Green University of Bangladesh

Department of Computer Science and Engineering (CSE)
Faculty of Sciences and Engineering
Spring 2024, B.Sc. in CSE (DAY)

Lab Report NO # 03

Course Title: Artificial Intelligence Lab
Course Code: CSE 316 **Section: CSE 213 – D1**

Problems / Tasks / Domains:

Write a program to perform DFS traversal on a 2D plane by taking the user input of grid size N. After that generate N×N matrix and place the obstacles randomly. Now print the matrix and take user input of starting and goal state, find the path from source to destination using DFS.

Student Details

Name	ID
Md. Shahidul Islam Prodhan	**213902017**

Lab Assigned Date: 20th March 2024

Submission Date: 1st April 2024

Course Teacher's Name: Fairuz Shaiara, Lecturer

[For Teacher's use only: Don't write anything inside this box]

Lab Report Status

Marks:	**Signature:**
Comments:	**Date:**

1. TITLE OF THE LAB REPORT EXPERIMENT

Implement Depth-First Search.

2. OBJECTIVES/AIM

The primary objectives of this lab is -

- To understand how to represent states and nodes in a graph.
- To understand how Depth-First Search (DFS) works on a two-dimensional (2D) plane.

3. PROCEDURE / ANALYSIS / DESIGN

- Problem Description:

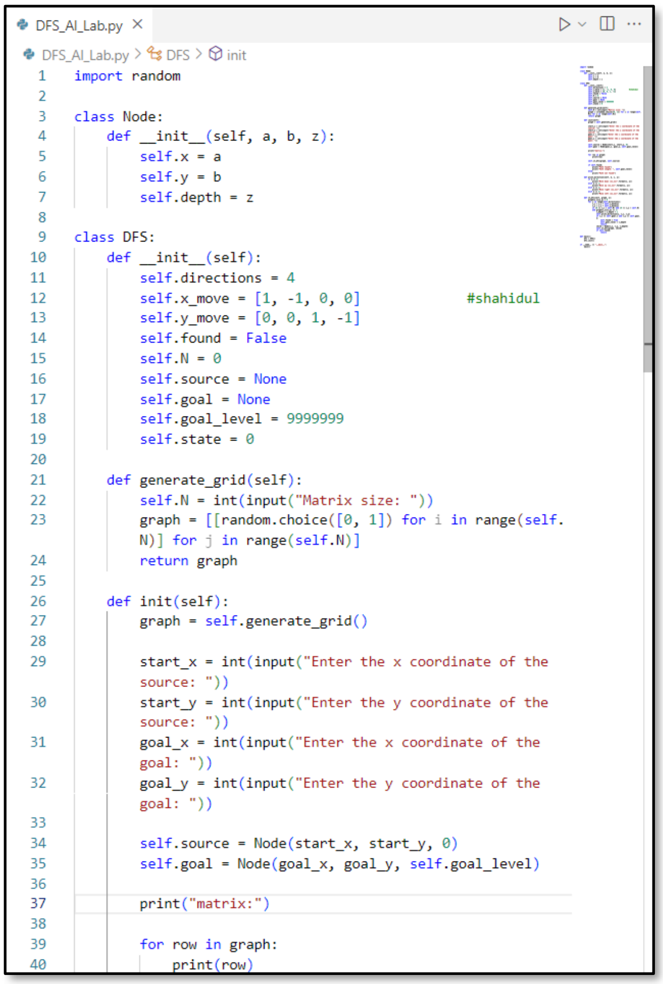
The task at hand involves implementing a Depth-First Search (DFS) algorithm to find a path from a given source node to a target node in a two-dimensional grid environment. The grid consists of cells that can either be open spaces or obstacles, where movement is only allowed through open spaces.

- Main Objectives
    - **Pathfinding**: Find a path from a specified source cell to a target cell in the grid.
    - **Grid Generation**: Randomly generate a grid with obstacles and open spaces.
    - **User Interaction**: Prompt the user to input the coordinates of the source and target cells.
    - **Output Visualization**: Display the generated grid and the path found from the source to the target.

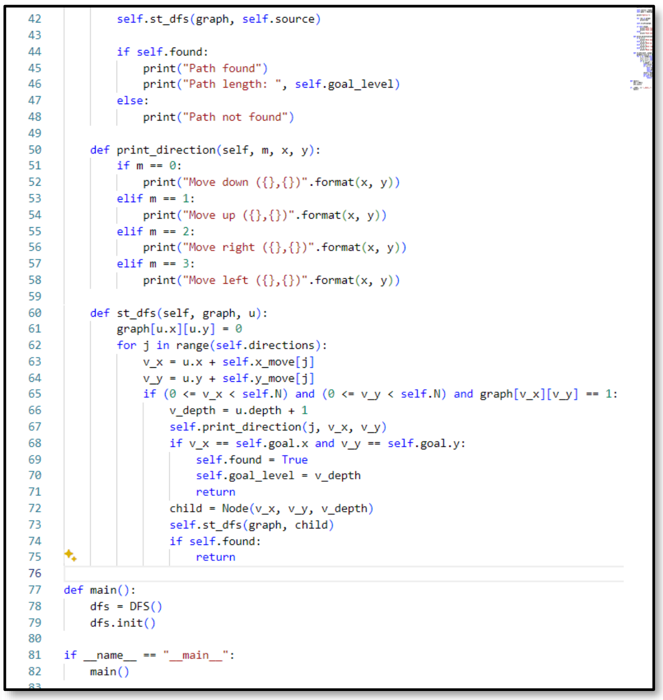
- Brief overview of the code

1. It defines a ***Node*** class to represent nodes in the grid, with attributes for x and y coordinates, and depth.
2. It defines a ***DFS*** class with methods for initializing the search, generating a random grid, printing directions, and performing the DFS search.
3. The ***generate_grid()*** method randomly generates a grid of 0s and 1s, where 0 represents an obstacle and 1 represents an open space.
4. The ***init()*** method initializes the search by taking user input for the source and goal coordinates, generates the grid, and initiates the DFS search.
5. The ***st_dfs()*** method performs the depth-first search recursively, exploring neighboring nodes in four directions (up, down, left, right) until it finds the goal or exhausts all possibilities.
6. The ***main()*** function creates an instance of the DFS class and initiates the search.

4. IMPLEMENTATION

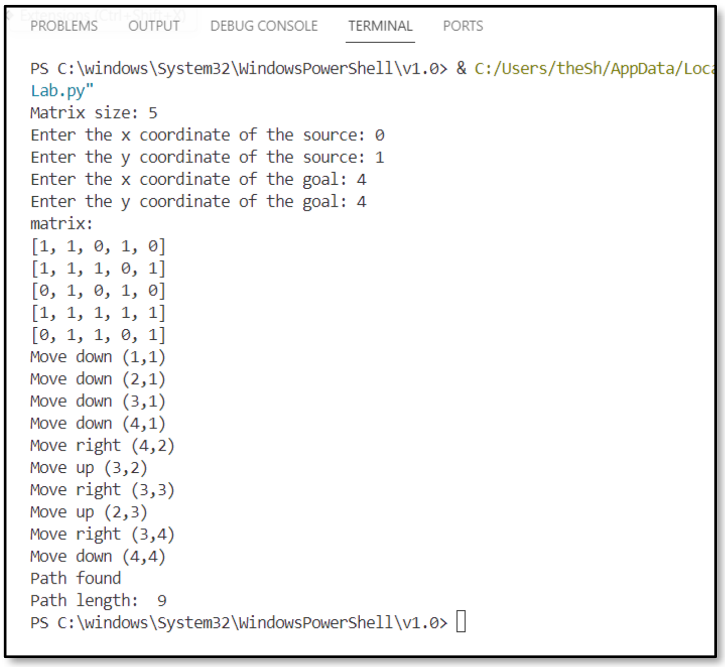


```python
import random

class Node:
    def __init__(self, a, b, z):
        self.x = a
        self.y = b
        self.depth = z

class DFS:
    def __init__(self):
        self.directions = 4
        self.x_move = [1, -1, 0, 0]             #shahidul
        self.y_move = [0, 0, 1, -1]
        self.found = False
        self.N = 0
        self.source = None
        self.goal = None
        self.goal_level = 9999999
        self.state = 0

    def generate_grid(self):
        self.N = int(input("Matrix size: "))
        graph = [[random.choice([0, 1]) for i in range(self.N)] for j in range(self.N)]
        return graph

    def init(self):
        graph = self.generate_grid()

        start_x = int(input("Enter the x coordinate of the source: "))
        start_y = int(input("Enter the y coordinate of the source: "))
        goal_x = int(input("Enter the x coordinate of the goal: "))
        goal_y = int(input("Enter the y coordinate of the goal: "))

        self.source = Node(start_x, start_y, 0)
        self.goal = Node(goal_x, goal_y, self.goal_level)

        print("matrix:")

        for row in graph:
            print(row)
```



```python
        self.st_dfs(graph, self.source)

        if self.found:
            print("Path found")
            print("Path length: ", self.goal_level)
        else:
            print("Path not found")

    def print_direction(self, m, x, y):
        if m == 0:
            print("Move down ({},{})".format(x, y))
        elif m == 1:
            print("Move up ({},{})".format(x, y))
        elif m == 2:
            print("Move right ({},{})".format(x, y))
        elif m == 3:
            print("Move left ({},{})".format(x, y))

    def st_dfs(self, graph, u):
        graph[u.x][u.y] = 0
        for j in range(self.directions):
            v_x = u.x + self.x_move[j]
            v_y = u.y + self.y_move[j]
            if (0 <= v_x < self.N) and (0 <= v_y < self.N) and graph[v_x][v_y] == 1:
                v_depth = u.depth + 1
                self.print_direction(j, v_x, v_y)
                if v_x == self.goal.x and v_y == self.goal.y:
                    self.found = True
                    self.goal_level = v_depth
                    return
                child = Node(v_x, v_y, v_depth)
                self.st_dfs(graph, child)
                if self.found:
                    return

def main():
    dfs = DFS()
    dfs.init()

if __name__ == "__main__":
    main()
```

5. TEST RESULT / OUTPUT



```
PS C:\windows\System32\WindowsPowerShell\v1.0> & C:/Users/theSh/AppData/Loca
Lab.py"
Matrix size: 5
Enter the x coordinate of the source: 0
Enter the y coordinate of the source: 1
Enter the x coordinate of the goal: 4
Enter the y coordinate of the goal: 4
matrix:
[1, 1, 0, 1, 0]
[1, 1, 1, 0, 1]
[0, 1, 0, 1, 0]
[1, 1, 1, 1, 1]
[0, 1, 1, 0, 1]
Move down (1,1)
Move down (2,1)
Move down (3,1)
Move down (4,1)
Move right (4,2)
Move up (3,2)
Move right (3,3)
Move up (2,3)
Move right (3,4)
Move down (4,4)
Path found
Path length:  9
PS C:\windows\System32\WindowsPowerShell\v1.0> 
```

6. ANALYSIS AND DISCUSSION

The lab report explored the implementation of the depth-first search (DFS) algorithm for pathfinding in a grid environment. Key points and learning outcomes include:

- Understanding the concept of DFS and its application in graph traversal.
- Designing and implementing a recursive DFS algorithm to navigate through obstacles in a grid.
- Analyzing the strengths and weaknesses of DFS, including its simplicity in implementation and potential scalability issues.
- Highlighting the importance of considering the underlying principles and design choices when utilizing DFS for pathfinding tasks.

One notable aspect of DFS is its simplicity in implementation, thanks to the recursive approach. However, this simplicity comes with potential drawbacks, such as the risk of encountering infinite loops in cyclic graphs or reaching stack overflow with deep recursion in large grids.

Despite its limitations, DFS remains a valuable tool for pathfinding, especially in scenarios where memory consumption is a concern or when the grid is relatively small. By understanding its underlying principles and considering its strengths and weaknesses, we can leverage DFS effectively in various applications, from maze-solving algorithms to network traversal in computer science.

7. SUMMARY

In conclusion, The Depth-First Search (DFS) algorithm implemented in this lab report offers an efficient solution for pathfinding in a grid environment. By systematically exploring neighboring nodes, DFS can navigate through obstacles to find a path from the source to the goal.
Overall, the lab report provided valuable insights into the practical application of DFS and its significance in solving real-world problems. By applying the knowledge gained from this exercise, one can better appreciate the versatility and effectiveness of DFS in various domains of computer science and beyond.