**Manual of code**

**Lab 05**

**Explanation of the Code**

This code implements a Depth-First Search (DFS) algorithm to traverse a graph. DFS is a method for exploring a graph by moving as far as possible along each branch before backtracking. The code uses a stack to keep track of nodes to visit next, which is a common approach for implementing DFS iteratively (without recursion).

**How the Code Works**

1. The dfs Function

The dfs function performs the Depth-First Search. It takes two inputs:

* graph: A dictionary representing the graph. Each key is a node, and the value is a list of its neighboring nodes.
* start: The starting node for the traversal.

****

**2. Data Structures Used**

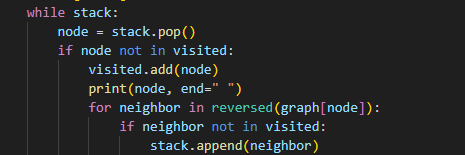
* visited: A set to keep track of nodes that have already been visited. This ensures that each node is processed only once.
* stack: A list that acts as a stack (Last-In-First-Out or LIFO) to store nodes that need to be visited.

****

**3. Traversing the Graph**

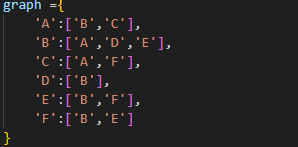
The algorithm works as follows:

1. Start by adding the start node to the stack.
2. While the stack is not empty:
   * Pop the top node from the stack.
   * If the node has not been visited:
     + Mark it as visited.
     + Print the node (or process it in some way).
     + Add its neighbors to the stack in reverse order (so that the first neighbor is processed last).

****

**4. The Graph**

The graph is represented as a dictionary where each key is a node, and the value is a list of its neighboring nodes. Here's the graph used in the code:

****

**}**

**5. Running the DFS**

The DFS is started from node 'A':

****

Step-by-Step Execution

Here’s what happens when the DFS is run:

1. Start at node 'A':
   * Mark 'A' as visited.
   * Add its neighbors 'B' and 'C' to the stack (in reverse order: ['C', 'B']).
2. Pop 'C' from the stack:
   * Mark 'C' as visited.
   * Add its neighbor 'F' to the stack.
3. Pop 'F' from the stack:
   * Mark 'F' as visited.
   * Add its neighbors 'E' and 'B' to the stack (in reverse order: ['B', 'E']).
4. Pop 'B' from the stack:
   * 'B' is already visited, so skip it.
5. Pop 'E' from the stack:
   * Mark 'E' as visited.
   * Add its neighbor 'B' to the stack (but 'B' is already visited, so it’s skipped).
6. Pop 'B' from the stack:
   * 'B' is already visited, so skip it.
7. Pop 'D' from the stack:
   * Mark 'D' as visited.
   * Add its neighbor 'B' to the stack (but 'B' is already visited, so it’s skipped).

**Output**

The output of the DFS traversal starting from node 'A' is:

****