Name: Mehedi Hasan Saim

Id: 221-15-4844

DFS (Depth First Search) Algorithms

1) Objectives of these Algorithms

2) Example by showing each iteration

3) Code + Complexity analysis (Time+place with proper Mathematical derivation)

4) Advantage + Disadvantage

5) Real-life Applications

Solution:

1) Objectives of DFS:

- To traverse or search a graph or tree data structure in a depthward motion.
- To explore as far as possible along each branch before backtracking.

2) Example with Iterations:

- Consider a simple graph:

```
A -- B
|    |
C -- D
```

Starting from vertex A, here's how DFS proceeds:

- Iteration 1: Visit A.

- Iteration 2: Move to B, visit B.
- Iteration 3: Move to C, visit C.
- Iteration 4: Move to D, visit D.
- Iteration 5: No unvisited neighbors, backtrack to C.
- Iteration 6: No unvisited neighbors, backtrack to B.
- Iteration 7: No unvisited neighbors, backtrack to A.
- Iteration 8: All vertices visited.

3) C Code for DFS :

```
#include <stdio.h>
#include <stdbool.h>
#define MAX_VERTICES 100
int graph[MAX_VERTICES][MAX_VERTICES];
bool visited[MAX_VERTICES];
int numVertices;

void dfs(int vertex) {
    visited[vertex] = true;
    printf("Visited vertex %d\n", vertex);

    for (int i = 0; i < numVertices; i++) {
        if (graph[vertex][i] && !visited[i]) {
            dfs(i);
        }
    }
}
```

```
int main() {
    numVertices = 4;            // Adjust as needed
    for (int i = 0; i < numVertices; i++) {
        visited[i] = false;
        for (int j = 0; j < numVertices; j++) {
            graph[i][j] = 0;
        }
    }
    graph[0][1] = 1;
    graph[0][2] = 1;
    graph[1][0] = 1;
    graph[1][3] = 1;
    graph[2][0] = 1;
    graph[2][3] = 1;
    graph[3][1] = 1;
    graph[3][2] = 1;

    // Start DFS from vertex 0
    dfs(0);
    return 0;
}
```

- Complexity Analysis :

- Time Complexity: O(V + E), where V is the number of vertices, and E is the number of edges in the graph.

- Space Complexity: O(V), for the visited array.

4) Advantages and Disadvantages:

Advantages:

- Simple to implement.

- Memory-efficient as it only requires a small amount of additional memory for the stack.

- Suitable for solving problems like topological sorting and strongly connected components in a graph.

Disadvantages:

- May not find the shortest path in unweighted graphs.

- Can go very deep in a deep graph and may not visit nodes near the start until very late, which might not be suitable for certain applications.

- Not ideal for finding paths in maze-like environments or similar scenarios where breadth-first search may be more efficient.

5) Real-life Applications:

- Maze Solving
- Web Crawling
- Game Development
- Network Routing
- Artificial Intelligence
- Social Networking