Green University of Bangladesh

Department of Computer Science and Engineering (CSE)
Semester: (Spring, Year: 2025), B.Sc. in CSE (Day)

Maze Solver using IDDFS

Course Title: Artificial Intelligence Lab
Course Code: CSE-316
Section: 221-D22

Students Details

Name	ID
Md Showaib Rahman Tanveer	221902084

Submission Date: 03/04/2025
Course Teacher's Name: Md. Sabbir Hosen Mamun

[For teachers use only: Don't write anything inside this box]

Lab Report Status	
Marks: **Comments:**	**Signature:** **Date:**

1 Introduction

Maze-solving is a fundamental problem in computer science and artificial intelligence. In this lab, we implement the Iterative Deepening Depth-First Search (IDDFS) algorithm to find a path from a start position to a target position in a given maze.

2 Objectives

- Understand the working of the IDDFS algorithm.
- Implement IDDFS in Python to solve a maze.
- Analyze the efficiency and limitations of the algorithm.

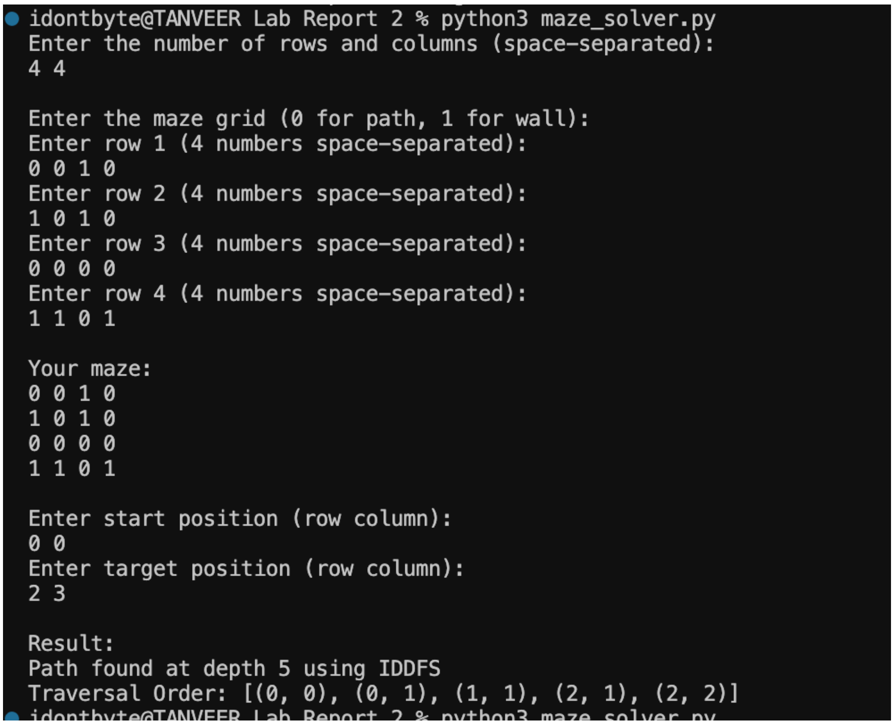
3 Procedure

1. Accept user input for the maze dimensions.
2. Construct the maze grid with 0s (paths) and 1s (walls).
3. Take user input for the start and target positions.
4. Implement IDDFS to search for a path.
5. Display the path found (if any) and the depth at which it was discovered.

4 Code

```
def is_valid(x, y, rows, cols, maze, visited):
    return (0 <= x < rows and
            0 <= y < cols and
            maze[x][y] == 0 and
            not visited[x][y])

def dfs(maze, start, target, visited, path, depth, max_depth):
    rows, cols = len(maze), len(maze[0])
    x, y = start

    if depth > max_depth:
        return False, path
    if start == target:
        return True, path

    visited[x][y] = True
    path.append(start)
    directions = [(1, 0), (0, 1), (-1, 0), (0, -1)]
    for dx, dy in directions:
        new_x, new_y = x + dx, y + dy
        if is_valid(new_x, new_y, rows, cols, maze, visited):
            found, new_path = dfs(maze, (new_x, new_y), target, visited, path.copy
            if found:
                return True, new_path

    visited[x][y] = False
    path.pop()
    return False, path

def iddfs(maze, start, target):
    rows, cols = len(maze), len(maze[0])
    max_depth = rows * cols

    for depth in range(max_depth + 1):
        visited = [[False] * cols for _ in range(rows)]
        found, path = dfs(maze, start, target, visited, [], 0, depth)
        if found:
            return True, path, depth

    return False, [], max_depth

def solve_maze():
    print("Enter the number of rows and columns (space-separated):")
    rows, cols = map(int, input().split())

    print("\nEnter the maze grid (0 for path, 1 for wall):")
    maze = []
    for i in range(rows):
        print(f"Enter row {i+1} ({cols} numbers space-separated):")
        row = list(map(int, input().split()))
        if len(row) != cols:
            print(f"Error: Row {i+1} must contain exactly {cols} numbers")
            return
        maze.append(row)

    print("\nEnter start position (row column):")
    start_x, start_y = map(int, input().split())
    print("Enter target position (row column):")
    target_x, target_y = map(int, input().split())

    start = (start_x, start_y)
    target = (target_x, target_y)
    found, path, depth = iddfs(maze, start, target)

    print("\nResult:")
    if found:
        print(f"Path found at depth {depth} using IDDFS")
        print(f"Traversal Order: {path}")
    else:
        print(f"Path not found at max depth {depth} using IDDFS")

if __name__ == "__main__":
    solve_maze()
```

5 Output



```
● idontbyte@TANVEER Lab Report 2 % python3 maze_solver.py
  Enter the number of rows and columns (space-separated):
  4 4

  Enter the maze grid (0 for path, 1 for wall):
  Enter row 1 (4 numbers space-separated):
  0 0 1 0
  Enter row 2 (4 numbers space-separated):
  1 0 1 0
  Enter row 3 (4 numbers space-separated):
  0 0 0 0
  Enter row 4 (4 numbers space-separated):
  1 1 0 1

  Your maze:
  0 0 1 0
  1 0 1 0
  0 0 0 0
  1 1 0 1

  Enter start position (row column):
  0 0
  Enter target position (row column):
  2 3

  Result:
  Path found at depth 5 using IDDFS
  Traversal Order: [(0, 0), (0, 1), (1, 1), (2, 1), (2, 2)]
● idontbyte@TANVEER Lab Report 2 % python3 maze_solver.py
```

Figure 1: Example Output of the IDDFS Maze Solver

6 Conclusion

The IDDFS algorithm successfully finds a path from the start to the target in a maze, if one exists, while limiting the depth of search iteratively. It is memory-efficient compared to BFS but can be computationally expensive in deep search trees. The results validate its effectiveness in solving maze problems with a well-defined search space.

7 GitHub Repository

The complete code and additional details can be found in the following repository:
https://github.com/idontbyte69/Academic/tree/master/AI%20LAB%20CSE%20316/Lab%20Report/Lab%20Report%202