### **AIM**

To implement the **Depth First Search (DFS)** algorithm using Python to traverse or search through nodes in a **graph** (represented using adjacency lists or matrices).

### **REQUIREMENTS**

* Python 3.x installed
* IDE or code editor (VSCode, PyCharm, etc.)
* Graph representation using adjacency list
* Understanding of recursion or stack

### **PROCEDURE / ALGORITHM**

1. **Represent the graph** using an adjacency list (dictionary in Python).
2. **Initialize**:  
   * A set to store visited nodes.
   * A recursive or iterative function for DFS traversal.
3. **DFS Algorithm (recursive)**:  
   * Start from a source node.
   * Mark the node as visited and process it (e.g., print).
   * Recur for all its unvisited adjacent vertices.
4. Stop when all reachable nodes are visited.

Code:

1. **Recursive DFS** (classic method )

def dfs\_recursive(graph, node, visited=None):

if visited is None:

visited = set()

if node not in visited:

print(node, end=" ")

visited.add(node)

for neighbor in graph[node]:

dfs\_recursive(graph, neighbor, visited)

# Sample graph (adjacency list)

graph = {

'A': ['B', 'C'],

'B': ['D', 'E'],

'C': ['F'],

'D': [],

'E': ['F'],

'F': []

}

def dfs\_iterative(graph, start):

visited = set()

stack = [start]

while stack:

node = stack.pop()

if node not in visited:

print(node, end=" ")

visited.add(node)

# Add neighbors in reverse for consistent order

stack.extend(reversed(graph[node]))

# Same graph

graph = {

'A': ['B', 'C'],

'B': ['D', 'E'],

'C': ['F'],

'D': [],

'E': ['F'],

'F': []

}

print("Iterative DFS starting from A:")

dfs\_iterative(graph, 'A')

print("Recursive DFS starting from A:")

dfs\_recursive(graph, 'A')

Output:

Recursive DFS starting from A:

A B D E F C

Iterative DFS starting from A:

A B D E F C