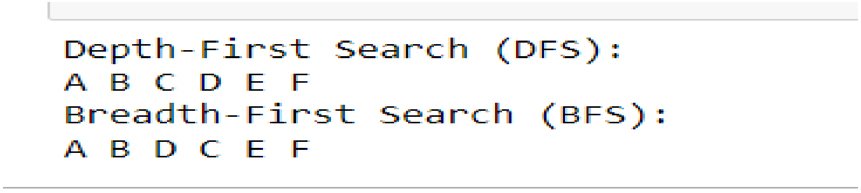
Code:

```
graph = {
    'A': ['B', 'D'],
    'B': ['A', 'C'],
    'C': ['B'],
    'D': ['A', 'E', 'F'],
    'E': ['D'],
    'F': ['D']
}
def dfs_recursive(graph, node, visited):
    if node not in visited:
        print(node, end=' ')
        visited.add(node)
        for neighbor in graph[node]:
            if neighbor not in visited:
                dfs_recursive(graph, neighbor, visited)
def dfs(graph):
    visited = set()
    for node in graph:
        if node not in visited:
            dfs_recursive(graph, node, visited)
print("Depth-First Search (DFS):")
dfs(graph)
from collections import deque
def bfs(graph):
    visited = set()
    queue = deque()
    for node in graph:
        if node not in visited:
            queue.append(node)
            visited.add(node)
            while queue:
```

```
        current_node = queue.popleft()
        print(current_node, end=' ')  # Process the current node
        for neighbor in graph[current_node]:
            if neighbor not in visited:
                queue.append(neighbor)
                visited.add(neighbor)
print("\nBreadth-First Search (BFS):")
bfs(graph)
```

Output:



```
Depth-First Search (DFS):
A B C D E F
Breadth-First Search (BFS):
A B D C E F
```

Time Complexity:

Depth-First Search (DFS):

In the worst case, where we traverse the entire graph, the time complexity of the recursive DFS is O(V + E), which means it visits each vertex and edge at most once.

Breadth-First Search (BFS):

In BFS, we explore all vertices at a given level before moving on to the next level.

In the worst case, BFS will visit all V vertices and all E edges once, resulting in a time complexity of O(V + E).