Green University of Bangladesh
Department of Computer Science and Engineering(CSE)
Faculty of Sciences and Engineering
Semester: (Spring, Year:2024), B.Sc. in CSE (Day)

LAB REPORT NO 03

Course Title: Artificial Intelligence Lab

Course Code: CSE 316 **Section: 213-D1**

Lab Experiment Name: Implement Depth-First Search.

Student Details

Name	ID
Md Abu Ubaida Jubaer Sayeb	213902113

Lab Date : **19 March 2024**
Submission Date : **02 April 2024**
Course Teacher's Name : **Fairuz Shaiara**

[For Teachers use only: Don't Write Anything inside this box]

Lab Report Status

Marks: ……………………………
Signature:.....................
Comments:..
Date:..............................

1. TITLE OF THE LAB EXPERIMENT

Implement Depth-First Search.

2. OBJECTIVES/AIM

- Traverse through a graph or tree structure systematically, exploring as far as possible along each branch before backtracking.
- Determine connectivity and identify cycles within the graph efficiently by visiting nodes in a depth-first manner.

3. PROCEDURE / ANALYSIS / DESIGN

Algorithm:

- Define a Node class to represent a point in the grid with coordinates (x, y) and depth information.
- Implement a Depth First Search (DFS) class with methods to generate a grid, initialize search parameters, print directions, and perform depth-first search.
- In the DFS class, initialize parameters such as directions (4 directions: up, down, left, right), moves in x and y directions, and variables to track the goal state and whether it's found.
- Implement methods to generate a random grid, initialize search parameters by taking input for source and goal coordinates, and display the grid.
- Define a method to print the direction of movement based on the move index.
- Implement the recursive depth-first search method (st_dfs) to traverse the grid, updating the depth and checking for goal state at each step.
- In the main function, create an instance of the DFS class and initialize the search.

- If the goal is found, print the number of moves required to reach the goal. If the goal cannot be reached, print a corresponding message.

4. IMPLEMENTATION

Code:

DFS.py

DFS.py > ...

```
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]
        self.y_move = [0, 0, 1, -1]
        self.found = False
        self.N = 0
        self.source = None
        self.goal = None
        self.goal_level = 999999
        self.state = 0

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

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

        start_x = int(input("Source X: "))
        start_y = int(input("Source Y: "))
        goal_x = int(input("Goal X: "))
        goal_y = int(input("Goal Y: "))

        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)

        self.st_dfs(graph, self.source)

        if self.found:
            print("Goal found")
            print("Number of moves required =", self.goal.depth)
        else:

            print("Goal cannot be reached from the starting block")

    def print_direction(self, m, x, y):
        if m == 0:
            print("Moving Down ({}, {})".format(x, y))
        elif m == 1:
            print("Moving Up ({}, {})".format(x, y))
        elif m == 2:
            print("Moving Right ({}, {})".format(x, y))
        else:
            print("Moving 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.depth = v_depth
                    return
                child = Node(v_x, v_y, v_depth)
                self.st_dfs(graph, child)
                if self.found:
                    return

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

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

Figure 1: DFS Code

5. TEST RESULT / OUTPUT

Output:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    SQL CONSOLE

PS C:\Users\User\Desktop\JS ANIMATION> & "C:/Program Files/Python312/python.exe" "c:/Users/User/Desktop/JS ANIMATION/DFS.py"
Matrix Size: 5
Source X: 0
Source Y: 1
Goal X: 4
Goal Y: 4
Matrix:
[0, 1, 1, 0, 0]
[1, 1, 1, 1, 1]
[1, 0, 1, 1, 0]
[1, 0, 0, 0, 0]
[0, 1, 1, 0, 0]
Moving Down (1, 1)
Moving Right (1, 2)
Moving Down (2, 2)
Moving Right (2, 3)
Moving Up (1, 3)
Moving Right (1, 4)
Moving Up (0, 2)
Moving Left (1, 0)
Moving Down (2, 0)
Moving Down (3, 0)
Goal cannot be reached from the starting block
PS C:\Users\User\Desktop\JS ANIMATION>
```

Figure 2: DFS Output

6. ANALYSIS AND DISCUSSION

Depth-First Search (DFS) is a fundamental algorithm used to traverse graphs or trees. It explores as far as possible along each branch before backtracking. This approach is implemented using a stack, making it memory-efficient but prone to infinite loops in graphs with cycles. DFS is widely used in various applications, such as pathfinding, topological sorting, and solving puzzles.