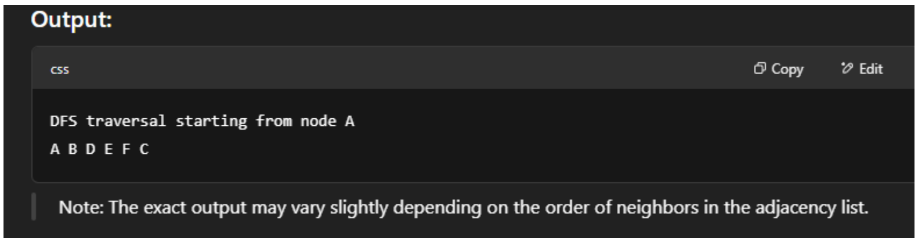
```
# Depth First Search using recursion on an undirected graph

# Create the graph as an adjacency list
graph = {
  'A': ['B', 'C'],
  'B': ['A', 'D', 'E'],
  'C': ['A', 'F'],
  'D': ['B'],
  'E': ['B', 'F'],
  'F': ['C', 'E']
}
# Set to keep track of visited nodes
visited = set()
# Recursive DFS function
def dfs(node):
  if node not in visited:
    print(node, end=' ')
    visited.add(node)
    for neighbor in graph[node]:
      dfs(neighbor)
# Start DFS from a given node
start_node = 'A'
print("DFS traversal starting from node", start_node)
dfs(start_node)
```



```
DFS traversal starting from node A
A B D E F C
```

Note: The exact output may vary slightly depending on the order of neighbors in the adjacency list.

Viva Questions for This Practical

1. **What is DFS (Depth First Search) and how does it work?**
   - Explain the general concept of DFS and how it explores all the vertices of a graph or tree by going as deep as possible before backtracking.
2. **What data structure is used to represent the graph in this code?**
   - The graph is represented as an adjacency list, where each node (vertex) points to a list of its neighbors.
3. **Why do we use a set to track visited nodes in DFS?**
   - A set is used to store visited nodes to ensure we don't revisit the same node, which could cause an infinite loop in case of cycles in the graph.
4. **Explain the base case of the recursive DFS function.**
   - The base case checks if the current node has already been visited. If so, it terminates further recursive calls for that node.
5. **What happens if a graph has cycles in it?**
   - If the graph has cycles, the DFS algorithm will still work correctly, as the visited set prevents revisiting the same node. Without this, DFS could enter an infinite loop.
6. **Can DFS be implemented iteratively?**
   - Yes, DFS can also be implemented using a stack (explicitly) instead of recursion. This avoids the limitations of recursion depth in certain cases.
7. **What is the time complexity of the DFS algorithm?**
   - The time complexity of DFS is **O(V + E)**, where V is the number of vertices and E is the number of edges in the graph.
8. **What is the space complexity of this DFS implementation?**
   - The space complexity is **O(V)** due to the visited set and the recursion stack.
9. **What will happen if you start the DFS traversal from a different node, say node 'C'?**
   - The traversal order will change depending on the neighbors and the graph's structure, but the algorithm will still visit all reachable nodes starting from 'C'.

10. **Can DFS be used in a directed graph?**

- Yes, DFS can be used on both directed and undirected graphs. The key difference is how edges are traversed; in a directed graph, the direction of the edges must be respected.

11. **What is the difference between DFS and BFS (Breadth First Search)?**
    - DFS explores as deep as possible along a branch before backtracking, whereas BFS explores all neighbors at the present depth level before moving on to nodes at the next depth level.
12. **How can you modify the DFS code to store the traversal path in a list rather than printing it?**
    - Instead of printing node, append the node to a list (e.g., path.append(node)).
13. **What happens if there are multiple connected components in the graph?**
    - In the case of multiple connected components, the DFS will only visit the nodes connected to the starting node. To visit all nodes, we would need to initiate DFS from each unvisited node.
14. **What will happen if the graph is empty?**
    - If the graph is empty, the DFS will not print anything as there are no nodes to visit.
15. **How would you handle a weighted graph with DFS?**
    - DFS can be applied to a weighted graph, but the weights are typically not considered in the basic DFS algorithm. You could modify the algorithm to account for weights if needed.