DFS is a fundamental **graph traversal algorithm** used to explore **nodes** and **edges** of a graph systematically.

It starts at a designated **root node** and explores **as far as possible** along each branch before **backtracking**.

DFS is particularly useful in scenarios like:

- Find a path between two nodes.
- Checking if a graph contains any cycles.
- Identifying isolated subgraphs within a larger graph.
- Topological Sorting**:** Scheduling tasks without violating dependencies.

How DFS Works:

- **Start at the Root Node:** Mark the starting node as visited.
- **Explore Adjacent Nodes:** For each neighbor of the current node, do the following:
    - If the neighbor hasn't been visited, recursively perform DFS on it.
- **Backtrack:** Once all paths from a node have been explored, backtrack to the previous node and continue the process.
- **Termination:** The algorithm ends when all nodes reachable from the starting node have been visited.

Recursive DFS:

```
• void dfs_recursive(vector<vector<int>> &adj, int node, vector<bool> &visited):
•     visited[node] = true;
•     cout<<”visiting node ”<<node<<endl;
•     for(auto &nb : adj[node]){
•          if(vis[nb]==false){
•                  dfs_recursive(adj,nb,visited);
•           }
•     }
```

Iterative DFS:

```
void dfs_iterative(vector<vector<int>> &adj, int node,int n){
     vector<bool> vis(n,false);
     stack<int> st;
     st.push(node);
```

```
  while(!st.empty()){
       int u = st.pop();
      if(vis[u]==false){
          vis[u]=true;
          cout<<”visiting node “<<u<<endl;
          # Add neighbors to stack
          for(auto &nb : adj[u]){
               if(vis[nb]==false){
                   st.push(nb);
               }
          }
     }
}
```

Time Complexity: O(V + E), where V is the number of vertices and E is the number of edges. This is because the algorithm visits each vertex and edge once.
Space Complexity: O(V), due to stack used for recursion (in recursive implementation) or an explicit stack (in iterative implementation).

LeetCode Problems:

1. Path Sum II (LeetCode #113)
2. Clone Graph (LeetCode #133)
3. All Paths From Source to Target (LeetCode #797)
4. Time Needed to Inform All Employees (LeetCode #1376)
5. Longest Increasing Path in a Matrix (LeetCode #329)