Depth First Search

Problem:
Given an undirected graph represented as an adjacency list, perform a Depth-First Search (DFS) traversal starting from a specified node, and print the graph structure and the DFS traversal order.

Description:
This program models an undirected graph using an adjacency list, where each node has a list of adjacent nodes (neighbors). The program performs the following tasks:

1. **Print the graph structure**: It displays the graph's adjacency list representation, showing each node and its adjacent nodes.
2. **DFS Traversal**: It uses a stack to perform a DFS traversal starting from a given node, marking visited nodes and traversing all adjacent nodes.

Code:
```
#include <iostream>
#include <vector>
#include <stack>
#include <set>

using namespace std;

// Function to print the graph structure (adjacency list)
void printGraph(const vector<vector<int>> &adjList)
{
    cout << "Graph Structure (Adjacency List):" << endl;
    for (int i = 1; i < adjList.size(); ++i)
    {
        cout << i << " -> ";
        for (int neighbor : adjList[i])
        {
            cout << neighbor << " ";
        }
        cout << endl;
    }
}

// Function to perform DFS using a stack
void dfs(int start, const vector<vector<int>> &adjList)
{
    vector<bool> visited(adjList.size(), false);
    stack<int> s;

    s.push(start);
    visited[start] = true;
```

```
    cout << "\nDFS Traversal: ";
    while (!s.empty())
    {
        int node = s.top();
        s.pop();

        cout << node << " ";

        // Traverse all adjacent nodes of the current node
        for (int neighbor : adjList[node])
        {
            if (!visited[neighbor])
            {
                visited[neighbor] = true;
                s.push(neighbor);
            }
        }
    }
    cout << endl;
}

int main()
{
    // Example graph represented as an adjacency list
    vector<vector<int>> adjList = {
        {},         // 0 (unused)
        {2, 3},     // 1 -> 2, 3
        {1, 4, 5}, // 2 -> 1, 4, 5
        {1},        // 3 -> 1
        {2},        // 4 -> 2
        {2}         // 5 -> 2
    };

    // Print the graph structure
    printGraph(adjList);

    // Perform DFS traversal starting from node 1
    dfs(1, adjList);

    return 0;
}
```

Output:

```
Graph Structure (Adjacency List):
1 -> 2 3
2 -> 1 4 5
3 -> 1
4 -> 2
5 -> 2

DFS Traversal: 1 3 2 5 4
```

Approach:

- **Graph Representation**: The graph is represented as an adjacency list, where each index corresponds to a node and contains a list of its adjacent nodes.
- **DFS Traversal**: The `dfs` function starts from the given node and uses a stack to explore its neighbors. It marks nodes as visited when they are traversed to avoid revisiting them.
- **Graph Printing**: The `printGraph` function prints each node along with its adjacent nodes.

Complexity Analysis:

- **Time Complexity**:
  - Printing the graph structure takes O(E) time, where E is the number of edges in the graph because each edge is printed once.
  - The DFS traversal also takes O(V + E) time, where V is the number of vertices and E is the number of edges. This is because each node and each edge is processed once.
- **Space Complexity**:
  - The space complexity is O(V) due to the space required to store the visited nodes and the stack used for DFS traversal.
  - The adjacency list requires O(V + E) space to store the graph's edges and vertices.

Thus, the overall time complexity is O(V + E), and the space complexity is O(V + E).