Name: Achyut Shukla Batch: AA – A1 PRN: 20070122005

DEPTH FIRST SEARCH

DFS stands for "Depth-First Search." It is a graph traversal algorithm used in computer science and graph theory to explore or search a graph or tree data structure in a specific way.

DFS starts at the root (or any arbitrary node in the case of a graph), explores as far as possible along each branch before backtracking. The algorithm explores a tree or graph by visiting a node and then recursively visiting all its adjacent nodes, marking visited nodes to avoid cycles, until all reachable nodes have been visited. DFS can be implemented using either recursion or an explicit stack data structure.

Code

```
#include <bits/stdc++.h>

using namespace std;


class Graph

{

public:

   map<int, bool> visited;

   map<int, list<int>> adj;


   void addEdge(int v, int w);


   void DFS(int v);

};


void Graph::addEdge(int v, int w)

{

   adj[v].push_back(w);

}


void Graph::DFS(int v)

{

   visited[v] = true;

   cout << v << " ";
```

```
    list<int>::iterator i;
    for (i = adj[v].begin(); i != adj[v].end(); ++i)
        if (!visited[*i])
            DFS(*i);
}

int main()
{
    Graph g;
    g.addEdge(0, 1);
    g.addEdge(0, 3);
    g.addEdge(1, 0);
    g.addEdge(1, 2);
    g.addEdge(2, 1);
    g.addEdge(2, 3);
    g.addEdge(3, 3);

    cout << "Following is Depth First Traversal (starting from vertex 1) \n";

    g.DFS(1);

    return 0;
}
```

Complexity analysis of DFS

Time Complexity - O(V + E)

Space – O(V)

Advantages of DFS:

Simplicity: DFS is a simple algorithm to implement, making it easy to understand and implement in various programming languages.

Memory efficiency: DFS can be implemented using recursion, which typically uses less memory compared to other traversal algorithms like Breadth-First Search (BFS) that require a queue data structure. This can be an advantage in memory-constrained environments.

Space efficiency: DFS can be used to explore a graph or tree in a memory-efficient manner by maintaining only a small amount of information in the stack or recursion stack.

Early detection: DFS can be used to detect cycles in a graph early during the traversal process. If a back edge is encountered during the traversal, it indicates the presence of a cycle, allowing for early detection and termination of the traversal.

Disadvantages of DFS:

Completeness: DFS does not guarantee to find the shortest path or the optimal solution, as it may follow a path that is not the shortest or optimal based on the traversal order.

Time complexity: The time complexity of DFS is O(V + E), which may not be the most efficient for certain types of graphs, especially for graphs with a large number of edges or where the desired solution is not close to the starting point. In such cases, other algorithms like BFS or more advanced algorithms may be more efficient.

Stack overflow: If implemented recursively, DFS can be prone to stack overflow errors when traversing very deep or complex graphs, especially in cases where the graph has a large depth or the recursion stack is not managed properly.