Complexity

The algorithm requires $O(N \times M)$ space for the visited matrix, recursion stack, and path storage. Hence, the overall space complexity is $O(N \times M)$.

6. Limitations

While the Binary Search optimized IDS significantly reduces search overhead, it does not guarantee the shortest possible path in all cases, unlike Breadth-First Search (BFS). Additionally, recursive DLS may face stack depth limitations for extremely large mazes.

7. Conclusion

This project demonstrates how algorithmic properties such as monotonicity can be leveraged to optimize classical search techniques. The integration of Binary Search with IDS results in a scalable and efficient maze-solving strategy suitable for academic and competitive programming applications.