116. Populating Next Right Pointers in Each Node

Given a binary tree

```
    struct TreeLinkNode {
      TreeLinkNode *left;
      TreeLinkNode *right;
      TreeLinkNode *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`.

Note:

- You may only use constant extra space.
- You may assume that it is a perfect binary tree (ie, all leaves are at the same level, and every parent has two children).

For example, Given the following perfect binary tree,

```
         1
       /  \
      2    3
     / \  / \
    4  5  6  7
```

After calling your function, the tree should look like:

```
         1 -> NULL
       /  \
      2 -> 3 -> NULL
     / \  / \
    4->5->6->7 -> NULL
```

树是完全二叉树的，但是要做的是，把每个节点的next指针指向它的兄弟。如果到右没有的话，就将next -> NULL

我最先想到的是层序遍历，不妨先试试。

这种方法会超时。

```
/**
 * Definition for binary tree with next pointer.
 * struct TreeLinkNode {
```

```
 *  int val;
 *  TreeLinkNode *left, *right, *next;
 *  TreeLinkNode(int x) : val(x), left(NULL), right(NULL), next(NULL) {}
 * };
 */
class Solution {
public:
    void connect(TreeLinkNode *root) {
        if( root == NULL ) return;

        queue<TreeLinkNode* > queue;
        queue.push(root);

        int level_num = 1;
        while(root != NULL || !queue.empty() ){
            int level_tmp = 0;

            /// 把下一层的节点依次加入队列
            for( int i=0; i <level_num; i++ ){
                if( queue.front()->left != NULL ){
                    ++level_tmp;
                    queue.push( queue.front()->left );
                }
                if( queue.front()->right != NULL ){
                    ++level_tmp;
                    queue.push( queue.front()->right );
                }

                 TreeLinkNode* pre = queue.front(); queue.pop();

                if( i == level_num -1 ) // last
                    pre -> next = NULL;
                else
                    pre -> next = queue.front();
            }

//             /// 处理每一层的
//             for( int i=0; i< level_num; i++ ){
//                 TreeLinkNode* pre = queue.front(); queue.pop();

//                 if( i == level_num -1 ) // last
//                     pre -> next = NULL;
//                 else
//                     pre -> next = queue.front();
//             }

            level_num = level_tmp;
        }
    }
};
```

这儿有个更好的方法

```
    void connect(TreeLinkNode *root) {
    if (root == NULL) return;
    TreeLinkNode *pre = root;
    TreeLinkNode *cur = NULL;
    while(pre->left) {
        cur = pre;      /// 父节点
        while(cur) {
            cur->left->next = cur->right;   /// 处理左右节点
            if(cur->next) cur->right->next = cur->next->left;   /// 如果父有兄弟节点，处理父的兄弟的孩子节点
            cur = cur->next;                                    /// 处理父的兄弟
        }
        pre = pre->left;                                        /// 左子树继续向下走
    }
}
```