



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

**Lab Report NO - 01**  
**Course Title:** Artificial intelligence Lab  
**Course Code:** CSE-316 **Section:**221\_D22

**Student Details**

Name		ID
1.	Md Moinul Hasan	221902281

**Lab Date** : 18.02.2025  
**Submission Date** : 25.02.2025  
**Course Teacher's Name** : Md. Sabbir Hosen Mamun

**Lab Report Status**

**Marks:** .....  
**Comments:**.....

**Signature:**.....  
**Date:**.....

### **1. TITLE OF THE LAB REPORT EXPERIMENT:**

2. Write a program that generates a random  $N \times N$  grid ( $N$  between 4 and 7) with non-obstacle source and goal states. It performs DFS to find a path from source to goal and prints the grid, source, goal, DFS path, and topological order of node traversal.

### **3. OBJECTIVES:**

This program is designed to simulate a pathfinding process on a grid made up of open and blocked cells. The goal is to find a way from a starting point (source) to a target point (goal) using Depth First Search (DFS). The program generates the grid randomly, marks the start and goal, and then uses DFS to find a path, if one exists. The output shows the grid with the path taken and the order in which the cells were visited.

### **4. PROCEDURE**

#### **Create the Grid:**

First, the user chooses the grid size (between 4 and 7). The grid is then randomly filled with cells that are either open (0) or blocked (1). Optionally, we can ensure there's a path from the source to the goal.

#### **Choose Source and Goal:**

The source and goal positions are chosen randomly. These two points must be different to ensure a valid path search.

#### **Pathfinding with DFS:**

The program uses DFS to explore the grid starting from the source. It keeps track of visited cells and backtracks when necessary, eventually finding the goal (if possible).

#### **Display Results:**

The program prints the grid with markers for the source, goal, and the path found. It also shows the order in which cells were visited by the DFS algorithm.

## 5. IMPLEMENTATION

```
import random

def generate_grid(grid_size, guaranteed_path=False):
    grid = [[1 for _ in range(grid_size)] for _ in range(grid_size)]

    if guaranteed_path:
        source_x, source_y = random.randint(0, grid_size-1), random.randint(0, grid_size-1)
        grid[source_x][source_y] = 0

        goal_x, goal_y = random.randint(0, grid_size-1), random.randint(0, grid_size-1)
        while grid[goal_x][goal_y] == 0:
            goal_x, goal_y = random.randint(0, grid_size-1), random.randint(0, grid_size-1)
        grid[goal_x][goal_y] = 0

        for i in range(grid_size):
            for j in range(grid_size):
                if random.random() < 0.5:
                    grid[i][j] = 0
    else:
        grid = [[random.choice([0, 0, 0, 1]) for _ in range(grid_size)] for _ in range(grid_size)]

    return grid

def find_valid_position(grid):
    grid_size = len(grid)
    while True:
        x, y = random.randint(0, grid_size-1), random.randint(0, grid_size-1)
        if grid[x][y] == 0:
            return (x, y)

def dfs(grid, start, goal):
    grid_size = len(grid)
    stack = [start]
    visited = set()
    parent = {}
    topological_order = []

    while stack:
        current_node = stack.pop()

        if current_node in visited:
            continue

        visited.add(current_node)
        topological_order.append(current_node)

        if current_node == goal:
            break
```

```

x, y = current_node
neighbors = [(x-1, y), (x+1, y), (x, y-1), (x, y+1)]
random.shuffle(neighbors)

for nx, ny in neighbors:
    if 0 <= nx < grid_size and 0 <= ny < grid_size and grid[nx][ny] == 0 and (nx, ny) not in
visited:
        stack.append((nx, ny))
        parent[(nx, ny)] = current_node

path = []
if goal in visited:
    current = goal
    while current != start:
        path.append(current)
        current = parent[current]
    path.append(start)
    path.reverse()

return path, topological_order

def print_grid(grid, source, goal, path):
    grid_size = len(grid)
    for i in range(grid_size):
        for j in range(grid_size):
            if (i, j) == source:
                print("S", end=" ")
            elif (i, j) == goal:
                print("G", end=" ")
            elif (i, j) in path:
                print(".", end=" ")
            else:
                print("1", end=" ")
        print()

if __name__ == "__main__":
    print("Welcome to the Grid Path Finder!")

    while True:
        try:
            grid_size = int(input("Please enter grid size (between 4 and 7): "))
            if 4 <= grid_size <= 7:
                break
            else:
                print("Invalid input. The grid size should be between 4 and 7.")
        except ValueError:
            print("Invalid input. Please enter a valid integer between 4 and 7.")

    path_choice = input("Do you want a guaranteed path between source and goal? (yes/no): ")
    path_choice = path_choice.lower()

```

```

guaranteed_path = path_choice == "yes"

grid = generate_grid(grid_size, guaranteed_path)

source = find_valid_position(grid)
goal = find_valid_position(grid)

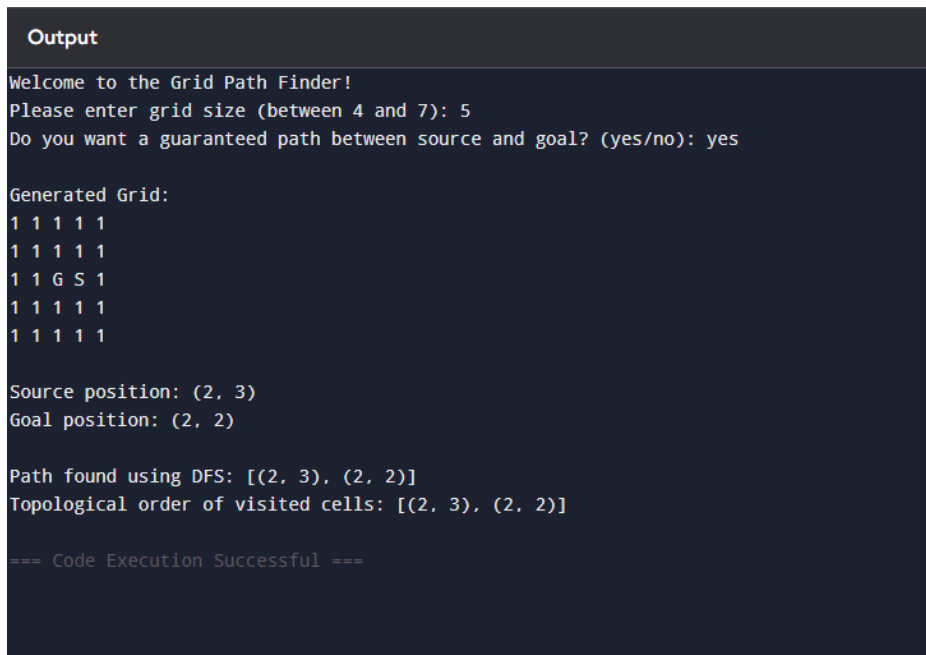
while goal == source:
    goal = find_valid_position(grid)

path, topological_order = dfs(grid, source, goal)

print("\nGenerated Grid:")
print_grid(grid, source, goal, path)
print(f"\nSource position: {source}")
print(f"Goal position: {goal}")
print(f"\nPath found using DFS: {path if path else 'No path found'}")
print(f"Topological order of visited cells: {topological_order}")

```

## **5.TEST RESULT / OUTPUT**



```

Output
Welcome to the Grid Path Finder!
Please enter grid size (between 4 and 7): 5
Do you want a guaranteed path between source and goal? (yes/no): yes

Generated Grid:
1 1 1 1 1
1 1 1 1 1
1 1 G S 1
1 1 1 1 1
1 1 1 1 1

Source position: (2, 3)
Goal position: (2, 2)

Path found using DFS: [(2, 3), (2, 2)]
Topological order of visited cells: [(2, 3), (2, 2)]

=== Code Execution Successful ===

```

Figure 1:Output

## **6. ANALYSIS AND DISCUSSION**

**Algorithm Efficiency:**

## **Efficiency of DFS:**

DFS works by exploring as deep as possible along one path before backtracking, which can sometimes make it less efficient, especially when a quick path is hidden behind many blocked cells. However, it guarantees finding a path if one exists, but it doesn't always find the shortest path.

## **Grid Representation:**

The grid is shown with S for the start, G for the goal, . for the path, and 1 for blocked cells, making it easy to visualize how the algorithm works.

## **Random Grid Generation:**

Since the grid is randomly generated, each run can produce different results. In some cases, the DFS algorithm may struggle to find a path if there are many blocked cells

## **Discussion:**

### **User Control:**

The program lets the user choose the grid size and whether they want a guaranteed path between the start and goal, giving them flexibility in how the grid is set up.

### **Challenges:**

DFS is simple, but it's not always the most efficient algorithm. It might not find the shortest path, especially in large or complex grids. It can also get stuck if there are many blocked cells, making the search slower.

### **Room for Improvement:**

A better option for finding the shortest path might be a different algorithm like Breadth First Search (BFS) or A\*. These would ensure the shortest path is found more efficiently

## **Summary:**

This program simulates the process of finding a path in a grid using DFS. The user can control the grid size and whether a guaranteed path exists. The program then uses DFS to explore the grid, displaying the path found and the order in which the cells were visited. While DFS works for smaller grids, it can be slow and inefficient for more complex ones, and could be improved by using faster algorithms like BFS or A\*.