Green University of Bangladesh
Computer Science and Engineering (CSE)

Faculty of Sciences and Engineering

Semester:Spring Year:2025

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

Student Details

Name	ID
Md Moinul Hasan	221902281

Submission Date: 06/04/2025

Course Teacher's Name: Md.Sabbir Hosen Mamun

Project Proposal Status

Marks: ..... Signature: .....

Comments: ..... Date: .....

Objective:

The primary objective of this project is to design, implement, and test the **Iterative Deepening Depth-First Search (IDDFS)** algorithm to effectively solve the problem of pathfinding within a **grid-based maze environment**. A maze is represented as a two-dimensional array where each cell can either be traversable (represented by 0) or blocked (represented by 1). The challenge lies in determining whether there exists a valid path from a predefined **starting position** to a specified **target position**, navigating only through valid, non-blocked cells.

This project aims to apply IDDFS as a strategic approach that blends the benefits of both Depth-First Search (DFS) and Breadth-First Search (BFS). While DFS is memory-efficient and can explore deeply into the graph or grid, it may miss the shortest path or get stuck in deep branches. BFS, on the other hand, guarantees the shortest path but can be memory-intensive due to storing all frontier nodes. IDDFS overcomes these limitations by performing a series of depth-limited DFS iterations, incrementally increasing the allowed search depth until the target is found or a defined maximum depth is reached.

By implementing IDDFS, the project focuses on:

- Understanding the theoretical foundation of recursive and iterative search strategies.
- Applying algorithmic logic to real-world-like grid-based problems.
- Testing the algorithm across multiple test cases to ensure robustness, correctness, and edge case handling.
- Analyzing the advantages and trade-offs of IDDFS in comparison to conventional search algorithms.

This project not only strengthens algorithm design and programming skills but also provides insight into how classical search techniques can be optimized for constrained environments like mazes, robotics navigation, and game AI.

Theory:

Iterative Deepening Depth-First Search (IDDFS) is a search algorithm that effectively combines the space-efficiency of Depth-First Search (DFS) with the completeness of Breadth-First Search (BFS). The core concept behind IDDFS is that it repeatedly performs depth-limited DFS, where the depth limit starts at 1 and is incrementally increased with each iteration. This process continues until the target is found or a predefined maximum depth is reached.

Unlike pure DFS, which can potentially get stuck in deep, infinite paths without finding the goal, IDDFS guarantees that the algorithm will explore all possible paths within the search space. It does so by ensuring that it examines paths in a manner similar to BFS, exploring all nodes at each depth level before moving deeper into the search tree. However, the key difference is that IDDFS maintains the space efficiency of DFS by never storing more than a single path in memory at a time. This is particularly advantageous when dealing with large search spaces, as it avoids the need for large memory allocations, a common drawback of BFS.

By incrementing the depth limit in each iteration, IDDFS ensures that the algorithm explores progressively deeper levels of the search tree, eventually reaching the target if it exists. The iterative approach allows the algorithm to find the shortest path in terms of depth, like BFS, while still maintaining a low memory footprint. IDDFS is particularly well-suited for problems where the depth of the goal is unknown or where the search space is large, making it a powerful tool in many applications, including pathfinding, decision-making, and problem-solving scenarios.

Code:

```
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 not visited[x][y]


def dls(maze, x, y, target, depth, visited, path, traversal_order):
    if depth < 0:
        return False
    visited[x][y] = True
    traversal_order.append((x, y))
    path.append((x, y))
    if (x, y) == target:
        return True
    directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]
    for dx, dy in directions:
        nx, ny = x + dx, y + dy
        if is_valid(nx, ny, maze, visited):
```

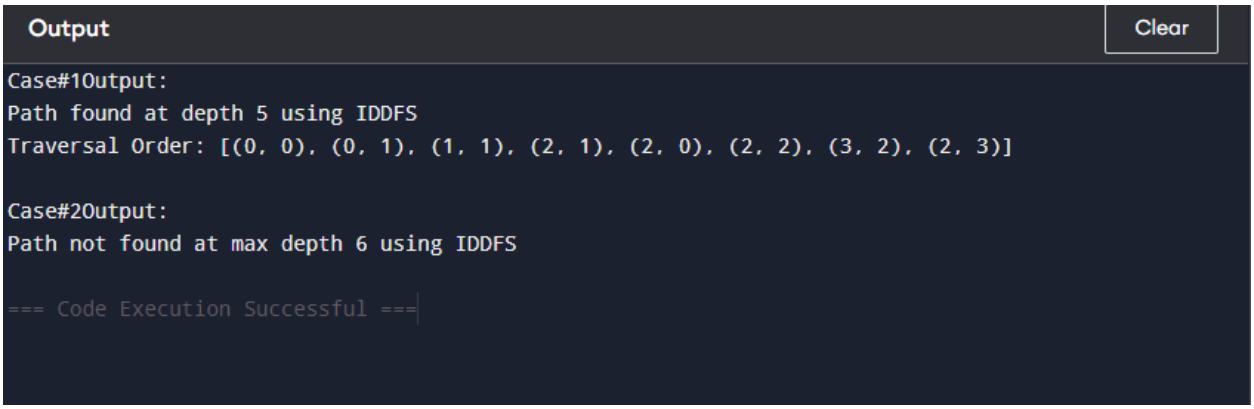
```
            if dls(maze, nx, ny, target, depth - 1, visited, path, traversal_order):
                return True
    path.pop()
    return False

def iddfs(maze, start, target, max_depth):
    for depth in range(max_depth + 1):
        visited = [[False for _ in range(len(maze[0]))] for _ in range(len(maze))]
        path = []
        traversal_order = []
        if dls(maze, start[0], start[1], target, depth, visited, path, traversal_order):
            print(f"Path found at depth {depth} using IDDFS")
            print("Traversal Order:", traversal_order)
            return
    print(f"Path not found at max depth {max_depth} using IDDFS")

maze1 = [
    [0, 0, 1, 0],
    [1, 0, 1, 0],
    [0, 0, 0, 0],
    [1, 1, 0, 1]
]
start1 = (0, 0)
target1 = (2, 3)
print("Case#1Output:")
iddfs(maze1, start1, target1, 10)

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

```
    [0, 1, 0],
    [0, 1, 0]
]
start2 = (0, 0)
target2 = (2, 2)
print("\nCase#2Output:")
iddfs(maze2, start2, target2, 6)
```



Conclusion:

Through this lab, I gained a deeper understanding of how Iterative Deepening Depth-First Search (IDDFS) operates by combining the strengths of both Depth-First Search (DFS) and Breadth-First Search (BFS). IDDFS explores the grid in a layered fashion, incrementally increasing the depth limit for each iteration, ensuring that it eventually explores all possible paths up to the maximum depth. This approach allows it to guarantee the shortest path in terms of depth, similar to BFS, but with a significantly lower memory usage, much like DFS.

One of the key advantages of IDDFS over pure DFS is its ability to avoid getting stuck in deep or infinite paths, which is a common limitation of DFS when dealing with large or complex search spaces. Additionally, IDDFS is particularly effective for problems where the depth of the goal is unknown, making it a valuable algorithm in many search-based applications, such as

pathfinding or decision-making tasks. By leveraging the best aspects of both DFS and BFS, IDDFS offers a balanced approach that ensures optimal performance without the high memory requirements associated with BFS, making it an ideal choice for certain types of problems.