



## *Green University of Bangladesh*

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

---

### **Lab Report 02 - Iterative deepening depth-first search (IDDFS)**

---

*Course Title: Artificial Intelligence Lab  
Course Code: CSE-316  
Section: 221 D7*

#### Student Details

<b>Name</b>	<b>ID</b>
Sabbir Ahmed	221902129

*Lab Date: 23-03-2025  
Submission Date: 02-04-2025  
Course Teacher's Name: Md. Sabbir Hosen Mamun*

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

<u>Lab Project Status</u>	
<b>Marks:</b>	<b>Signature:</b>
<b>Comments:</b>	<b>Date:</b>

# Contents

1	Problem Title . . . . .	2
2	Objective(s) . . . . .	2
3	Algorithm Implementation . . . . .	2
3.1	Depth-Limited Search (DLS) . . . . .	2
3.2	Iterative Deepening Depth-First Search (IDDFS) . . . . .	3
4	Python Implementation . . . . .	4
5	Input: . . . . .	6
5.1	Test Case 1 . . . . .	6
5.2	Test Case 2 . . . . .	6
6	Discussion . . . . .	7
7	Conclusion . . . . .	7

# 1 Problem Title

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 Objective(s)

The key objectives are:

- **Learning Python with a problem:** Build a pathfinding algorithm using Iterative Deepening Depth-First Search (IDDFS) to improve your Python skills.
- **Efficient Search Method:** IDDFS combines the memory efficiency of depth-first search with the thoroughness of breadth-first search by gradually increasing the depth limit until the destination is found.
- **Practical Application:** Use this algorithm to find a path from a starting point to a destination in a graph or grid, applying key concepts from class and practice exercises.

## 3 Algorithm Implementation

Here we describe the algorithms used for finding a path in the maze.

### 3.1 Depth-Limited Search (DLS)

The Depth-Limited Search explores the maze depth-first up to a specified limit.

---

**Algorithm 1:** Depth-Limited Search (DLS)

---

**Input :** maze grid *maze*,  
current position *current* (x, y),  
target position *target* (x, y),  
current search depth *depth*,  
maximum search depth *limit*,  
set of visited locations *visited*,  
list representing the current path *path*  
**Output :** Boolean indicating if target was found (*found*),  
The path list if found (*result\_path*), otherwise *Null*

```
1 Function DepthLimitedSearch(maze, current, target, depth, limit, visited, path):
2   Add current to visited;
3   Append current to path;
4   if current = target then
5     return True, path;
6     // Target found
7   if depth = limit then
8     Remove last element from path;
9     // Backtrack: Hit depth limit
10    return False, Null;
11  Define possible moves (e.g., up, down, left, right);
12  Get maze dimensions rows, cols;
13  for each move (dx, dy) in moves do
14    Calculate neighbor position (nx, ny) = (current.x + dx, current.y + dy);
15    if neighbor is within maze bounds ( $0 \leq nx < rows$ ,  $0 \leq ny < cols$ ) AND
16      maze[nx][ny] is traversable (e.g., 0) AND neighbor is NOT IN visited then
17        // Explore valid, unvisited neighbor
18        found, result_path = DepthLimitedSearch(maze, neighbor, target,
19          depth + 1, limit, visited, path);
20        if found then
21          return True, result_path;
22          // Pass success upwards
23  Remove last element from path;
24  // Backtrack: No path found from this node within limit
25  return False, Null;
```

---

### 3.2 Iterative Deepening Depth-First Search (IDDFS)

The Iterative Deepening DFS uses DLS repeatedly with incrementally increasing depth limits to find the shortest path in terms of depth.

---

**Algorithm 2:** Iterative Deepening Depth-First Search (IDDFS)

---

**Input :** maze grid *maze*,

start position *start* (*x*, *y*),

target position *target* (*x*, *y*),

maximum possible search depth *max\_depth*

**Output :** Boolean indicating if target was found (*found*),

Depth of the found path (*found\_depth*) or *max\_depth* if not found,

The path list if found (*result\_path*), otherwise *Null*

```
1 Function IterativeDeepeningDFS(maze, start, target, max_depth):
2   for limit from 0 to max_depth do
3     Create an empty set visited;
4     Create an empty list path;
5     // Perform DLS with the current depth limit
6     found, result_path = DepthLimitedSearch(maze, start, target, 0, limit,
7       visited, path);
8     if found then
9       found_depth = length(result_path) - 1;
10      // Calculate number of steps
11      return True, found_depth, result_path;
12      // Shortest path found
13    // Target not found within max_depth
14  return False, max_depth, Null;
```

---

## 4 Python Implementation

```
1 def depth_limited_search(maze, current, target, depth, limit,
2   visited, path):
3     x, y = current
4     path.append(current)
5     visited.add(current)
6
7     if current == target:
8         return True, path
9
10    if depth == limit:
11        path.pop()
12        return False, None
13
14    moves = [(-1, 0), (1, 0), (0, -1), (0, 1)]
15    rows, cols = len(maze), len(maze[0])
16
17    for dx, dy in moves:
18        nx, ny = x + dx, y + dy
19        neighbor = (nx, ny)
20        if 0 <= nx < rows and 0 <= ny < cols and maze[nx][ny]
           == 0 and neighbor not in visited:
```

```

21         found, result_path = depth_limited_search(maze,
22             neighbor, target, depth + 1, limit, visited,
23             path)
24         if found:
25             return True, result_path
26     path.pop()
27     return False, None
28
29 def iterative_deepening_dfs(maze, start, target, max_depth):
30     for limit in range(max_depth + 1):
31         visited = set()
32         path = []
33         found, result_path = depth_limited_search(maze, start
34             , target, 0, limit, visited, path)
35         if found:
36             return True, len(result_path) - 1, result_path
37     return False, max_depth, None
38
39 if __name__ == "__main__":
40     try:
41         rows, cols = map(int, input("Enter number of rows and
42             columns: ").split())
43         maze = []
44         print("Enter the maze grid (0 for empty, 1 for wall):")
45         for _ in range(rows):
46             row = list(map(int, input().strip().split()))
47             maze.append(row)
48
49         start_input = input("Start: ").split()
50         target_input = input("Target: ").split()
51         start = (int(start_input[0]), int(start_input[1]))
52         target = (int(target_input[0]), int(target_input[1]))
53
54         max_depth = rows * cols
55
56         found, found_depth, traversal_path =
57             iterative_deepening_dfs(maze, start, target,
58             max_depth)
59
60         if found:
61             print(f"Path found at depth {found_depth} using
62                 IDDFS")
63             print("Traversal Order:", traversal_path)
64         else:
65             print(f"Path not found at max depth {max_depth}
66                 using IDDFS")
67     except ValueError:

```

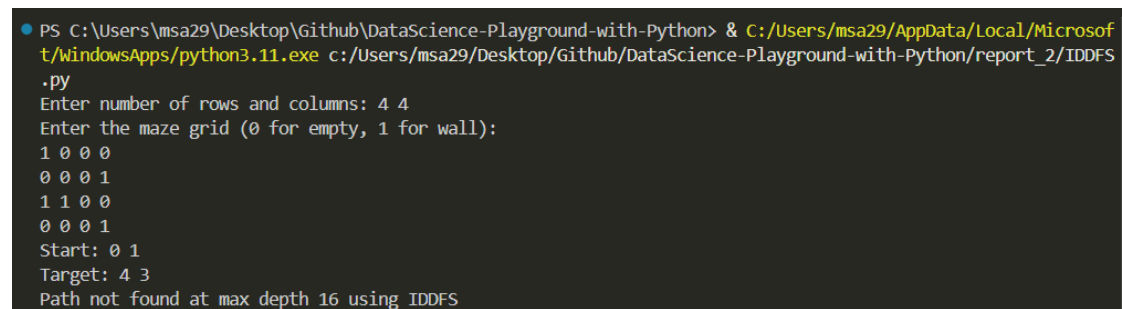
## 5 Input:

### 5.1 Test Case 1

#### Terminal

```
Enter number of rows and columns: 4 4
Enter the maze grid (0 for empty, 1 for wall):
1 0 0 0
0 0 0 1
1 1 0 0
0 0 0 1
Start: 0 1
Target: 4 3
Path not found at max depth 16 using IDDFS
```

#### Screenshot



```
PS C:\Users\msa29\Desktop\Github\DataScience-Playground-with-Python> & C:/Users/msa29/AppData/Local/Microsof
t/windowsApps/python3.11.exe c:/Users/msa29/Desktop/Github/DataScience-Playground-with-Python/report_2/IDDFS
.py
Enter number of rows and columns: 4 4
Enter the maze grid (0 for empty, 1 for wall):
1 0 0 0
0 0 0 1
1 1 0 0
0 0 0 1
Start: 0 1
Target: 4 3
Path not found at max depth 16 using IDDFS
```

Figure 1: Test Case with no path

### 5.2 Test Case 2

#### Terminal

```
Enter number of rows and columns: 4 4
Enter the maze grid (0 for empty, 1 for wall):
0 1 1 1
0 1 0 0
0 0 0 1
1 1 1 1
Start: 0 0
Target: 2 2
Path found at depth 4 using IDDFS
Traversal Order: [(0, 0), (1, 0), (2, 0), (2, 1), (2, 2)]
```

## Screenshot

```
PS C:\Users\msa29\Desktop\Github\DataScience-Playground-with-Python> & C:/Users/msa29/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/msa29/Desktop/Github/DataScience-Playground-with-Python/report_2/IDDFS.py
Enter number of rows and columns: 4 4
Enter the maze grid (0 for empty, 1 for wall):
0 1 1 1
0 1 0 0
0 0 0 1
1 1 1 1
Start: 0 0
Target: 2 2
Path found at depth 4 using IDDFS
Traversal Order: [(0, 0), (1, 0), (2, 0), (2, 1), (2, 2)]
```

Figure 2: Test Case with no path

## 6 Discussion

In this code, we have two main algorithms,

1. **depth\_limited\_search** that Performs a depth-limited DFS from the current cell. Returns a tuple (found, path) where found is True if target is reached.
2. **iterative\_deepening\_dfs** that uses iterative deepening DFS (IDDFS) to search for a path. Returns a tuple (found, found\_depth, path) where:
  - found: True if a path was found.
  - found\_depth: depth at which the target was found.
  - path: list of cells (tuples) representing the traversal order.

The IDDFS approach combines the depth-first search's space efficiency with the completeness of breadth-first search by incrementally increasing the search depth. The implementation allowed movement in four directions (up, down, left, and right) while avoiding obstacles and ensuring backtracking when necessary. The user can give input with spaces in CLI and the program will configure it automatically because we have utilized Python's built-in functions like **split()** and **strip()**, start and goal positions, and displaying traversal paths or failure messages. The program also shows the path it traverses to reach the destination as the problem requires.

## 7 Conclusion

This code successfully implemented the IDDFS algorithm for pathfinding in a grid environment. The iterative deepening approach ensured completeness while maintaining low memory usage, making it an effective alternative to traditional DFS and BFS. The code reinforced key programming concepts in Python, including recursion, iterative control structures, and set-based tracking for visited nodes.

We have implemented two important algorithms for IDDFS, one it DFS and another is DLS to make the traverse limited.we could also include diagonal movement, weighted path traversal, and real-time visualization of the search process to make it more functional. Overall, this code provided valuable insights into search algorithms, grid-based problem-solving, and algorithmic efficiency in AI applications.