Q1: Depth First Search (DFS) using Stack – Version 1

Objective:
To implement **Depth First Search** (DFS) using an explicit stack and track the path to a goal node.

Code Explanation:

1. The class `DepthFirstSearch` is initialized with a graph (dictionary of nodes and their children).
2. `dfs_stack(start, goal)` method:
    - Uses a `stack` to manage nodes and a `visited` list to track visited nodes.
    - Pops the last node from the stack, visits it if not already visited.
    - If it is the goal, prints confirmation and returns the visited path.
    - Otherwise, pushes children of the current node onto the stack in **reverse order** to maintain left-to-right visiting.
3. If the goal is not found after traversal, a message is displayed.

Output/Usage:

- For the given graph, starting from **A** and searching for **G**:
- `Visiting node: A`
- `Visiting node: C`
- `Visiting node: F`
- `Visiting node: G`
- `Goal 'G' found!`

  → Visited order: `['A', 'C', 'F', 'G']`.

Q2: Depth First Search (DFS) using Stack – Version 2

Objective:
To implement a simpler DFS with stack extension for traversal.

Code Explanation:

1. Similar to Version 1, but **without explicit printing of every node**.
2. Instead of pushing children in a loop, it extends the stack directly using `stack.extend(self.graph[node][::-1])`.
3. The method returns the visited list when the goal is found or after traversal ends.

Output/Usage:

- For the same graph, searching for **F**:

- `Visited: ['A', 'B', 'D', 'E', 'C', 'F']`
- The order shows DFS exploring left before right.

Q3: Tree Traversals – Preorder, Inorder, Postorder

Objective:
To explain the three common depth-first traversals of binary trees.

Explanation:

1. **Preorder Traversal (Root → Left → Right)**
    - Visit the root node first.
    - Traverse the left subtree.
    - Traverse the right subtree.
    - Example (for tree with root A): `A, B, D, E, C, F`.
    - **Use case:** Copying a tree.
2. **Inorder Traversal (Left → Root → Right)**
    - Traverse the left subtree first.
    - Visit the root node.
    - Traverse the right subtree.
    - Example: `D, B, E, A, F, C`.
    - **Use case:** Produces nodes in **sorted order** for binary search trees (BST).
3. **Postorder Traversal (Left → Right → Root)**
    - Traverse the left subtree.
    - Traverse the right subtree.
    - Visit the root node last.
    - Example: `D, E, B, F, C, A`.
    - **Use case:** Deleting/freeing a tree, evaluating expressions.