```
                if (head == null) {
                        return;
                }
                setAnswers(head.left, ans);
                sets.union(head.left, head);
                ancestorMap.put(sets.findFather(head), head);
                setAnswers(head.right, ans);
                sets.union(head.right, head);
                ancestorMap.put(sets.findFather(head), head);
                LinkedList<Node> nList = queryMap.get(head);
                LinkedList<Integer> iList = indexMap.get(head);
                Node node = null;
                Node nodeFather = null;
                int index = 0;
                while (nList != null && !nList.isEmpty()) {
                        node = nList.poll();
                        index = iList.poll();
                        nodeFather = sets.findFather(node);
                        if (ancestorMap.containsKey(nodeFather)) {
                                ans[index] = ancestorMap.get(nodeFather);
                        }
                }
        }

}
```

二叉树节点间的最大距离问题

【题目】

从二叉树的节点 A 出发，可以向上或者向下走，但沿途的节点只能经过一次，当到达节点 B 时，路径上的节点数叫作 A 到 B 的距离。

比如，图 3-48 所示的二叉树，节点 4 和节点 2 的距离为 2，节点 5 和节点 6 的距离为 5。给定一棵二叉树的头节点 head，求整棵树上节点间的最大距离。

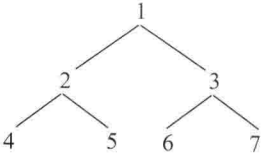

图 3-48

【要求】

如果二叉树的节点数为 N，时间复杂度要求为 $O(N)$。

【难度】

尉 ★★☆☆

【解答】

一个以 h 为头的树上，最大距离只可能来自以下三种情况：

- h 的左子树上的最大距离。
- h 的右子树上的最大距离。
- h 左子树上离 h.left 最远的距离 + 1(h) + h 右子树上离 h.right 最远的距离。

三个值中最大的那个就是整棵 h 树中最远的距离。

根据如上分析，设计解法的过程如下：

1．整个过程为后序遍历，在二叉树的每棵子树上执行步骤 2。

2．假设子树头为 h，处理 h 左子树，得到两个信息，左子树上的最大距离记为 lMax，左子树上距离 h 左孩子的最远距离记为 maxfromLeft。同理，处理 h 右子树得到右子树上的最大距离记为 rMax 和距离 h 右孩子的最远距离记为 maxFromRight。那么 maxfromLeft + 1 + maxFromRight 就是跨 h 节点情况下的最大距离，再与 lMax 和 rMax 比较，把三者中的最值作为 h 树上的最大距离返回，maxfromLeft+1 就是 h 左子树上离 h 最远的点到 h 的距离，maxFromRight+1 就是 h 右子树上离 h 最远的点到 h 的距离，选两者中最大的一个作为 h 树上距离 h 最远的距离返回。如何返回两个值？一个正常返回，另一个用全局变量表示。

具体过程请参看如下代码中的 maxDistance 方法，其中，record[0]就表示另一个返回值。

```
public int maxDistance(Node head) {
        int[] record = new int[1];
        return posOrder(head, record);
}

public int posOrder(Node head, int[] record) {
        if (head == null) {
                record[0] = 0;
                return 0;
        }
        int lMax = posOrder(head.left, record);
        int maxfromLeft = record[0];
```

```
        int rMax = posOrder(head.right, record);
        int maxFromRight = record[0];
        int curNodeMax = maxfromLeft + maxFromRight + 1;
        record[0] = Math.max(maxfromLeft, maxFromRight) + 1;
        return Math.max(Math.max(lMax, rMax), curNodeMax);
    }
```

先序、中序和后序数组两两结合重构二叉树

【题目】

已知一棵二叉树的所有节点值都不同，给定这棵二叉树正确的先序、中序和后序数组。请分别用三个函数实现任意两种数组结合重构原来的二叉树，并返回重构二叉树的头节点。

【难度】

先序与中序结合　士　★☆☆☆
中序与后序结合　士　★☆☆☆
先序与后序结合　尉　★★☆☆

【解答】

先序与中序结合重构二叉树的过程如下：

1. 先序数组中最左边的值就是树的头节点值，记为 h，并用 h 生成头节点，记为 head。然后在中序数组中找到 h，假设位置是 i。那么在中序数组中，i 左边的数组就是头节点左子树的中序数组，假设长度为 l，则左子树的先序数组就是先序数组中 h 往右长度也为 l 的数组。

比如：先序数组为[1,2,4,5,8,9,3,6,7]，中序数组为[4,2,8,5,9,1,6,3,7]，二叉树头节点的值是 1，在中序数组中找到 1 的位置，1 左边的数组为[4,2,8,5,9]，是头节点左子树的中序数组，长度为 5；先序数组中 1 的右边长度也为 5 的数组为[2,4,5,8,9]，就是左子树的先序数组。

2. 用左子树的先序和中序数组，递归整个过程建立左子树，返回的头节点记为 left。

3. i 右边的数组就是头节点右子树的中序数组，假设长度为 r。先序数组中右侧等长的部分就是头节点右子树的先序数组。

比如步骤 1 的例子，中序数组中 1 右边的数组为[6,3,7]，长度为 3；先序数组右侧等长的部分为[3,6,7]，它们分别为头节点右子树的中序和先序数组。