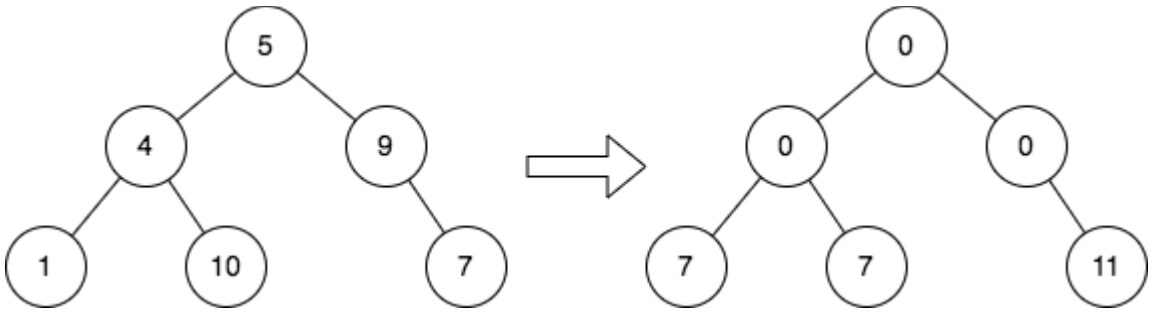
2641. Cousins in Binary Tree II

Given the `root` of a binary tree, replace the value of each node in the tree with the **sum of all its cousins' values**.

Two nodes of a binary tree are **cousins** if they have the same depth with different parents.

Return *the* `root` *of the modified tree*.

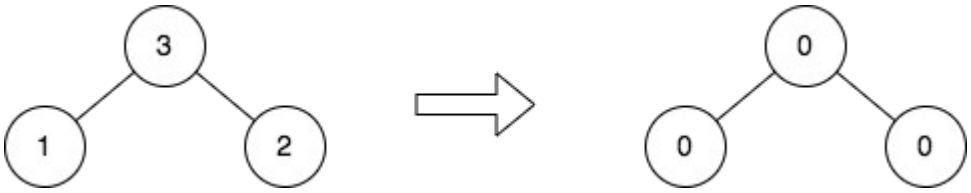
Note that the depth of a node is the number of edges in the path from the root node to it.



Example 1:

```
Input: root = [5,4,9,1,10,null,7]
Output: [0,0,0,7,7,null,11]
Explanation: The diagram above shows the initial binary tree and the binary tree
after changing the value of each node.
- Node with value 5 does not have any cousins so its sum is 0.
- Node with value 4 does not have any cousins so its sum is 0.
- Node with value 9 does not have any cousins so its sum is 0.
- Node with value 1 has a cousin with value 7 so its sum is 7.
- Node with value 10 has a cousin with value 7 so its sum is 7.
- Node with value 7 has cousins with values 1 and 10 so its sum is 11.
```

Example 2:




```
Input: root = [3,1,2]
Output: [0,0,0]
Explanation: The diagram above shows the initial binary tree and the binary tree
after changing the value of each node.
- Node with value 3 does not have any cousins so its sum is 0.
- Node with value 1 does not have any cousins so its sum is 0.
- Node with value 2 does not have any cousins so its sum is 0.
```

Constraints:

- The number of nodes in the tree is in the range `[1, 105]`.
- `1 <= Node.val <= 104`

2641. Cousins in Binary Tree II

```
/*
   Two passes BFS (level order traversal)
   Time complexity: Ω(n),O(n)
   Space complexity: Ω(n+h),O(n+n)
    n : total number of nodes in the tree
    h : height of the binary tree= log2(n)
*/
class Solution{
   public:
      TreeNode* replaceValueInTree(TreeNode* root){
         if(!root) return root;

         std::queue<TreeNode*> q;
         std::vector<long long> levels_sums;

         // Pass1: compute the sum of each level
         q.push(root);

         while(!q.empty()){
            int cur_subtree_size=q.size();
            long long s=0;
            while(cur_subtree_size>0){
               TreeNode* cur_node=q.front();
               q.pop();

               cur_subtree_size--;

               s+=cur_node->val;

               if(cur_node->left) q.push(cur_node->left);
               if(cur_node->right) q.push(cur_node->right);
            }

            levels_sums.push_back(s);
         }
```

```
        // Pass2:
        q.push(root);
        int level=1;
        root->val=0;
        while(!q.empty()){
            int cur_subtree_size=q.size();
            while(cur_subtree_size>0){
                TreeNode* cur_node=q.front();
                q.pop();

                cur_subtree_size--;

                int brothers_sums=0;
                if(cur_node->left) brothers_sums=cur_node->left->val;
                if(cur_node->right) brothers_sums+=cur_node->right->val;

                if(cur_node->left) cur_node->left->val=levels_sums[level]-
brothers_sums,q.push(cur_node->left);
                if(cur_node->right) cur_node->right->val=levels_sums[level]-
brothers_sums,q.push(cur_node->right);
            }

            level++;
        }


        return root;
    }
};
```

2641. Cousins in Binary Tree II

```
/*
   One passe BFS (level order traversal)
   Time complexity: Ω(n),O(n)
   Space complexity: Ω(n),O(n)
   n: total number of nodes in the tree
*/
class Solution{
   public:
      TreeNode* replaceValueInTree(TreeNode* root){
         if(!root) return root;
         std::queue<TreeNode*> q;
         q.push(root);
         int prev_level_sum=root->val;
         while(!q.empty()){
            int cur_subtree_size=q.size();
            long long cur_level_sum=0;
            while(cur_subtree_size>0){
               TreeNode* cur_node=q.front();
               q.pop();
               cur_subtree_size--;

               // Update node value to cousin sum.
               cur_node->val=prev_level_sum-cur_node->val;

               // Compute the sum for both the brothers
               int brothers_sum=0;
               if(cur_node->left) brothers_sum=cur_node->left->val;
               if(cur_node->right) brothers_sum+=cur_node->right->val;

               if(cur_node->left){
                  cur_level_sum += cur_node->left->val; // Accumulate current level sum.
                  cur_node->left->val=brothers_sum; // Update left child's value.
                  q.push(cur_node->left);  // Add to queue for next level.
               }
               if(cur_node->right){
                  cur_level_sum += cur_node->right->val; // Accumulate current level sum.
                  cur_node->right->val=brothers_sum; // Update left child's value.
                  q.push(cur_node->right);  // Add to queue for next level.
               }
            }
            prev_level_sum=cur_level_sum; // Update previous level for next iteration
         }
         return root;
      }
};
```