# Maze Path Finder — DAA Lab Project

**Course:** Design and Analysis of Algorithms (III Semester)

**Course Code:** 24CS01PR0302

**Domain:** Algorithm Design — Backtracking / Pathfinding

**Abstract:** This mini project demonstrates a maze path-finding solution implemented in C using a recursive backtracking algorithm. The program reads a maze from a text file (30×30), where '1' denotes an open cell and '0' denotes a blocked cell. The algorithm explores possible routes from the start (top-left) to the destination (bottom-right), marking visited cells and backtracking when necessary. A web-based visualizer complements the implementation to animate the solving process and to highlight shortest paths using BFS where applicable.

| | |
|---|---|
| Team Member | Rishi Gupta |
| Tools & Languages | C (GCC), HTML/CSS/JavaScript (Visualizer) |
| Files Submitted | maze.c, input.txt, visualize/ (index.html, style.cs |
| Report Length | 5 pages (2 blank for covering details) |

## Methodology & Algorithm

**Problem Statement:** Given a grid-based maze, determine if there exists a path from the top-left cell to the bottom-right cell without passing through blocked cells. The solution reads maze data from a file and outputs visited cells and path existence. **Approach:** Implement depth-first search (DFS) based backtracking in C. Key steps: • Store maze in a 2D array (1=open, 0=wall). • Maintain a visited[][] array to avoid revisiting cells. • Recursively attempt moves in the order: Down, Right, Up, Left. • Backtrack when a branch fails and continue until the destination is found or all options are exhausted.

## Pseudocode (simplified):

```
solve(x, y):
  if (x,y) is destination and open: mark visited and return true
  if (x,y) is safe: mark visited
    for each direction in [down, right, up, left]:
      if solve(next) return true
    unmark visited (backtrack)
  return false
```

## Time & Space Complexity

**Time Complexity:** Worst-case can be exponential when many cells are open and many branches exist. In practical terms for moderate mazes the running time typically behaves around O(n^2) for an n×n grid. **Space Complexity:** O(n^2) due to the visited array and recursion stack worst-case depth.

## Results & Conclusion

Running the program with the provided input.txt (30×30) yields a clear console result indicating whether a path was found. Additionally the program prints a visited-cell matrix showing the cells examined during the search. The visualizer offers a complementary animated view where BFS can be used to highlight the shortest path. **Sample Output:** Path Found! Visited Cells (1 = visited): 1 1 0 0 1 ... ...

## Future Work & Improvements

• Replace DFS with BFS (or add BFS option) to guarantee shortest paths. • Add GUI for interactive testing (allowing custom start/end and maze editing). • Export visualizer traces and include step-by-step snapshots in the report. • Optimize for very large mazes and include performance charts.

## References

1. Thomas H. Cormen et al., Introduction to Algorithms. 2. Jon Kleinberg & Éva Tardos, Algorithm Design. 3. GeeksforGeeks, TutorialsPoint for algorithm examples and implementations.