

B.TECH. (2020-24)
Artificial Intelligence

LAB Assignment
(BFS and DFS Path Finding)
on
ARTIFICIAL INTELLIGENCE
[CSE401]



Submitted To
Mr. Soumya Ranjan Nayak

Submitted By
HITESH
A023119820027
4CSE11 (AI)

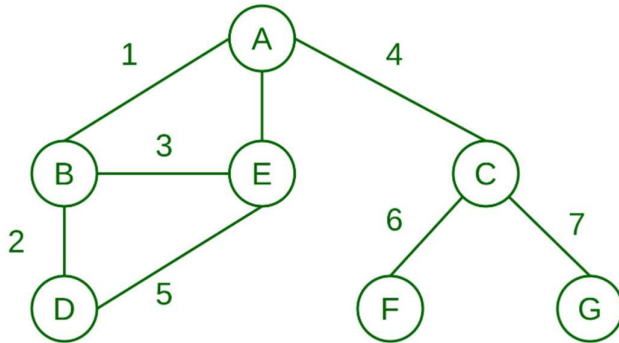
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
AMITY SCHOOL OF ENGINEERING AND TECHNOLOGY
AMITY UNIVERSITY UTTAR PRADESH
NOIDA (U.P)

LAB Assignment

Program 1: Finding path between start and goal state using Depth First Search Algorithm.

Language Used: Python

Graph Used:



Code:

```
def dfs(adj_list, start, target, path, visited = set()):
    path.append(start)
    visited.add(start)
    if start == target:
        return path
    for neighbour in adj_list[start]:
        if neighbour not in visited:
            result = dfs(adj_list, neighbour, target, path, visited)
            if result is not None:
                return result
    path.pop()
    return None
```

```
adj_list = {'A': ['B', 'E', 'C'],
            'B': ['A', 'D', 'E'],
            'C': ['A', 'F', 'G'],
            'D': ['B', 'E'],
            'E': ['A', 'B', 'D'],
            'F': ['C'],
            'G': ['C']}
```

```
traversal_path = []
traversal_path = dfs(adj_list, 'A', 'D', traversal_path)
print(traversal_path)
```

Output:

```
File Edit Selection View Go Run Terminal Help
DFS.py - Untitled (Workspace) - Visual Studio Code

BFS.py DFS.py x
AI LAB > LAB 2 > DFS.py > ...
1 def dfs(adj_list, start, target, path, visited = set()):
2     path.append(start)
3     visited.add(start)
4     if start == target:
5         return path
6     for neighbour in adj_list[start]:
7         if neighbour not in visited:
8             result = dfs(adj_list, neighbour, target, path, visited)
9             if result is not None:
10                return result
11     path.pop()
12     return None
13
14
15 adj_list = {'A': ['B', 'E', 'C'],
16             'B': ['A', 'D'],
17             'C': ['A', 'F', 'G'],
18             'D': ['B'],
19             'E': ['A', 'D'],
20             'F': ['C'],
21             'G': ['C']}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

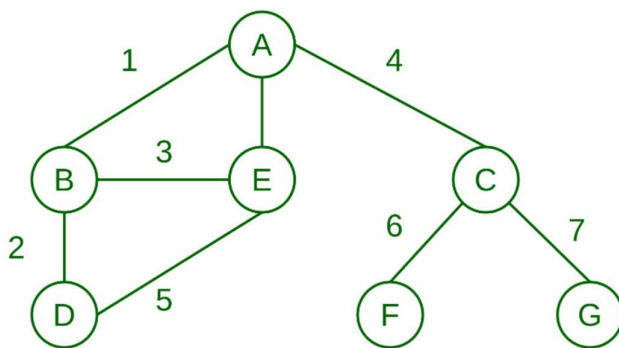
Install the latest PowerShell for new features and improvements! <https://aka.ms/PSWindows>

PS E:\Amity Data\Semester 4\AI\AI LAB> python -u "e:\Amity Data\Semester 4\AI\AI LAB\LAB 2\DFS.py"
['A', 'B', 'D']
PS E:\Amity Data\Semester 4\AI\AI LAB>

Program 2: Finding path between start and goal state using Breadth First Search Algorithm.

Language Used: Python

Graph Used:



Code:

```
def BFS_SP(graph, start, goal):
    explored = []

    # Queue for traversing the
    # graph in the BFS
    queue = [[start]]
```

```

# If the desired node is
# reached
if start == goal:
    print("Same Node")
    return

# Loop to traverse the graph
# with the help of the queue
while queue:
    path = queue.pop(0)
    node = path[-1]

    # Condition to check if the
    # current node is not visited
    if node not in explored:
        neighbours = graph[node]

        # Loop to iterate over the
        # neighbours of the node
        for neighbour in neighbours:
            new_path = list(path)
            new_path.append(neighbour)
            queue.append(new_path)

            # Condition to check if the
            # neighbour node is the goal
            if neighbour == goal:
                print("Shortest path = ", *new_path)
                return
            explored.append(node)

# Condition when the nodes
# are not connected
print("So sorry, but a connecting" \
      "path doesn't exist :(")

return

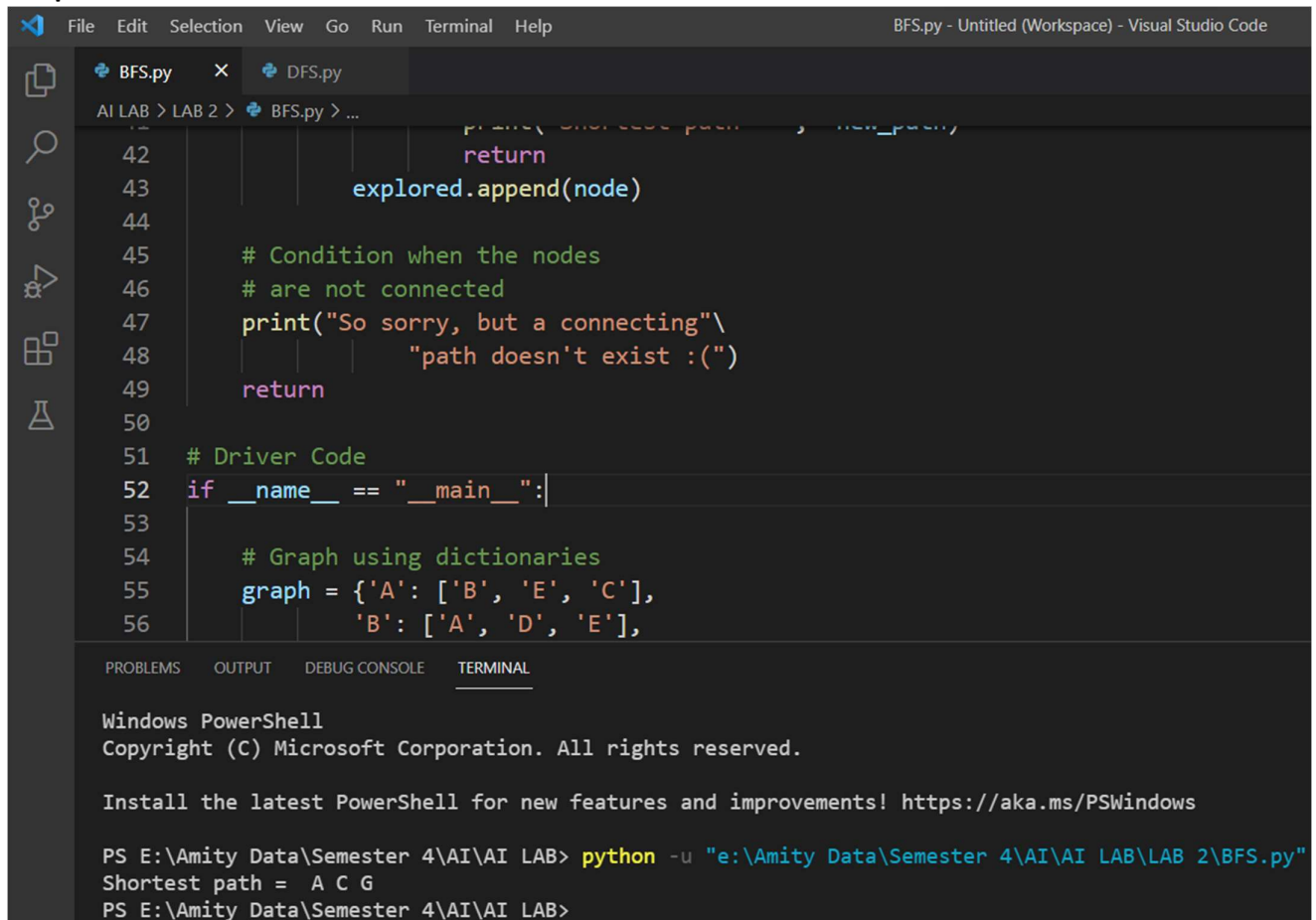
# Driver Code
if __name__ == "__main__":

    # Graph using dictionaries
    graph = {'A': ['B', 'E', 'C'],
            'B': ['A', 'D', 'E'],
            'C': ['A', 'F', 'G'],
            'D': ['B', 'E'],
            'E': ['A', 'B', 'D'],
            'F': ['C'],
            'G': ['C']}

```

```
# Function Call
BFS_SP(graph, 'A', 'G')
```

Output:



The screenshot shows the Visual Studio Code interface with a file named `BFS.py` open. The code defines a Breadth-First Search (BFS) function to find the shortest path between two nodes in a graph. The graph is represented as a dictionary where each node is a key and its neighbors are the values. The function `BFS_SP` takes the graph, a start node, and a target node as input. It uses a queue to explore nodes level by level until the target node is reached. If the target node is not reachable, it returns a message indicating that the path does not exist. The driver code at the bottom creates a graph with nodes A, B, C, D, and E, and their connections. It then calls `BFS_SP` to find the shortest path from node A to node G.

```
42         print(shortest_path, new_path)
43         return
44     explored.append(node)
45     # Condition when the nodes
46     # are not connected
47     print("So sorry, but a connecting"\
48           "path doesn't exist :(")
49     return
50
51 # Driver Code
52 if __name__ == "__main__":
53
54     # Graph using dictionaries
55     graph = {'A': ['B', 'E', 'C'],
56             'B': ['A', 'D', 'E'],
```

The terminal output shows the execution of the script. It displays the shortest path from node A to node G as `A C G`.

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS E:\Amity Data\Semester 4\AI\AI LAB> python -u "e:\Amity Data\Semester 4\AI\AI LAB\LAB 2\BFS.py"
Shortest path =  A C G
PS E:\Amity Data\Semester 4\AI\AI LAB>
```