Breadth-First Search (BFS) and Depth-First Search (DFS)

Breadth-First Search (BFS) - Step by Step

Goal: Visit nodes level by level (i.e., visit all nodes at depth 1, then at depth 2, and so on).

Starting BFS from node 1:

1. **Initialize the Queue**:
    - Queue = [1]
    - Visited = {1}
2. **Process Node 1**:
    - Dequeue 1, print it: 1
    - Enqueue neighbors of 1 (nodes 2 and 3):
        - Queue = [2, 3]
        - Visited = {1, 2, 3}
3. **Process Node 2**:
    - Dequeue 2, print it: 2
    - Enqueue neighbors of 2 (nodes 4 and 5):
        - Queue = [3, 4, 5]
        - Visited = {1, 2, 3, 4, 5}
4. **Process Node 3**:
    - Dequeue 3, print it: 3
    - Enqueue the neighbor of 3 (node 6):
        - Queue = [4, 5, 6]
        - Visited = {1, 2, 3, 4, 5, 6}
5. **Process Node 4**:
    - Dequeue 4, print it: 4
    - Node 4 has no neighbors:
        - Queue = [5, 6]
        - Visited = {1, 2, 3, 4, 5, 6}
6. **Process Node 5**:
    - Dequeue 5, print it: 5
    - Node 5 has no neighbors:
        - Queue = [6]
        - Visited = {1, 2, 3, 4, 5, 6}
7. **Process Node 6**:
    - Dequeue 6, print it: 6
    - Node 6 has no neighbors:
        - Queue = []
        - Visited = {1, 2, 3, 4, 5, 6}

BFS Order: **1, 2, 3, 4, 5, 6**

Depth-First Search (DFS) - Step by Step

Goal: Visit nodes by going as deep as possible in the graph (i.e., move to one branch completely before backtracking).

Starting DFS from node 1:

1. **Initialize the Stack**:
    - Stack = [1]
    - Visited = {1}
2. **Process Node 1**:
    - Pop 1, print it: 1
    - Push the neighbors of 1 (nodes 3 and 2) onto the stack:
        - Stack = [3, 2]
        - Visited = {1}
3. **Process Node 2**:
    - Pop 2, print it: 2
    - Push the neighbors of 2 (nodes 5 and 4) onto the stack:
        - Stack = [3, 5, 4]
        - Visited = {1, 2}
4. **Process Node 4**:
    - Pop 4, print it: 4
    - Node 4 has no neighbors:
        - Stack = [3, 5]
        - Visited = {1, 2, 4}
5. **Process Node 5**:
    - Pop 5, print it: 5
    - Node 5 has no neighbors:
        - Stack = [3]
        - Visited = {1, 2, 4, 5}
6. **Process Node 3**:
    - Pop 3, print it: 3
    - Push the neighbor of 3 (node 6) onto the stack:
        - Stack = [6]
        - Visited = {1, 2, 3, 4, 5}
7. **Process Node 6**:
    - Pop 6, print it: 6
    - Node 6 has no neighbors:
        - Stack = []
        - Visited = {1, 2, 3, 4, 5, 6}

DFS Order: **1, 2, 4, 5, 3, 6**

Java Code for BFS

```
import java.util.*;

public class BFSExample {
    public static void bfs(int start, Map<Integer, List<Integer>> graph) {
        Queue<Integer> queue = new LinkedList<>();
        Set<Integer> visited = new HashSet<>();

        queue.add(start);
        visited.add(start);

        while (!queue.isEmpty()) {
            int node = queue.poll();
            System.out.print(node + " ");

            for (int neighbor : graph.get(node)) {
                if (!visited.contains(neighbor)) {
                    queue.add(neighbor);
                    visited.add(neighbor);
                }
            }
        }
    }

    public static void main(String[] args) {
        // Example graph
        Map<Integer, List<Integer>> graph = new HashMap<>();
        graph.put(1, Arrays.asList(2, 3));
        graph.put(2, Arrays.asList(4, 5));
        graph.put(3, Arrays.asList(6));
        graph.put(4, new ArrayList<>());
        graph.put(5, new ArrayList<>());
        graph.put(6, new ArrayList<>());

        System.out.println("BFS traversal starting from node 1:");
        bfs(1, graph);
    }
}
```

BFS traversal starting from node 1:
1 2 3 4 5 6

Java Code for DFS

```
import java.util.*;

public class DFSExample {
    public static void dfs(int start, Map<Integer, List<Integer>> graph) {
        Stack<Integer> stack = new Stack<>();
        Set<Integer> visited = new HashSet<>();

        stack.push(start);
        visited.add(start);

        while (!stack.isEmpty()) {
            int node = stack.pop();
            System.out.print(node + " ");

            for (int neighbor : graph.get(node)) {
                if (!visited.contains(neighbor)) {
                    stack.push(neighbor);
                    visited.add(neighbor);
                }
            }
        }
    }

    public static void main(String[] args) {
        // Example graph
        Map<Integer, List<Integer>> graph = new HashMap<>();
        graph.put(1, Arrays.asList(2, 3));
        graph.put(2, Arrays.asList(4, 5));
        graph.put(3, Arrays.asList(6));
        graph.put(4, new ArrayList<>());
        graph.put(5, new ArrayList<>());
        graph.put(6, new ArrayList<>());

        System.out.println("DFS traversal starting from node 1:");
        dfs(1, graph);
    }
}
```

DFS traversal starting from node 1:
1 3 6 2 5 4