**Name : Ali Azhar**

**Roll No : SU92-BSAIM-S24-026**

**Section : 3A**

**Subject : Artificial Intelligence**

**Documentation of Task 1**

**“DFS with Stack and Node”**

**1. Introduction**

Depth-First Search (DFS) is a graph traversal algorithm used to explore nodes and edges in a **depthward motion**. It uses a **stack data structure** (either explicit or implicit via recursion) to track nodes.

**2. Graph Representation**

The graph is represented using an **adjacency list**, where each node is stored as a key, and its adjacent nodes are stored as a list of values.

**3. DFS Function Explanation**

The function performs a depth-first traversal starting from a given node and explores as far as possible along each branch before backtracking. The algorithm ensures that nodes are visited only once and maintains a stack to track traversal.

**4. User Input and Execution**

The user provides a **goal node**, and DFS starts from a root node, traversing until the goal is found or all nodes are explored.

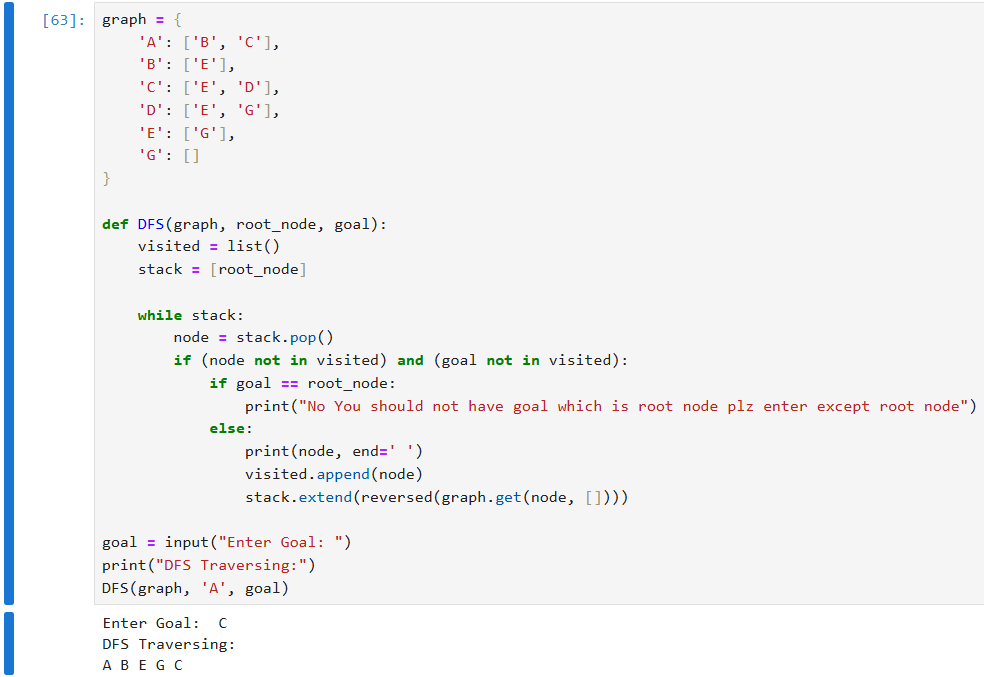
**5. Example Execution**

**Input :** The user enters a goal node to search for in the traversal process.

**Output :** DFS prints the path taken to reach the goal node.

**6. Conclusion**

DFS is a fundamental algorithm for graph traversal, useful in **pathfinding, maze solving, and network analysis**. It efficiently explores nodes using a stack-based approach and stops once the goal node is reached.



**Documentation of Task 2**

**“Pre, In, Post Orders”**

## ****1. Introduction****

Tree traversal is a method of visiting all the nodes in a tree data structure. The three primary traversal methods in **Depth-First Search (DFS)** are **Inorder, Preorder, and Postorder**.

## ****2. Inorder Traversal (Left → Root → Right)****

* Visits the **left subtree** first.
* Then visits the **root node**.
* Finally, visits the **right subtree**.
* Commonly used in **Binary Search Trees (BSTs)** to retrieve elements in sorted order.

### ****Example Order:****

If the tree structure is:

A

/ \

B C

Traversal order: **B A C**

## ****3. Preorder Traversal (Root → Left → Right)****

* Visits the **root node** first.
* Then visits the **left subtree**.
* Finally, visits the **right subtree**.
* Used in creating **copies of trees** and **prefix expression evaluation**.

### ****Example Order:****

A

/ \

B C

Traversal order: **A B C**

## ****4. Postorder Traversal (Left → Right → Root)****

* Visits the **left subtree** first.
* Then visits the **right subtree**.
* Finally, visits the **root node**.
* Used in **deleting trees** and **postfix expression evaluation**.

### ****Example Order:****

A

/ \

B C

Traversal order: **B C A**

## ****5. Comparison Table****

|  |  |
| --- | --- |
| **Traversal Type** | **Order of Visiting** |
| Inorder | Left → Root → Right |
| Preorder | Root → Left → Right |
| Postorder | Left → Right → Root |

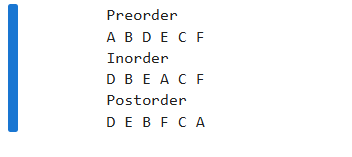
## ****6. Applications****

* **Inorder:** Retrieves sorted elements in BSTs.
* **Preorder:** Used in creating trees.
* **Postorder:** Helps in deleting or evaluating expressions.

## ****7. Conclusion****

These three traversal methods are fundamental in tree operations, providing different ways to process nodes effectively.

**OUTPUT :**

****