统计完全二叉树的节点数

【题目】

给定一棵完全二叉树的头节点 head，返回这棵树的节点个数。

【要求】

如果完全二叉树的节点数为 N，请实现时间复杂度低于 $O(N)$的解法。

【难度】

尉 ★★☆☆

【解答】

遍历整棵树当然可以求出节点数，但这肯定不是最优解法，本书不再详述。

如果完全二叉树的层数为 h，本书的解法可以做到时间复杂度为 $O(h^2)$，具体过程如下：

1．如果 head==null，说明是空树，直接返回 0。

2．如果不是空树，就求树的高度，求法是找到树的最左节点看能到哪一层，层数记为 h。

3．这一步是求解的主要逻辑，也是一个递归过程记为 bs(node,l,h)，node 表示当前节点，l 表示 node 所在的层数，h 表示整棵树的层数是始终不变的。bs(node,l,h)的返回值表示以 node 为头的完全二叉树的节点数是多少。初始时 node 为头节点 head，l 为 1，因为 head 在第 1 层，一共有 h 层始终不变。那么这个递归的过程可以用两个例子来说明，如图 3-50 和图 3-51 所示。

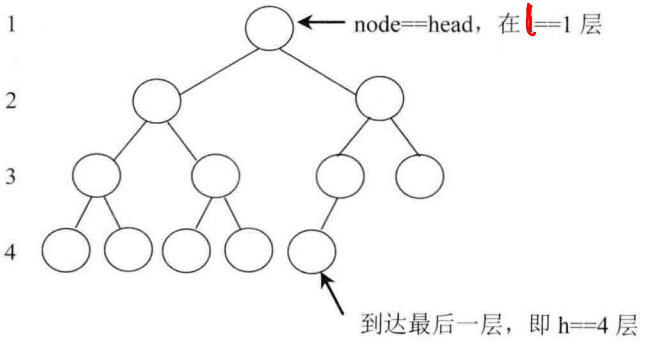

图 3-50

找到 node 右子树的最左节点，如果像图 3-50 的例子一样，发现它能到达最后一层，即 h==4 层。此时说明 node 的整棵左子树都是满二叉树，并且层数为 $h-l$ 层，一棵层数为 $h-l$ 的满二叉树，其节点数为 $2^{h-l}-1$ 个。如果加上 node 节点自己，那么节点数为 2^(h-l)-1+1==2^(h-l)个。此时如果再知道 node 右子树的节点数，那么以 node 为头的完全二叉树上到底有多少个节点就求出来了。那么 node 右子树的节点数到底是多少呢？就是 bs(node.right,l+1,h)的结果，递归去求即可。最后整体返回 2^(h-l)+bs(node.right,l+1,h)。

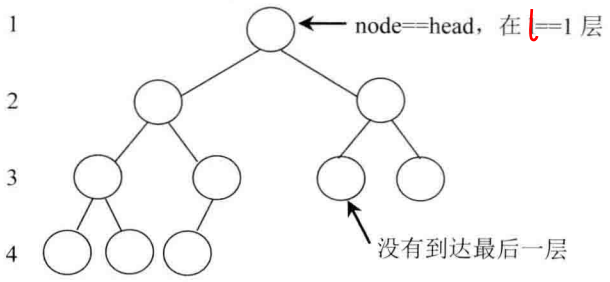

图 3-51

找到 node 右子树的最左节点，如果像图 3-51 的例子一样，发现它没有到达最后一层，说明 node 的整棵右子树都是满二叉树，并且层数为 $h-l-1$ 层，一棵层数为 $h-l-1$ 的满二叉树，其节点数为 $2^{h-l-1}-1$ 个。如果加上 node 节点自己，那么节点数为 2^(h-l-1)-1+1==2^(h-l,-1)个。此时如果再知道 node 左子树的节点数，那么以 node 为头的完全二叉树上到底有多少个节点就求出来了。node 左子树的节点数到底是多少呢？就是 bs(node.left,l+1,h)的结果，递归去求即可，最后整体返回 2^(h-l-1)+bs(node.left,l+1,h)。

全部过程请参看如下代码中的 nodeNum 方法。

```
public int nodeNum(Node head) {
        if (head == null) {
                return 0;
        }
        return bs(head, 1, mostLeftLevel(head, 1));
}

public int bs(Node node, int l, int h) {
        if (l == h) {
                return 1;
        }
        if (mostLeftLevel(node.right, l + 1) == h) {
                return (1 << (h - l)) + bs(node.right, l + 1, h);
        } else {
                return (1 << (h - l - 1)) + bs(node.left, l + 1, h);
        }
```

```
    }

    public int mostLeftLevel(Node node, int level) { 最左子树的深度
            while (node != null) {
                    level++;
                    node = node.left;
            }
            return level - 1;
    }
```

每一层只会选择一个节点 node 进行 bs 的递归过程，所以调用 bs 函数的次数为 $O(h)$。每次调用 bs 函数时，都会查看 node 右子树的最左节点，所以会遍历 $O(h)$个节点，整个过程的时间复杂度为 $O(h^2)$。