DATE:25.07.25

TASK 1
Breadth first search and Depth First Search

Implementation of Graph search algorithms (Breadth first search and Depth First Search) using following constraints.
BFS: Pick any node, visit the adjacent unvisited vertex, mark it as visited, display it, and insert it in a queue. If there are no remaining adjacent vertices left, remove the first vertex from the queue. Apply recursion concept to follow the above steps until the queue is empty or the desired node is found.
DFS: Pick any node. If it is unvisited, mark it as visited and recur on all its adjacent nodes. Repeat until all the nodes are visited, or the node to be searched is found.
Tools- Python

PROBLEM STATEMENT: CO1 S3

A forest filled with many interconnected spots, forming a network similar to a graph. Each spot represents a location you can visit, and the trails between them represent paths you can follow. Somewhere in this forest, a hidden treasure awaits discovery. To find it, you can use either the Breadth-First Search (BFS) or Depth-First Search (DFS) technique. BFS explores the forest level by level, starting from your current position and visiting all the nearby spots before moving deeper. This method uses a queue and is effective if the treasure is closer to your starting point. On the other hand, DFS dives deep into one path before backtracking, exploring as far as possible along each branch using recursion or a stack. This method is helpful when the treasure is hidden deep within the forest. Both methods allow you to systematically search all possible paths until the treasure is found.

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

```
--- Breadth-First Search (BFS) ---
Visited: A
Visited: B
Visited: C
Visited: D
Visited: E
Visited: F
Visited: G
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.