Experiment No.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.

Code:

```
from collections import defaultdict

class Graph:
    def __init__(self):
        self.graph = defaultdict(list)
    def add_edge(self, u, v):
        self.graph[u].append(v)
        self.graph[v].append(u)
    def dfs(self, start, visited):
        visited.add(start)
        print(start, end=' ')
        for neighbor in self.graph[start]:
            if neighbor not in visited:
                self.dfs(neighbor, visited)
    def bfs(self, start, visited):
        queue = []
        queue.append(start)
        visited.add(start)
        while queue:
            node = queue.pop(0)
            print(node, end=' ')
            for neighbor in self.graph[node]:
```

```
            if neighbor not in visited:
                queue.append(neighbor)
                visited.add(neighbor)
# Take input for the graph
g = Graph()
n = int(input("Enter the number of vertices: "))
m = int(input("Enter the number of edges: "))
print("Enter edges as pairs (u v), one pair per line:")
for _ in range(m):
    u, v = map(int, input().split())
    g.add_edge(u, v)
start_vertex = int(input("Enter the starting vertex for DFS and BFS: "))
visited_dfs = set()
visited_bfs = set()
print("\nDFS starting from vertex", start_vertex)
g.dfs(start_vertex, visited_dfs)
print("\nBFS starting from vertex", start_vertex)
g.bfs(start_vertex, visited_bfs)
```

Output:

```
Enter the number of vertices: 6
Enter the number of edges: 5
Enter edges as pairs (u v), one pair per line:
0 1
0 2
1 3
1 4
2 5
Enter the starting vertex for DFS and BFS: 0

DFS starting from vertex 0
0 1 3 4 2 5
BFS starting from vertex 0
0 1 2 3 4 5 %
```