116. Populating Next Right Pointers in Each Node

You are given a **perfect binary tree** where all leaves are on the same level, and every parent has two children. The binary tree has the following definition:

```
struct Node {
  int val;
  Node *left;
  Node *right;
  Node *next;
}
```

Populate each next pointer to point to its next right node. If there is no next right node, the next pointer should be set to `NULL`.

Initially, all next pointers are set to `NULL`.

Example 1:

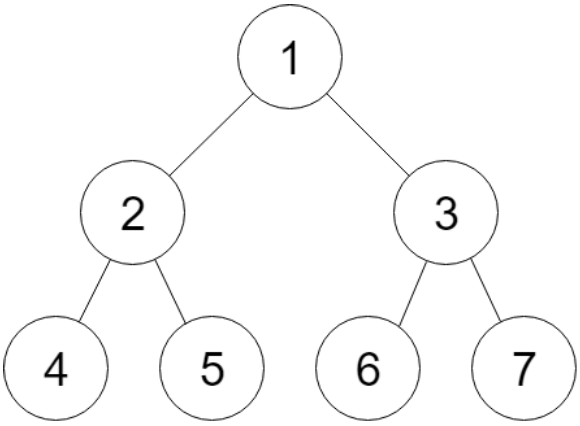

Figure A

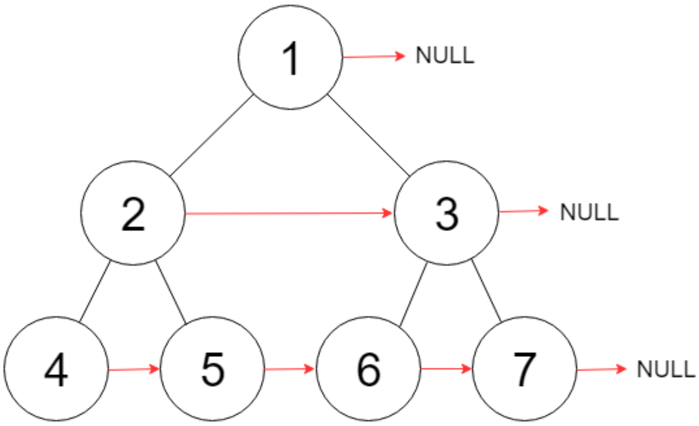

Figure B

```
Input: root = [1,2,3,4,5,6,7]
Output: [1,#,2,3,#,4,5,6,7,#]
Explanation: Given the above perfect binary tree (Figure A), your function
should populate each next pointer to point to its next right node, just like
in Figure B. The serialized output is in level order as connected by the
next pointers, with '#' signifying the end of each level.
```

Example 2:

```
Input: root = []
Output: []
```

Constraints:

- The number of nodes in the tree is in the range `[0, 2<sup>12</sup> - 1]`.
- `-1000 <= Node.val <= 1000`

Follow-up:

- You may only use constant extra space.
- The recursive approach is fine. You may assume implicit stack space does not count as extra space for this problem.

```
class Solution:
    def connect(self, root: 'Node') -> 'Node':
        if root is None:
            return root
        stack = [root]

        while len(stack)>0:
            size = len(stack)

            while size>0:
                temp = stack.pop(0)
                if temp.left:
                    stack.append(temp.left)
                if temp.right:
                    stack.append(temp.right)

                size-=1
                if size==0:
                    temp.next = None
                else:
                    temp.next = stack[0]
        return root
```

```
def connect(self, root: 'Node') -> 'Node':
        if root is None:
            return root
        black = root

        while black!=None and black.left!=None:

            n = black

            while True:
                n.left.next = n.right

                if n.next==None:
                    break
                n.right.next = n.next.left
                n = n.next

            black = black.left

        return root
```

```
def connect(self, root):
        if not root or not root.left: return root

        self.connect(root.left)
        self.connect(root.right)

        lft = root.left
        rgh = root.right
        lft.next = rgh

        while lft.right:
            lft = lft.right
            rgh = rgh.left
            lft.next = rgh

        return root
```