****

**Name: Mahlika**

**Roll: Su92BSAIM059**

**Task: 05**

**Sir: Rasikh Ali**

**AI LAB TASK :5**

**Documentation for Depth-First Search (DFS) using Stack**

**Introduction**

This project demonstrates the implementation of **Depth-First Search (DFS)** traversal in a graph using an **iterative stack-based approach**. Unlike recursive DFS, which relies on function calls and system stack, this method uses a manually controlled stack. The traversal starts from a given node, explores as deep as possible along each path, and then backtracks when no further nodes are available. This ensures all reachable nodes are visited in a systematic order.

**Main Functions Used**

**Graph Representation**  
The graph is stored as an adjacency list, where each node maps to a list of its neighboring nodes. This makes it efficient to look up which nodes are directly connected.

**DFS Function (Stack-Based)**  
A visited list is maintained to keep track of nodes that have already been explored. A stack is used to decide the next node to process, following the Last-In-First-Out (LIFO) principle. Nodes are popped from the stack, checked if they are already visited, and if not, they are marked as visited. The neighbors of the current node are then added to the stack in reverse order to preserve the correct DFS sequence. Finally, the traversal order is returned once all nodes have been visited.

**Program Execution**  
The DFS traversal begins from a chosen starting node, such as node ‘A’. The result is the exact order in which the nodes are visited during the DFS traversal.

**Benefits**

* Avoids recursion depth limits, so it works even for large graphs.
* Provides more control and flexibility in traversal compared to recursion.
* Uses memory efficiently because it avoids extra function calls.
* Gives a clear and exact order of traversal, which is easy to understand.

**Problems Faced and Solutions**

One of the first problems faced was the issue of duplicate visits. In the beginning, the algorithm would sometimes visit the same node multiple times, which made the traversal incorrect. To solve this, a check was added to ensure that each node is only added to the visited list once. This fixed the problem and made the traversal accurate. The traversal order was wrong when neighbors were added normally. By adding them in reverse order, the leftmost node was visited first, giving the correct DFS path.