Clone Graph Documentation

Problem Statement

You are given a reference to a node in a connected undirected graph. The goal is to return a deep copy (clone) of the graph. Each node in the graph contains an integer value (val) and a list of its neighbors (neighbors).

Node Class Definition

```
class Node {
   public int val;
   public List<Node> neighbors;
}
```

Input

- The graph is represented in the test case using an adjacency list.
- Each node's value corresponds to its index (1-indexed) for simplicity. For example, the node with val = 1 is the first node, the node with val = 2 is the second node, and so on.
- The given node will always be the first node with val = 1.

Output

- The function should return a reference to the deep copy of the given node, which represents the cloned graph.

<u>Test Case Format</u>

- The graph is represented using an adjacency list, which is a collection of unordered lists. Each list describes the set of neighbors of a node in the graph.

<u>Example 1</u>

Input: adjList = [[2,4],[1,3],[2,4],[1,3]]

Output: [[2,4],[1,3],[2,4],[1,3]]

Explanation:

There are 4 nodes in the graph.

- The 1st node (val = 1) has neighbors 2nd node (val = 2) and 4th node (val = 4).
- The 2nd node (val = 2) has neighbors 1st node (val = 1) and 3rd node (val = 3).
- The 3rd node (val = 3) has neighbors 2nd node (val = 2) and 4th node (val = 4).
- The 4th node (val = 4) has neighbors 1st node (val = 1) and 3rd node (val = 3).

<u>Example 2</u>

Input: adjList = [[]]

Output: [[]]

Explanation: The input contains one empty list. The graph consists of only one node with val = 1 and it does not have any neighbors.

Example 3

Input: adjList = []

Output: []

Explanation: This is an empty graph, it does not have any nodes.

Constraints

- The number of nodes in the graph is in the range [0, 100].
- (1 leq text{Node.val} leq 100)
- Node.val is unique for each node.
- There are no repeated edges and no self-loops in the graph.
- The graph is connected, and all nodes can be visited starting from the given node.

Approach

To clone the graph, we can use a Depth-First Search (DFS) approach. The steps involved are:

1. Check for Empty Input: Return None if the input node is None.
2. Dictionary for Clones: Use a dictionary to keep track of the cloned nodes.
3. DFS Function: Define a recursive DFS function to clone the nodes.
    - If a node has already been cloned, return the clone from the dictionary.
    - Otherwise, clone the node and add it to the dictionary.
    - Iterate through the neighbours of the node, recursively clone them, and add the clones to the neighbours of the cloned node.
4. Return Cloned Graph: Start the DFS from the given node and return the cloned graph.