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 01: Graph Traversal

Course Title: Artificial Intelligence Lab

Course Code: CSE-316 Section:221-14

Student Details

Name		ID
1.	Md.Mohibullah	221902083

Lab Date : 19-02-2025

Submission Date : 25-02-2025

Course Teacher's Name : Md. Sabbir Hosen Mamun

Lab Report Status

Marks: …… Signature:.....

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

Title: Write a program that generates a random N×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.

Objectives:

- Generate a random N×N grid (where N is between 4 and 7) with obstacles.
- Define valid source and goal nodes within the grid.
- Use Depth-First Search (DFS) to find a path from the source to the goal.
- Print the grid, source, goal, DFS path (if found), and the topological order of traversal.

Introduction:

Depth-First Search (DFS) is a fundamental graph traversal algorithm that explores as far as possible along a branch before backtracking. It is widely used in pathfinding, maze-solving, and graph analysis due to its simplicity and effectiveness in exploring connected components.

In this program, we apply DFS to a randomly generated 2D grid to determine whether a path exists between a given source node and a goal node. The grid represents a plane where some cells are blocked (obstacles), and others are valid for traversal. By implementing DFS, we aim to explore the grid systematically, identify a feasible path from the source to the goal, and analyze the traversal process.

This implementation not only demonstrates the practical application of DFS but also provides insights into how the algorithm navigates through obstacles and explores possible routes in a constrained environment. Additionally, the program outputs the topological order of traversal, offering a clear visualization of the sequence in which nodes are visited during the search.

Methodology:

The following steps outline the methodology used to implement and analyze the Depth-First Search (DFS) algorithm on a randomly generated 2D grid:

1. Grid Generation :

- A random N×N grid is created, where N is chosen between 4 and 7.
- Each cell in the grid is randomly assigned a value of 1 (valid cell) or 0 (obstacle).
- The grid serves as the environment for the DFS traversal.

2. Source and Goal Selection :

- Two distinct cells, representing the source and goal nodes, are randomly selected from the grid.
- These nodes are ensured to be valid (non-obstacle) and distinct to guarantee a feasible starting point and destination.

3. DFS Implementation :

- The DFS algorithm is implemented using an explicit stack to simulate recursion.
- Starting from the source node, the algorithm explores neighboring cells in four possible directions: up, down, left, and right.
- Each visited node is marked as "visited" by setting its value to 0 to prevent revisiting.
- If the goal node is reached, the algorithm terminates, and the path is recorded.

4. Path Reconstruction :

- The path from the source to the goal is reconstructed by storing the sequence of moves and coordinates during traversal.
- If no path exists, the program indicates that the goal is unreachable.

5. Topological Order :

- The order in which nodes are visited during the DFS traversal is recorded as the topological order.
- This provides insight into the exploration pattern of the algorithm.

6. Output :

- The program outputs:
- The generated grid, showing obstacles and valid cells.
- The coordinates of the source and goal nodes.
- The step-by-step DFS path from the source to the goal (if found).
- The topological order of node traversal.

Implementation:

Here are Python code:

python.py

```python
import random
class Node:
    def __init__(self, x, y):
        self.x = x  # x-coordinate of the node
        self.y = y  # y-coordinate of the node

class DFS:
    def __init__(self):
        self.directions = [
            (1, 0, "Down"),  # Down
            (-1, 0, "Up"),   # Up
            (0, 1, "Right"), # Right
            (0, -1, "Left")  # Left
        ]
        self.found = False  # Flag to indicate whether the goal is found
        self.N = 0  # Size of the grid
        self.source = None  # Source node
        self.goal = None  # Goal node
        self.path = []  # Path from source to goal
        self.topological_order = []  # Topological order of node traversal
```

```python

    def init(self):
        self.N = random.randint(4, 7)  # Randomly generate grid size N between 4 and 7
        graph = [[random.choice([0, 1]) for _ in range(self.N)] for _ in range(self.N)]  # Generate grid
        # Ensure source and goal nodes are valid (non-obstacle)
        while True:
            source_x, source_y = random.randint(0, self.N - 1), random.randint(0, self.N - 1)
            goal_x, goal_y = random.randint(0, self.N - 1), random.randint(0, self.N - 1)
            if graph[source_x][source_y] == 1 and graph[goal_x][goal_y] == 1 and (source_x, source_y) != (goal_x, goal_y):
                break

        self.source = Node(source_x, source_y)
        self.goal = Node(goal_x, goal_y)
        print("\nGenerated Grid:")
        for row in graph:
            print(" ".join(map(str, row)))

        print(f"\nSource: ({self.source.x}, {self.source.y})")
        print(f"Goal: ({self.goal.x}, {self.goal.y})")

        self.st_dfs(graph, self.source)

        if self.found:
            print("\nGoal found!")
            print("Path taken:")
            for move, coord in self.path:
                print(f"{move} -> ({coord[0]}, {coord[1]})")

            print("\nTopological Order of Node Traversal:")
            for node in self.topological_order:
                print(f"({node[0]}, {node[1]})", end=" -> ")
            print("END")
        else:
            print("\nGoal cannot be reached from the starting block.")

    def st_dfs(self, graph, current_node):
        stack = [(current_node, [])]  # Stack to store the current node and the path taken to reach it

        while stack:
            u, path_so_far = stack.pop()  # Pop the top node and its path

            self.topological_order.append((u.x, u.y))  # Add current node to topological order

            if u.x == self.goal.x and u.y == self.goal.y:
                self.found = True
                self.path = path_so_far + [("Goal", (u.x, u.y))]
                return

            if graph[u.x][u.y] == 0:
                continue
            graph[u.x][u.y] = 0

            for dx, dy, direction in self.directions:
                v_x, v_y = u.x + dx, u.y + dy

                if 0 <= v_x < self.N and 0 <= v_y < self.N and graph[v_x][v_y] == 1:
                    stack.append((Node(v_x, v_y), path_so_far + [(direction, (v_x, v_y))]))
if __name__ == "__main__":
    dfs = DFS()
    dfs.init()
```

Result:

```
Generated Grid:
1 0 0 1
0 1 1 1
0 1 0 1
1 1 0 0

Source: (3, 1)
Goal: (2, 3)

Goal found!
Path taken:
Up -> (2, 1)
Up -> (1, 1)
Right -> (1, 2)
Right -> (1, 3)
Down -> (2, 3)
Goal -> (2, 3)

Topological Order of Node Traversal:
(3, 1) -> (3, 0) -> (2, 1) -> (1, 1) -> (1, 2) -> (1, 3) -> (0, 3) -> (2, 3) -> END
PS E:\8th semester\AI Lab\Lab Report 01>
```

Conclusion

The implementation of the Depth-First Search (DFS) algorithm on a randomly generated 2D grid successfully demonstrated its ability to determine whether a path exists between a source and a goal node. The program effectively handled grids of varying sizes (between 4×4 and 7×7) with randomly placed obstacles, ensuring that both the source and goal nodes were valid and distinct.

Pathfinding Capability :

DFS was able to explore the grid systematically and identify a valid path from the source to the goal when one existed.The step-by-step moves and coordinates provided a clear visualization of the traversal process.

Topological Order :

The topological order of node traversal highlighted the sequence in which nodes were visited, offering insights into the depth-first nature of the algorithm.

Random Grid Generation :

The random generation of grids with obstacles introduced variability, making the program robust and adaptable to different scenarios.

Limitations :

DFS may not always find the shortest path, as it prioritizes depth over breadth. In larger grids or more complex environments, DFS could become inefficient due to its recursive nature and potential for deep exploration.

Applications :

This implementation can serve as a foundation for solving real-world problems such as maze-solving, robot navigation, and game AI pathfinding.

GitHub Link:
https://github.com/programmermahi/Artificial-Intelligence/tree/main/LabReport01-GraphTraversal