Approach to solve DFS

Call_DFS()

- Declare :
    - Visited vector with all elements as 0 of size number of nodes
    - Empty result vector
- Call for the DFS function as dfs(0,adj, visited, result)
- Return the result vector

DFS()

- Mark the passed node as visited
- Add the node to the result vector
- Traverse for the adjacent nodes of the source node :
    - If not visited then :
        - Call DFS(node, adj, visited, result)

Function Code

```
// DFS()
  void dfs(int node, vector<int> adj[], vector<int> &visited, vector<int>
&result) {
    // Marking the source node as visited
    visited[node] = 1;
    // Adding the source element to the result
    result.push_back(node);
    // Traversing for adjacent elements of the visited vector
    for (int i : adj[node]) {
        // checking if the node is visited or not
        if (!vis[i]) {
              // Calling the DFS of the node
            dfs(i, adj, vis, result);
            }
        }
    }

    // Call_DFS()
    // Function to return a list containing the DFS traversal of the graph.
    vector<int> dfsOfGraph(int n, vector<int> adj[]) {
        // Declaring

        // Visited vector having n elements of size n
```

```
        vector<int> visited(n,0);
         // Result vector
        vector<int> result;
         // Calling for the DFS of 0
        dfs(0,adj,visited,result);
         // Returning the result vector
        return result;

    }
```

Time Complexity

O(V+E)