116. Populating Next Right Pointers in Each Node

link

递归的写法很简单, 分两步, 一步用连接child, 然后根据next连接child的子节点.

```
    public Node connect(Node root) {
        first(root);
        second(root);
        return root;
    }
    public void first(Node node){
        if(node == null || node.left == null) return;
        node.left.next = node.right;
        first(node.left);
        first(node.right);
    }
    public void second(Node node){
        if(node == null || node.left == null) return;
        if(node.next != null){
            node.right.next = node.next.left;
        }
        second(node.left);
        second(node.right);
    }
//两次可以放一起
    public void connect(Node node){
        if(node == null || node.left == null) return;
            node.left.next = node.right;
        if(node.next != null)
            node.right.next = node.next.left;
        first(node.left);
        first(node.right);
    }
```

非递归, 从左到右一层一层遍历, 先写三层if, 然后第四层把if换成while

```
    public Node helper(Node node){
        if(node == null) return null;
        Node current = node;
        while(current.left != null){
            current.left.next = current.right;
            Node next = current.next;
            Node tmp = current;
            while(next != null ){
                tmp.right.next = next.left;
```

```
                next.left.next = next.right;
                tmp = next;
                next = next.next;
            }
            current = current.left;
        }
        return node;
    }
```

117. Populating Next Right Pointers in Each Node II

这里改了好几遍都改错了, 因为这里逻辑root.left需要root.right的next, 否则后面会有节点连不上

```
      public Node helper(Node root){
        if(root == null) return root;

        if(root.left != null && root.right != null){
            root.left.next = root.right;
        }
        if(root.next != null){
            if(root.right != null){
                root.right.next = getNext(root.next);
            }else if(root.left != null){
                root.left.next = getNext(root.next);
            }
        }
//这里左边的节点build会依靠右边的next, 所以必须先helper(root.right)
        helper(root.right);
        helper(root.left);
        return root;
    }
    public Node getNext(Node next){
        if(next == null) return null;
        if(next.left != null) return next.left;
        if(next.right != null) return next.right;
        return getNext(next.next);
    }
```

这么写可读性更高

```
    void connect(TreeLinkNode *root) {
        if (root == NULL) return;
        if (root->left) {
            if (root->right) {
                root->left->next = root->right;
```

```
            } else {
                root->left->next = fnext(root->next);
            }
        }

        if (root->right) {
            root->right->next = fnext(root->next);
        }

        connect(root->right);
        connect(root->left);
    }
```

真正的const space用循环的形式, 难点是在如何确定循环的条件, 比方说左子树没有子节点, 但是右子树有, 这里循环条件丢失. 考虑使用dummy head 如果父亲有next用父亲的next来判断.

```
Node dummy = new Node(0);
Node head = dummy;
while(node.next != null){
//这样就省略了复杂的判断条件, 并且head.next就是每一层bfs的第一个节点, 并且如果node.left和
    node.right都是null, dummy没有变.
//linkedlist里面的dummy头节点.
    if(node.left != null){
        dummy.next = node.left;
        dummy = node.left;
    }
    if(node.right != null){
        dummy.next = node.right;
        dummy = node.right;
    }
    node = node.next;
}
```