DATE:25.07.25

TASK:1

Implementation of Graph search algorithms

(Breadth first search and Depth First Search)

AIM

To Implement of Graph search algorithms (Breadth first search and Depth First Search) using Python

ALGORITHM

BFS

1. Create an empty visited set to keep track of visited spots.
2. Create a queue and add the starting node to it.
3. While the queue is not empty:
   - Remove the front node from the queue.
   - If the node has not been visited:
     - Mark it as visited and print it.
     - If the node is the treasure, stop and report success.
     - Add all of its neighboring nodes to the queue.
     -

DFS

1.Start with the given node and an empty visited set.

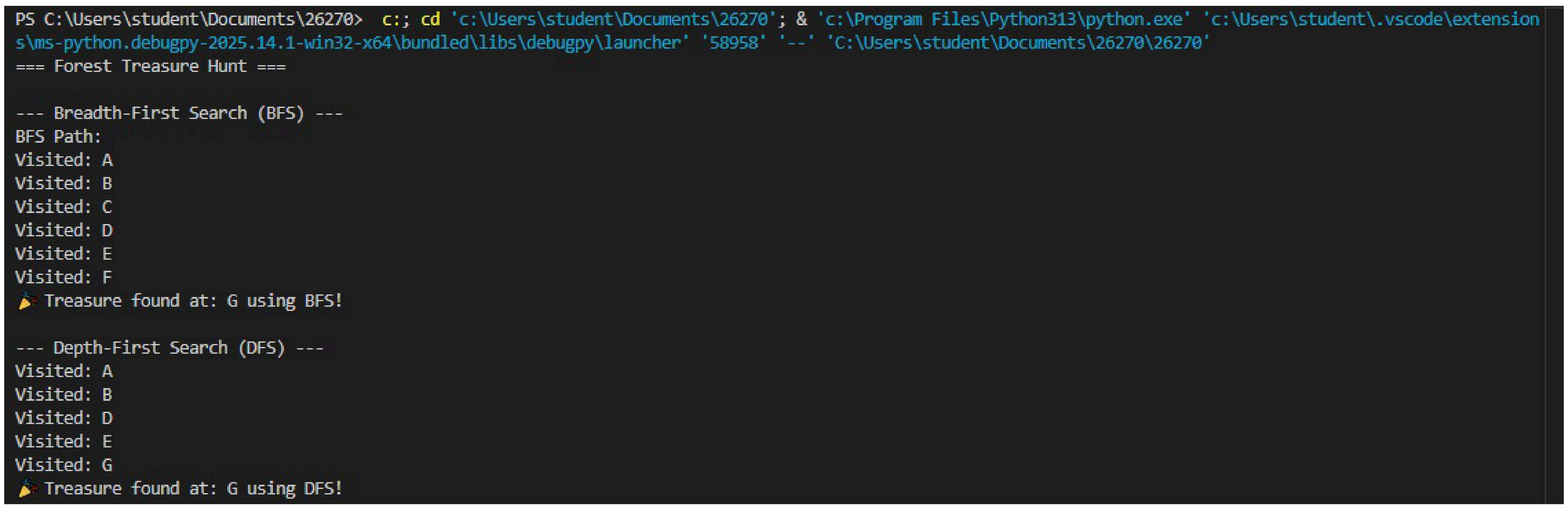
2. If the node is not visited:
   - Mark it as visited and print it.
   - If it is the treasure node, report success and stop.
   - For each neighbor of the current node:
     - Recursively apply DFS to that neighbor.
     - If treasure is found in any path, stop.

PROGRAM
Forest Treasure Hunt

```
from collections import deque

# Forest map represented as a graph
forest = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [],
    'E': ['G'],  # Treasure is at G
    'F': [],
    'G': []
}

# Function to perform BFS
def bfs(graph, start, treasure):
    visited = set()
    queue = deque([start])

    print("BFS Path:")
    while queue:
        node = queue.popleft()
        if node not in visited:
            print(f"Visited: {node}")
            visited.add(node)

            if node == treasure:
                print(f"🎉Treasure found at: {node} using BFS!")
                return

            for neighbor in graph[node]:
                queue.append(neighbor)
    print("Treasure not found in the forest using BFS.")
```

```
# Function to perform DFS
def dfs(graph, node, treasure, visited=None):
    if visited is None:
        visited = set()

    if node not in visited:
        print(f"Visited: {node}")
        visited.add(node)

        if node == treasure:
            print(f"🎉Treasure found at: {node} using DFS!")
            return True

        for neighbor in graph[node]:
            if dfs(graph, neighbor, treasure, visited):
                return True
    return False

# Run both searches
start_node = 'A'
treasure_node = 'G'

print("=== Forest Treasure Hunt ===\n")

print("--- Breadth-First Search (BFS) ---")
bfs(forest, start_node, treasure_node)

print("\n--- Depth-First Search (DFS) ---")
found = dfs(forest, start_node, treasure_node)
if not found:
    print("Treasure not found in the forest using DFS.")
```

OUTPUT



```
PS C:\Users\student\Documents\26270>  c:; cd 'c:\Users\student\Documents\26270'; & 'c:\Program Files\Python313\python.exe' 'c:\Users\student\.vscode\extension
s\ms-python.debugpy-2025.14.1-win32-x64\bundled\libs\debugpy\launcher' '58958' '--' 'C:\Users\student\Documents\26270\26270'
=== Forest Treasure Hunt ===

--- Breadth-First Search (BFS) ---
BFS Path:
Visited: A
Visited: B
Visited: C
Visited: D
Visited: E
Visited: F
🎉 Treasure found at: G using BFS!

--- Depth-First Search (DFS) ---
Visited: A
Visited: B
Visited: D
Visited: E
Visited: G
🎉 Treasure found at: G using DFS!
```

RESULT

Thus, the Implementation of Graph search algorithms (Breadth first search and Depth First Search) using Python was successfully executed and output was verified.