Practical 1

Implement depth first search algorithm and Breadth First Search algorithm, use an undirected graph
and develop a recursive algorithm for searching all the vertices of a graph or tree data structure.

Source Code :-

```
from collections import deque

# Function to create the graph
def create_graph():
    graph = {}
    while True:
        # Ask the user for input (node and its neighbors)
        node = input("Enter node (or type 'done' to finish): ").strip()

        if node.lower() == 'done':
            break

        if node not in graph:
            graph[node] = []

        # Ask for neighbors of the node
        neighbors = input(f"Enter neighbors of node {node} separated by
space: ").strip().split()

        # Add neighbors to the node in the graph (undirected edges)
        for neighbor in neighbors:
            if neighbor not in graph:
                graph[neighbor] = []
            graph[node].append(neighbor)
            graph[neighbor].append(node)  # Undirected graph

    return graph

# DFS (Recursive) Function
def dfs(graph, node, visited=None):
    if visited is None:
        visited = set()  # Keeps track of visited nodes

    # Mark the node as visited
    visited.add(node)
    print(node, end=" ")  # Process the node (here, just print it)

    # Visit all the neighbors
    for neighbor in graph[node]:
        if neighbor not in visited:
            dfs(graph, neighbor, visited)

# BFS (Iterative) Function
def bfs(graph, start):
    visited = set()  # Keeps track of visited nodes
    queue = deque([start])  # Queue for BFS

    while queue:
        node = queue.popleft()
        if node not in visited:
            print(node, end=" ")  # Process the node (here, just print
it)
            visited.add(node)

            # Add all unvisited neighbors to the queue
            for neighbor in graph[node]:
                if neighbor not in visited:
                    queue.append(neighbor)

# Function to display the graph
def display_graph(graph):
    print("\nGraph Representation:")
    for node,neighbors in graph.items():
        print(f"{node}: {', '.join(neighbors)}")

# Menu-driven function for DFS and BFS selection
def menu(graph):
    while True:
        print("\nMenu:")
        print("1. Display Graph")
        print("2. Perform DFS")
        print("3. Perform BFS")
        print("4. Exit")

        choice = input("Enter your choice: ").strip()

        if choice == '1':
            display_graph(graph)
        elif choice == '2':
            start_node = input("Enter starting node for DFS: ").strip()
            print("\nDFS Traversal:")
            if start_node in graph:
                dfs(graph, start_node)
            else:
                print("Invalid node.")
        elif choice == '3':
            start_node = input("Enter starting node for BFS: ").strip()
            print("\nBFS Traversal:")
            if start_node in graph:
                bfs(graph, start_node)
            else:
                print("Invalid node.")
        elif choice == '4':
            print("Exiting program.")
            break
        else:
            print("Invalid choice. Please try again.")

# Main function to drive the program
def main():
    print("Welcome to the Graph Traversal Program!")
    graph = create_graph()
    menu(graph)

if __name__ == "__main__":
    main()
```

Output :-

```
PS C:\Users\System05\Documents\13212> &
C:/Users/System05/AppData/Local/Programs/Python/Python313/python.exe
c:/Users/System05/Documents/13212/dem.py
Welcome to the Graph Traversal Program!
Enter node (or type 'done' to finish): a
Enter neighbors of node a separated by space: b
Enter node (or type 'done' to finish): b
Enter neighbors of node b separated by space: c
Enter node (or type 'done' to finish): c
Enter neighbors of node c separated by space: d
Enter node (or type 'done' to finish): d
Enter neighbors of node d separated by space: a
Enter node (or type 'done' to finish): done

Menu:
1. Display Graph
2. Perform DFS
3. Perform BFS
4. Exit
Enter your choice: 1

Graph Representation:
a: b, d
b: a, c
c: b, d
d: c, a

Menu:
1. Display Graph
2. Perform DFS
3. Perform BFS
4. Exit
Enter your choice: 2
Enter starting node for DFS: a

DFS Traversal:
a b c d
Menu:
1. Display Graph
2. Perform DFS
3. Perform BFS
4. Exit
Enter your choice: 3
Enter starting node for BFS: b

BFS Traversal:
b a c d
Menu:
1. Display Graph
2. Perform DFS
3. Perform BFS
4. Exit
Enter your choice: 1
```

```
Graph Representation:
a: b, d
b: a, c
c: b, d
d: c, a

Menu:
1. Display Graph
2. Perform DFS
3. Perform BFS
4. Exit
Enter your choice: 4
Exiting program.
```