树的变形篇

LeetCode 中此篇的题目有以下几道：

1. Populating Next Right Pointers in Each Node
2. Populating Next Right Pointers in Each Node II
3. Flatten Binary Tree to Linked List

1. Populating Next Right Pointers in Each Node，这道题是要将一棵树的每一层维护成一个链表，树本身是给定的。其实思路上很接近层序遍历 Binary Tree Level Order Traversal，只是这里不需要额外维护一个队列。因为这里每一层我们会维护成一个链表，这个链表其实就是每层起始的那个队列，因此我们只需要维护一个链表的起始指针就可以依次得到整个队列了。接下来就是有左加左入链表，有右加右入链表，直到链表没有元素了说明到达最底层了。算法的复杂度仍然是对每个结点访问一次，所以是 O(n)，而空间上因为不需要额外空间来存储队列了，所以是 O(1)。

代码如下：

```
public void connect(TreeLinkNode root) {
    TreeLinkNode curHead = root;
    TreeLinkNode nextHead = null;
    TreeLinkNode pre = null;
    while(curHead != null){
        TreeLinkNode cur = curHead;
        while(cur != null){
            if(cur.left != null){
                if(nextHead == null){
                    nextHead = cur.left;
                    pre = nextHead;
                } else {
                    pre.next = cur.left;
                    pre = pre.next;
                }
            }
            if(cur.right != null){
                if(nextHead == null){
                    nextHead = cur.right;
                    pre = nextHead;
```

```
                } else {
                    pre.next = cur.right;
                    pre = pre.next;
                }
            }
            cur = cur.next;
        }
        curHead = nextHead;
        nextHead = null;
    }
}
```

做题时的感悟:

1. 如果nextHead为空，说明下一行的链表还没有表头，这时就把第一个子结点赋值给nextHead，把nextHead作为pre。这样思路清晰一些。代码如下:

```
if(cur.left != null){
    if(nextHead == null){
        nextHead = cur.left;
        pre = nextHead;
    } else {
        pre.next = cur.left;
        pre = pre.next;
    }
}
```

2. 做这种题，一定要搞清楚指针的移动，何时清空。不然很有可能出现不循环或者死循环。

在外层循环：只要还有链表，即curHead != null，就要继续循环。

在内层循环：cur从curHead即该链表的头部开始，cur = cur.next;循环至该链表结束。循环的同时连接下层链表。

在外层循环：将下层链表头赋值给当前链表头，curHead = nextHead；注意，这里还要清空下层链表头指针，即nextHead = null。不然nextHead永远不会更新，就会出现死循环。

2. Populating Next Right Pointers in Each Node II,这道题目的要求和Populating Next Right Pointers in Each Node 是一样的，只是这里的二叉树没要求是完全二叉树。其实在实现 Populating Next Right Pointers in Each Node 的时候我们已经兼容了不是完全二叉树的情况，其实也比较好实现，就是在判断队列结点时判断一下他的左右结点是否存在就可以了。时间复杂度和空间复杂度不变，还是 O(n)和 O(1)。
代码与 Populating Next Right Pointers in Each Node 完全相同。

3. Flatten Binary Tree to Linked List，这是树的题目，要求把一颗二叉树按照先序遍历顺序展成一个链表，不过这个链表还是用树的结构，就是一直往右走（没有左孩子）来模拟链表。老套路还是用递归来解决，维护先序遍历的前一个结点pre，然后每次把pre的左结点置空，右结点设为当前结点。这里需要注意的一个问题就是我们要先把右子结点保存一下，以便等会可以进行递归，否则有可能当前结点的右结点会被覆盖，后面就取不到了。算法的复杂度时间上还是一次遍历，O(n)。空间上是栈的大小，O(logn)。
树的题目讨论得比较多了，主要思路就是递归，考虑好递归条件和结束条件，有时候如果递归过程会对树结构进行修改的话，要先保存一下结点。
代码如下：

```
public void flatten(TreeNode root) {
    ArrayList<TreeNode> pre = new ArrayList<TreeNode>();
    pre.add(null);
    helper(root, pre);
}

private void helper(TreeNode root, ArrayList<TreeNode> pre){
    if(root == null){
        return;
    }
    TreeNode right = root.right;
    if(pre.get(0) != null){
        pre.get(0).left = null;
        pre.get(0).right = root;
    }
    pre.set(0, root);
    helper(root.left, pre);
    helper(right, pre);
}
```

做题时的感悟:

1. 这道题要以一个只有右子结点的树来模拟链表，所以我们要将以root为根节点的左子树移接到pre的右孩子处，由于可能会覆盖掉pre的右孩子，所以要预先保存pre的右孩子以便后面继续递归。

*在pre为当前root的时候保存pre的右结点。

2. 本质上就是一次先序遍历，利用pre来进行树变一维链表的操作。