Experiment 4

Aim: Implement Depth-First Search (DFS) in Python.

Theory: Depth-First Search (DFS) is a fundamental graph traversal algorithm used in computer science for exploring graphs and trees. It starts from a root node and explores as far as possible along each branch before backtracking. This approach ensures that every path from the starting node to a terminal node is fully explored before moving to the next neighbor.

DFS can be implemented using a stack (iterative approach) or recursion (implicit stack). In the recursive approach, the algorithm visits a node, marks it as visited, and recursively explores each unvisited neighbor. Backtracking occurs when a node has no unvisited neighbors.

Applications of DFS include:

- Detecting cycles in graphs
- Solving puzzles with backtracking (e.g., mazes, Sudoku)
- Finding connected components in a graph
- Topological sorting in Directed Acyclic Graphs (DAGs)

Time Complexity: O(V + E), where V is the number of vertices and E is the number of edges.
Space Complexity: O(V) in the case of recursion due to the call stack.

Code:

```
from typing import Dict, List, Set, Any

class GraphDFSSolver:
    def __init__(self):
        """
        Initialize the graph for DFS traversal
        """
        self.graph: Dict[Any, List[Any]] = {}

    def add_edge(self, node: Any, neighbor: Any):
        """
        Add an edge to the graph
        """
        if node not in self.graph:
            self.graph[node] = []
        self.graph[node].append(neighbor)

    def dfs_recursive(self, start_node: Any) -> List[Any]:
        """
```

```
        Perform Depth-First Search using recursion
        """
        visited: Set[Any] = set()
        traversal_order: List[Any] = []

        def dfs_helper(node: Any):
            visited.add(node)
            traversal_order.append(node)
            if node in self.graph:
                for neighbor in self.graph[node]:
                    if neighbor not in visited:
                        dfs_helper(neighbor)

        dfs_helper(start_node)
        return traversal_order

# Example usage
def main():
    graph_solver = GraphDFSSolver()
    graph_solver.add_edge('A', 'B')
    graph_solver.add_edge('A', 'C')
    graph_solver.add_edge('B', 'D')
    graph_solver.add_edge('B', 'E')
    graph_solver.add_edge('C', 'F')
    graph_solver.add_edge('E', 'F')

    print("Recursive DFS:")
    print(graph_solver.dfs_recursive('A'))

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

Output: For the given graph, the recursive DFS traversal order starting from node 'A' would be:

```
DFS:
['A', 'B', 'D', 'E', 'F', 'C']
```