Date:23.07.25

TASK:1

Implementation of Graph search algorithms
(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

In a social networking platform, users are connected through friendships, forming a network of relationships. Each user can be represented as a node in a graph, and a friendship between two users is represented as an edge. To build a friend recommendation system, we can use graph search algorithms like Breadth-First Search (BFS) and Depth-First Search (DFS). BFS is useful for exploring the network level by level, helping to identify "friends of friends" who can be suggested as potential new connections. For example, starting from a user, BFS first finds their immediate friends, and then their friends' friends. DFS, on the other hand, explores each connection path deeply before backtracking, which is helpful for analyzing how far and in what ways users are connected throughout the network. By using both BFS and DFS, we can effectively explore the structure of the social network for various purposes like friend suggestions, connection analysis, and understanding the reach of each user.

BREADTH FIRST SEARCH AND DEPTH FIRST SEARCH

AIM

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

BFS – Shortest Path in a Social Network Graph

ALGORITHM

Step 1:

Start with an empty list visited to keep track of the nodes already visited.

Step 2:

Create an empty list queue which will be used to explore the graph level by level.

Step 3:

Insert the starting node into both visited and queue.

Step 4:

Repeat the following steps until the queue is empty.

Step 5:

Remove the front element from the queue and call it the current_node.

Step 6:

For each neighbor of the current_node, if it is not already in visited, then:

→ Add it to visited

→ Add it to the queue

Step 7:

Continue the process until the queue becomes empty. The visited list now contains nodes in BFS traversal order.

DFS

ALGORITHM

Step 1:

Start with an empty set visited to keep track of the nodes already visited.

Step 2:

Initialize a stack and push the starting node onto it.

Step 3:

Repeat the following steps until the stack becomes empty.

Step 4:

Pop the top element from the stack and call it current_node.

Step 5:

If current_node is not in visited, then:

→ Print or process the node.

→ Add current_node to the visited set.

Step 6:

For each neighbor of current_node in the graph:

→ If the neighbor is not in visited, push it onto the stack.

Step 7:

Continue the process until the stack becomes empty. The nodes will be visited in DFS order.

PROGRAM

Graph Representation (Friend Network)

```
from collections import deque

class Graph:
    def __init__(self):
        self.graph = {}

    def add_edge(self, u, v):
        if u not in self.graph:
            self.graph[u] = []
        if v not in self.graph:
            self.graph[v] = []
        self.graph[u].append(v)
        self.graph[v].append(u)

    def bfs(self, start):
        visited = set()
        queue = deque([start])
        visited.add(start)

        print("BFS Traversal:", end=" ")
        while queue:
            node = queue.popleft()
            print(node, end=" ")
```

```
            for neighbor in self.graph[node]:
                if neighbor not in visited:
                    visited.add(neighbor)
                    queue.append(neighbor)
        print()

    def dfs(self, start):
        visited = set()
        print("DFS Traversal:", end=" ")
        self._dfs_recursive(start, visited)
        print()

    def _dfs_recursive(self, node, visited):
        visited.add(node)
        print(node, end=" ")
        for neighbor in self.graph[node]:
            if neighbor not in visited:
                self._dfs_recursive(neighbor, visited)


if __name__ == "__main__":
    g = Graph()
    g.add_edge(0, 1)
    g.add_edge(0, 2)
    g.add_edge(1, 3)
    g.add_edge(1, 4)
    g.add_edge(2, 5)
    g.add_edge(2, 6)

    g.bfs(0)  # BFS starting from node 0
    g.dfs(0)  # DFS starting from node 0
```

OUTPUT

```
PS C:\Users\suman> & C:/Users/suman/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/suman/Documents/vtu26845.py
BFS Traversal: 0 1 2 3 4 5 6
DFS Traversal: 0 1 3 4 2 5 6
PS C:\Users\suman>
```

RESULT

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