Green University of Bangladesh
Department of Computer Science and Engineering (CSE)
Faculty of Sciences and Engineering
Semester: (Fall, Year:2025), B.Sc. in CSE (Day)

Lab Report NO: 1
Course Title: Artificial Intelligence Lab
Course Code: CSE 316 Section: 223-D4

Lab Experiment Name: You are given a 2D grid representing a maze, where each cell is either an empty space (0) or a wall (1). Your task is to implement a Python program that uses Iterative Deepening Depth-First Search (IDDFS) to determine whether a valid path exists from a given start cell to a specified target cell. You may move up, down, left, or right to adjacent empty cells, but you cannot pass through walls, and each cell may be visited only once during a single path exploration.

Student Details

	Name	ID
1.	Ruhul Amin	223002097

Lab Date :
Submission Date : 29-12-25
Course Teacher's Name : Sabbir Hosen Mamun

Lab Report Status

Marks: ..... Signature:.....
Comments:..... Date:.....

1. TITLE OF THE LAB REPORT EXPERIMENT:

You are given a 2D grid representing a maze, where each cell is either an empty space (0) or a wall (1). Your task is to implement a Python program that uses Iterative Deepening Depth-First Search (IDDFS) to determine whether a valid path exists from a given start cell to a specified target cell. You may move up, down, left, or right to adjacent empty cells, but you cannot pass through walls, and each cell may be visited only once during a single path exploration.

2. OBJECTIVES:

The objectives of this program are:

1. To represent a maze using a 2D grid where
   - 0 = empty cell (walkable)
   - 1 = wall (blocked)
2. To implement **Iterative Deepening Depth-First Search (IDDFS)**.
3. To determine whether a valid path exists from a **start cell** to a **target cell**.
4. To ensure movement is limited to **up, down, left, and right** directions only.
5. To prevent revisiting the same cell within a single search path.
6. To combine the **memory efficiency of DFS** with the **completeness of BFS**.
7.

3. PROCEDURE:

a. Represent the maze using a 2D grid (0 = empty, 1 = wall).
b. Define the start and target cells.
c. Set a maximum search depth.
d. Apply Iterative Deepening by increasing the depth limit step by step.
e. At each depth, perform Depth-Limited DFS exploring up, down, left, and right.
f. Avoid revisiting cells during the same path.
g. Stop when the target is found or the maximum depth is reached.

4. IMPLEMENTATION AND OUTPUT:

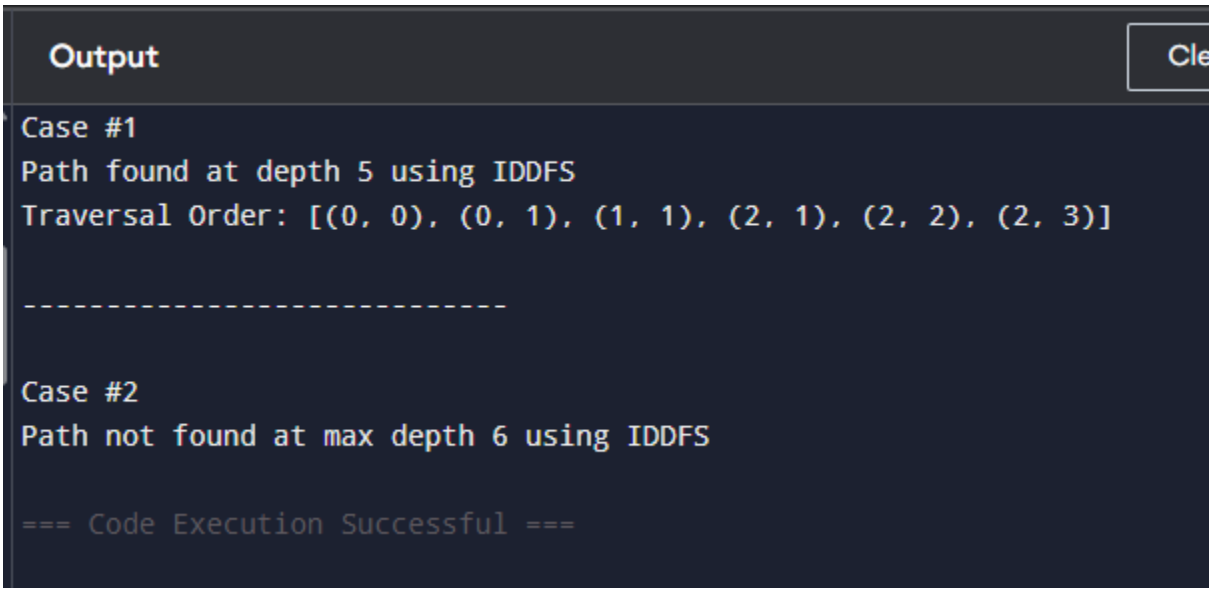
CODE:

```
# ---------- IDDFS for Maze Path Finding ----------

def is_valid(x, y, maze, visited):
    rows, cols = len(maze), len(maze[0])
    return (
        0 <= x < rows and
```

```
        0 <= y < cols and
        maze[x][y] == 0 and
        (x, y) not in visited
    )


def dfs_limited(maze, current, target, limit, visited, path):
    path.append(current)

    # Goal check
    if current == target:
        return True

    # Depth limit reached
    if limit == 0:
        path.pop()
        return False

    visited.add(current)
    x, y = current

    # Move order: Right, Down, Left, Up
    directions = [(0, 1), (1, 0), (0, -1), (-1, 0)]

    for dx, dy in directions:
        nx, ny = x + dx, y + dy
        if is_valid(nx, ny, maze, visited):
            if dfs_limited(maze, (nx, ny), target, limit - 1, visited, path):
                return True

    visited.remove(current)
    path.pop()
    return False


def iddfs(maze, start, target, max_depth):
    for depth in range(max_depth + 1):
        visited = set()
        path = []

        if dfs_limited(maze, start, target, depth, visited, path):
            print(f"Path found at depth {depth} using IDDFS")
            print(f"Traversal Order: {path}")
```

```
            return

    print(f"Path not found at max depth {max_depth} using IDDFS")


# ---------------- TEST CASE 1 ----------------
print("Case #1")

maze1 = [
    [0, 0, 1, 0],
    [1, 0, 1, 0],
    [0, 0, 0, 0],
    [1, 1, 0, 1]
]

start1 = (0, 0)
target1 = (2, 3)

iddfs(maze1, start1, target1, 6)

print("\n----------------------------\n")

# ---------------- TEST CASE 2 ----------------
print("Case #2")

maze2 = [
    [0, 1, 0],
    [0, 1, 0],
    [0, 1, 0]
]

start2 = (0, 0)
target2 = (2, 2)

iddfs(maze2, start2, target2, 6)
```

OUTPUT:



5. ANALYSIS AND DISCUSSION

In this experiment, **Iterative Deepening Depth-First Search (IDDFS)** was applied to a 2D maze to determine whether a valid path exists between a given start cell and a target cell. The maze was represented as a grid where empty cells were traversable and walls were restricted.

The algorithm performed repeated **depth-limited DFS** searches, starting from depth zero and gradually increasing the depth limit. This approach ensured that shallow solutions were explored first, similar to **Breadth-First Search (BFS)**, while maintaining the low memory usage of **Depth-First Search (DFS)**. As a result, the algorithm successfully found a valid path when one existed within the specified maximum depth.

During execution, IDDFS avoided revisiting the same cell within a single path by maintaining a visited set, which prevented infinite loops and redundant exploration. The restriction of movement to only four directions (up, down, left, right) made the search space well-defined and suitable for grid-based pathfinding.

The results show that IDDFS is effective for maze-solving problems where the depth of the solution is unknown. However, repeated DFS executions increase time complexity compared to BFS. Despite this, IDDFS remains a practical choice when

memory efficiency is a priority.

Overall, the experiment demonstrates that IDDFS provides a balanced solution by combining the completeness of BFS with the space efficiency of DFS, making it suitable for constrained search problems such as maze navigation.

6. SUMMARY

This work implemented **Iterative Deepening Depth-First Search (IDDFS)** to solve a 2D maze pathfinding problem. The maze was modeled as a grid with empty cells and walls, allowing movement in four directions only. By gradually increasing the depth limit and applying depth-limited DFS at each step, the algorithm efficiently determined whether a valid path exists from the start cell to the target cell. The approach successfully combined the low memory usage of DFS with the completeness of BFS, making IDDFS an effective and reliable method for maze navigation problems.

Github link: https://github.com/ruhulaminn1316/A-Artificial-Intelligence-Lab