```
public Node getNextNode(Node node) {
        if (node == null) {
                return node;
        }
        if (node.right != null) {
                return getLeftMost(node.right);
        } else {
                Node parent = node.parent;
                while (parent != null && parent.left != node) {
                        node = parent;
                        parent = node.parent;
                }
                return parent;
        }
}

public Node getLeftMost(Node node) {
        if (node == null) {
                return node;
        }
        while (node.left != null) {
                node = node.left;
        }
        return node;
}
```

在二叉树中找到两个节点的最近公共祖先

【题目】

给定一棵二叉树的头节点 head，以及这棵树中的两个节点 o1 和 o2，请返回 o1 和 o2 的最近公共祖先节点。

例如，图 3-41 所示的二叉树。

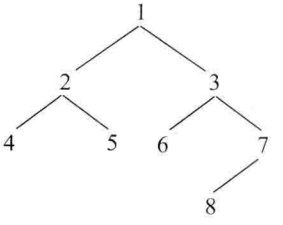

图 3-41

节点 4 和节点 5 的最近公共祖先节点为节点 2，节点 5 和节点 2 的最近公共祖先节点

为节点 2，节点 6 和节点 8 的最近公共祖先节点为节点 3，节点 5 和节点 8 的最近公共祖先节点为节点 1。

进阶：如果查询两个节点的最近公共祖先的操作十分频繁，想法让单条查询的查询时间减少。

再进阶：给定二叉树的头节点 head，同时给定所有想要进行的查询。二叉树的节点数量为 N，查询条数为 M，请在时间复杂度为 $O(N+M)$内返回所有查询的结果。

【难度】

原问题：士 ★☆☆☆

进阶问题：尉 ★★☆☆

再进阶问题：校 ★★★☆

【解答】

先来解决原问题。后序遍历二叉树，假设遍历到的当前节点为 cur。因为是后序遍历，所以先处理 cur 的两棵子树。假设处理 cur 左子树时返回节点为 left，处理右子树时返回 right。

1．如果发现 cur 等于 null，或者 o1、o2，则返回 cur。

2．如果 left 和 right 都为空，说明 cur 整棵子树上没有发现过 o1 或 o2，返回 null。

3．如果 left 和 right 都不为空，说明左子树上发现过 o1 或 o2，右子树上也发现过 o2 或 o1，说明 o1 向上与 o2 向上的过程中，首次在 cur 相遇，返回 cur。

4．如果 left 和 right 有一个为空，另一个不为空，假设不为空的那个记为 node，此时 node 到底是什么？有两种可能，要么 node 是 o1 或 o2 中的一个，要么 node 已经是 o1 和 o2 的最近公共祖先。不管是哪种情况，直接返回 node 即可。

以题目二叉树的例子来说明一下，假设 o1 为节点 6，o2 为节点 8，过程为后序遍历。

- 依次遍历节点 4、节点 5、节点 2，都没有发现 o1 或 o2，所以节点 1 的左子树返回为 null；
- 遍历节点 6，发现节点 6 等于 o1，返回节点 6，所以节点 3 左子树的返回值为节点 6；
- 遍历节点 8，发现节点 8 等于 o2，返回节点 8，所以节点 7 左子树的返回值为节点 8；
- 节点 7 的右子树为 null，所以节点 7 右子树的返回值为 null；
- 遍历节点 7，左子树返回节点 8，右子树返回 null，根据步骤 4，此时返回节点 8，

所以节点 3 的右子树的返回值为节点 8；

- 遍历节点 3，左子树返回节点 6，右子树返回节点 8，根据步骤 3，此时返回节点 3，所以节点 1 的右子树的返回值为节点 3；
- 遍历节点 1，左子树返回 null，右子树返回节点 3，根据步骤 4，最终返回节点 3。

找到两个节点最近公共祖先的详细过程请参看如下代码中的 lowestAncestor 方法。

```
public Node lowestAncestor(Node head, Node o1, Node o2) {
        if (head == null || head == o1 || head == o2) {
                return head;
        }
        Node left = lowestAncestor(head.left, o1, o2);
        Node right = lowestAncestor(head.right, o1, o2);
        if (left != null && right != null) {
                return head;
        }
        return left != null ? left : right;
}
```

进阶问题其实是先花较大的力气建立一种记录，以后执行每次查询时就可以完全根据记录进行查询。记录的方式可以有很多种，本书提供两种记录结构供读者参考，两种记录各有优缺点。

结构一：建立二叉树中每个节点对应的父节点信息，是一张哈希表。

如果对题目中的二叉树建立这种哈希表，哈希表中的信息如下：

key	value
节点 1	null
节点 2	节点 1
节点 3	节点 1
节点 4	节点 2
节点 5	节点 2
节点 6	节点 3
节点 7	节点 3
节点 8	节点 7

key 代表二叉树中的一个节点，value 代表其对应的父节点。只用遍历一次二叉树，这张表就可以创建好，以后每次查询都可以根据这张哈希表进行。

假设想查节点 4 和节点 8 的最近公共祖先，方法是使用如上的哈希表，把包括节点 4 在内的所有节点 4 的祖先节点放进另一个哈希表 A 中，A 表示节点 4 到头节点这条路径上

所有节点的集合。所以 A={节点 4，节点 2，节点 1}。然后使用如上的哈希表，从节点 8 开始往上逐渐移动到头节点。首先是节点 8，发现不在 A 中，然后是节点 7，发现也不在 A 中，接下来是节点 3，依然不在 A 中，最后是节点 1，发现在 A 中，那么节点 1 就是节点 4 和节点 8 的最近公共祖先。只要在移动过程中发现某个节点在 A 中，这个节点就是要求的公共祖先节点。

结构一的具体实现请参看如下代码中 Record1 类的实现，构造函数是创建记录过程，方法 query 是查询操作。

```
public class Record1 {
        private HashMap<Node, Node> map;

        public Record1(Node head) {
                map = new HashMap<Node, Node>();
                if (head != null) {
                        map.put(head, null);
                }
                setMap(head);
        }

        private void setMap(Node head) {
                if (head == null) {
                        return;
                }
                if (head.left != null) {
                        map.put(head.left, head);
                }
                if (head.right != null) {
                        map.put(head.right, head);
                }
                setMap(head.left);
                setMap(head.right);
        }

        public Node query(Node o1, Node o2) {
                HashSet<Node> path = new HashSet<Node>();
                while (map.containsKey(o1)) {
                        path.add(o1);
                        o1 = map.get(o1);
                }
                while (!path.contains(o2)) {
                        o2 = map.get(o2);
                }
                return o2;
        }

}
```

很明显，结构一建立记录的过程时间复杂度为 $O(N)$、额外空间复杂度为 $O(N)$。查询操作时，时间复杂度为 $O(h)$，其中，h 为二叉树的高度。

结构二：直接建立任意两个节点之间的最近公共祖先记录，便于以后查询时直接查。

建立记录的具体过程如下：

1．对二叉树中的每棵子树（一共 N 棵）都进行步骤 2。

2．假设子树的头节点为 h，h 所有的后代节点和 h 节点的最近公共祖先都是 h，记录下来。h 左子树的每个节点和 h 右子树的每个节点的最近公共祖先都是 h，记录下来。

为了保证记录不重复，设计一种好的实现方式是这种结构实现的重点。

结构二的具体实现请参看如下代码中 Record2 类的实现。

```
public class Record2 {
        private HashMap<Node, HashMap<Node, Node>> map;

        public Record2(Node head) {
                map = new HashMap<Node, HashMap<Node, Node>>();
                initMap(head);
                setMap(head);
        }

        private void initMap(Node head) {
                if (head == null) {
                        return;
                }
                map.put(head, new HashMap<Node, Node>());
                initMap(head.left);
                initMap(head.right);
        }

        private void setMap(Node head) {
                if (head == null) {
                        return;
                }
                headRecord(head.left, head);
                headRecord(head.right, head);
                subRecord(head);
                setMap(head.left);
                setMap(head.right);
        }

        private void headRecord(Node n, Node h) {
                if (n == null) {
                        return;
                }
                map.get(n).put(h, h);
                headRecord(n.left, h);
                headRecord(n.right, h);
```

```
        }

        private void subRecord(Node head) {
                if (head == null) {
                        return;
                }
                preLeft(head.left, head.right, head);
                subRecord(head.left);
                subRecord(head.right);
        }

        private void preLeft(Node l, Node r, Node h) {
                if (l == null) {
                        return;
                }
                preRight(l, r, h);
                preLeft(l.left, r, h);
                preLeft(l.right, r, h);
        }

        private void preRight(Node l, Node r, Node h) {
                if (r == null) {
                        return;
                }
                map.get(l).put(r, h);
                preRight(l, r.left, h);
                preRight(l, r.right, h);
        }

        public Node query(Node o1, Node o2) {
                if (o1 == o2) {
                        return o1;
                }
                if (map.containsKey(o1)) {
                        return map.get(o1).get(o2);
                }
                if (map.containsKey(o2)) {
                        return map.get(o2).get(o1);
                }
                return null;
        }

}
```

如果二叉树的节点数为N，想要记录每两个节点之间的信息，信息的条数为$((N-1)\times N)/2$。所以建立结构二的过程的额外空间复杂度为$O(N^2)$，时间复杂度为$O(N^2)$，单次查询的时间复杂度为$O(1)$。

再进阶的问题：请参看下一题“Tarjan 算法与并查集解决二叉树节点间最近公共祖先的批量查询问题”。