116. Populating Next Right Pointers in Each Node

Medium 2941 158 Add to List Share

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`.

Follow up:

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

Example 1:

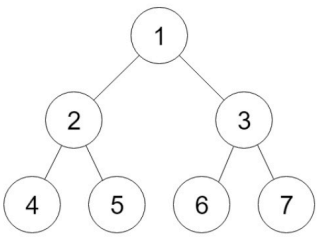


Figure A

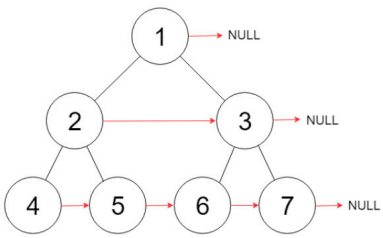
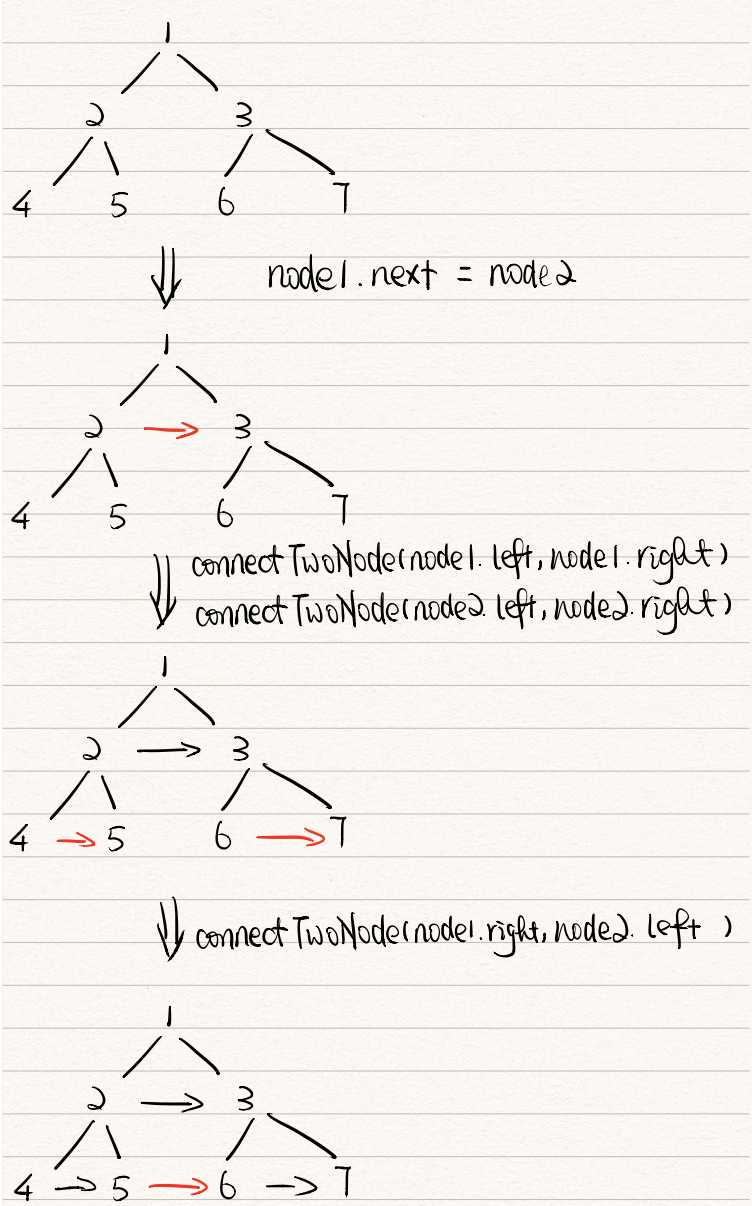


Figure B

```
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.
```

Constraints:

- The number of nodes in the given tree is less than `4096`.
- `-1000 <= node.val <= 1000`



```cpp
/*
// Definition for a Node.
class Node {
public:
    int val;
    Node* left;
    Node* right;
    Node* next;

    Node() : val(0), left(NULL), right(NULL), next(NULL) {}

    Node(int _val) : val(_val), left(NULL), right(NULL), next(NULL) {}

    Node(int _val, Node* _left, Node* _right, Node* _next)
        : val(_val), left(_left), right(_right), next(_next) {}
};
*/

class Solution {
public:
    Node* connect(Node* root) {
        // base case
        if (root == NULL) {
            return NULL;
        }

        connectTwoNode(root->left, root->right);

        return root;
    }

private:
    void connectTwoNode(Node* node1, Node* node2) {
        if (node1 == NULL || node2 == NULL) {
            return;
        }

        // connect the two node
        node1->next = node2;

        // deal with the chile node recursively
        connectTwoNode(node1->left, node1->right);
        connectTwoNode(node2->left, node2->right);

        // connect the two node cross the parent node
        connectTwoNode(node1->right, node2->left);
    }

};
```