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)

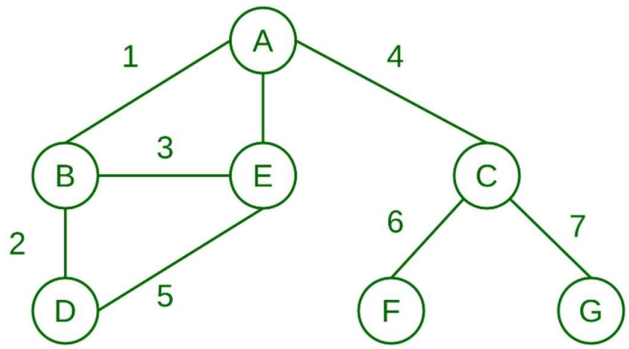
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
AMITY SCHOOL OF ENGINEERING AND TECHNOLOGY
AMITY UNIVERSITY UTTAR PRADESH
NOIDA (U.P)

LAB Assignment

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

Language Used: Python

Graph Used:



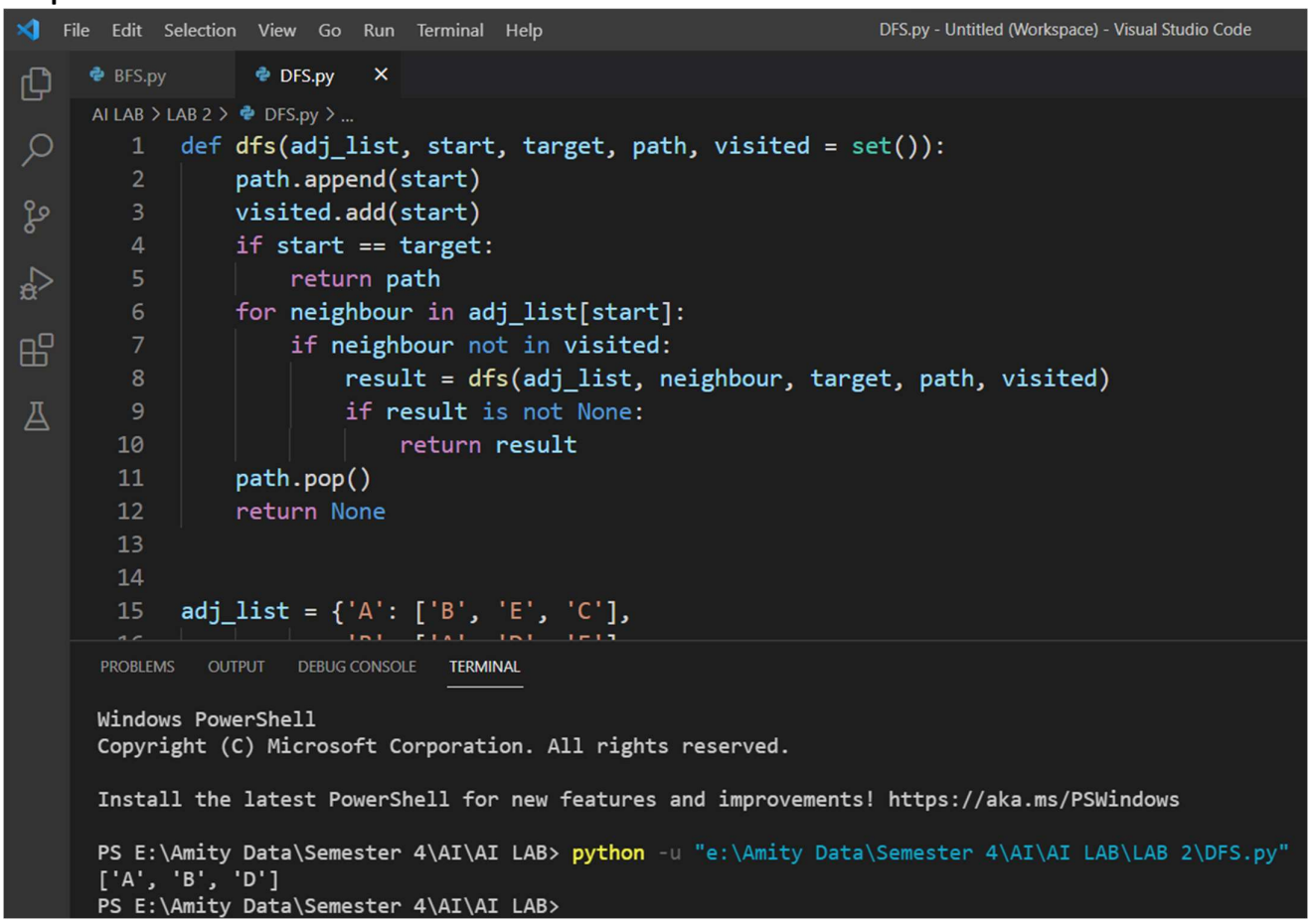
Code:

```
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)
```

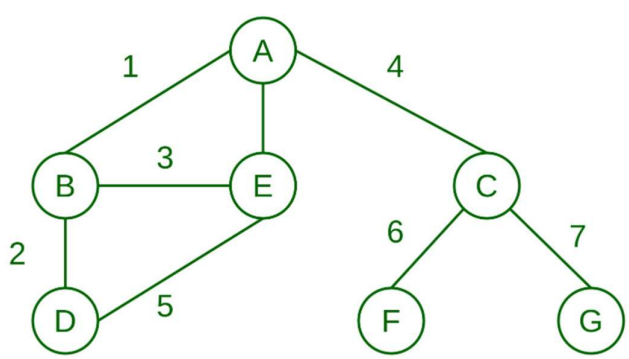
Output:



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: