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

Lab Report NO #2
Course Title: Artificial Intelligent
Course Code: CSE 316 Section: 221 D12

Lab Experiment Name: IDDFS

Student Details

	Name	ID
1.	Al Ekram Hossain	221002535

Submission Date : 09 MAY 2025
Course Teacher's Name : Md. Sabbir Hosen Mamun

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

Title : IDDFS

Objectives :

1. Maze Representation:
   - The maze is a 2D grid of integers.
   - 0 represents an empty cell that can be traversed.
   - 1 represents a wall that cannot be traversed.
2. Movement Rules:
   - You can move in four directions: up, down, left, and right.
   - You cannot revisit a cell in the same path exploration.
   - You cannot pass through walls.
3. Search Algorithm:
   - Use Iterative Deepening Depth-First Search (IDDFS):
     - A hybrid of DFS and BFS.
     - It performs depth-limited DFS repeatedly, increasing the depth limit with each iteration until the target is found or the maximum depth is reached.

Problem Analysis :

IDDFS is a combination of:

- Depth-First Search (DFS) for memory efficiency.
- Breadth-First Search (BFS) in its iterative deepening approach (increasing depth limit).

For each depth limit d, the algorithm:

- Performs a DFS up to depth d.
- If the target is not found, it increases the depth and repeats.

Complexity Analysis

Aspect	Complexity
Time (worst-case)	`O(b^d)` where `b` is the branching factor and `d` is the depth of the target node. Repeated exploration makes it slower than BFS/DFS for small graphs.
Space	`O(d)` because it uses DFS and stores only the current path, not the full tree like BFS.
Optimality	Not optimal — may not return the shortest path unless modified.
Completeness	Yes — it will find a path if one exists.

.

Implementation :

```
def is_valid_move(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 not visited[x][y]
def dfs_limited(maze, x, y, target, depth_limit, visited, path, traversal):
    if (x, y) == target:
        path.append((x, y))
        traversal.append((x, y))
        return True
    if depth_limit <= 0:
        return False

    visited[x][y] = True
    traversal.append((x, y))

    for dx, dy in [(-1,0), (1,0), (0,-1), (0,1)]:
        nx, ny = x + dx, y + dy
        if is_valid_move(nx, ny, maze, visited):
            if dfs_limited(maze, nx, ny, target, depth_limit - 1, visited, path, traversal):
                path.append((x, y))
                return True

    return False
```

```python
def iddfs(maze, start, target):
    max_depth = len(maze) * len(maze[0])
    for depth in range(max_depth):
        visited = [[False for _ in range(len(maze[0]))] for _ in range(len(maze))]
        path = []
        traversal = []
        if dfs_limited(maze, start[0], start[1], target, depth, visited, path, traversal):
            print(f"Path found at depth {depth} using IDDFS")
            print("Traversal Order:", traversal)
            return
    print("No path found using IDDFS")

# ---- Input Parsing ----
def parse_input():
    rows, cols = map(int, input().split())
    maze = []
    for _ in range(rows):
        maze.append(list(map(int, input().split())))
    start_x, start_y = map(int, input().split()[1:])
    target_x, target_y = map(int, input().split()[1:])
    return maze, (start_x, start_y), (target_x, target_y)
maze, start, target = parse_input()
iddfs(maze, start, target)
```

Output :

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS E:\codes> & C:/Users/Admin/AppData/Local/Microsoft/WindowsApps/python3.12.exe e:/codes/lab02.py
4 4
0 0 1 0
1 0 1 0
0 0 0 0
1 1 0 1
Start: 0 0
Target: 2 3
Path found at depth 5 using IDDFS
Traversal Order: [(0, 0), (0, 1), (1, 1), (2, 1), (2, 0), (2, 2), (2, 3)]
PS E:\codes>
```

Conclusion :

The problem effectively demonstrates the application of **Iterative Deepening Depth-First Search (IDDFS)** to solve a maze pathfinding challenge. By combining the space-efficiency of DFS and the completeness of BFS, IDDFS ensures that a solution, if it exists, is eventually found. The approach is particularly useful for large search spaces with unknown solution depth. This method avoids memory-intensive structures like queues used in BFS and avoids infinite loops common in basic DFS. It explores nodes progressively by depth, ensuring thorough and organized traversal. Overall, IDDFS is a powerful technique for problems requiring exhaustive yet memory-efficient search strategies.

.