133. 克隆图

给你无向 **连通** 图中一个节点的引用，请你返回该图的 **深拷贝**（克隆）。

图中的每个节点都包含它的值 `val`（`int`） 和其邻居的列表（`list[Node]`）。

```
class Node {

    public int val;

    public List<Node> neighbors;

}
```

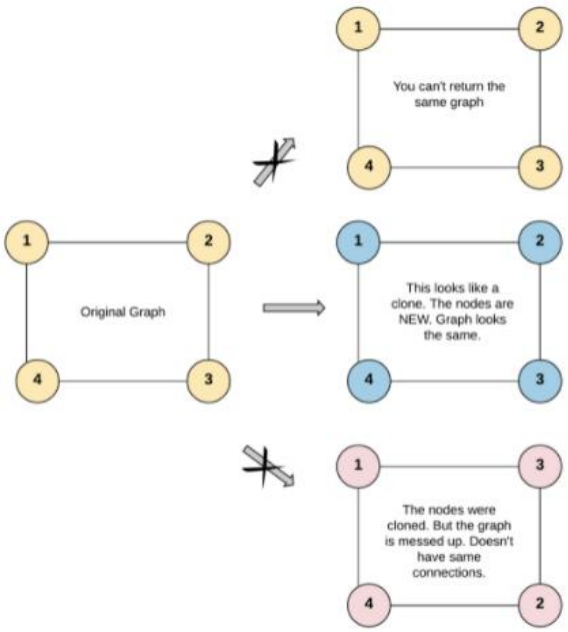
测试用例格式：

简单起见，每个节点的值都和它的索引相同。例如，第一个节点值为 1（`val = 1`），第二个节点值为 2（`val = 2`），以此类推。该图在测试用例中使用邻接列表表示。

邻接列表 是用于表示有限图的无序列表的集合。每个列表都描述了图中节点的邻居集。

给定节点将始终是图中的第一个节点（值为 1）。你必须将 **给定节点的拷贝** 作为对克隆图的引用返回。

示例 1：



输入： `adjList = [[2,4],[1,3],[2,4],[1,3]]`

输出： `[[2,4],[1,3],[2,4],[1,3]]`

解释：

图中有 4 个节点。

节点 1 的值是 1，它有两个邻居：节点 2 和 4 。

节点 2 的值是 2，它有两个邻居：节点 1 和 3 。

节点 3 的值是 3，它有两个邻居：节点 2 和 4 。

节点 4 的值是 4，它有两个邻居：节点 1 和 3 。

示例 2：

输入：adjList = [[]]

输出：[[]]

解释：输入包含一个空列表。该图仅仅只有一个值为 1 的节点，它没有任何邻居。

示例 3：

输入：adjList = []

输出：[]

解释：这个图是空的，它不含任何节点。

示例 4：

输入：adjList = [[2],[1]]

输出：[[2],[1]]

"""

```
# Definition for a Node.
class Node:
    def __init__(self, val = 0, neighbors = None):
        self.val = val
        self.neighbors = neighbors if neighbors is not None else []
"""

class Solution:
    def cloneGraph(self, node: 'Node') -> 'Node':
        map_list={}#字典类型 key 为旧节点 value 为新克隆节点

        def dfs(node):
            if not node:
                return node
             if node in map_list:#如果该节点被遍历过 已经存储在maplist中 则直接
取出返回
                return map_list[node]
            #如果没有被遍历到的节点 需要克隆复制出相同值的新节点
            new_node=Node(node.val,[])
            #新克隆节点需记录在 maplist 中表示已经遍历过了
            map_list[node]=new_node
            #递归克隆旧节点的邻居节点
            for n in node.neighbors:
                new_node.neighbors.append(dfs(n))
            return new_node

        return dfs(node)
```