令 c 等于 stack 的栈顶节点——节点 3，此时步骤 2 的条件②命中，将节点 7 压入 stack，h 为节点 6，stack 从栈顶到栈底为 7，3，1。

令 c 等于 stack 的栈顶节点——节点 7，此时步骤 2 的条件③命中，将节点 7 从 stack 中弹出并打印，h 变为节点 7，stack 从栈顶到栈底为 3，1。

令 c 等于 stack 的栈顶节点——节点 3，此时步骤 2 的条件③命中，将节点 3 从 stack 中弹出并打印，h 变为节点 3，stack 从栈顶到栈底为 1。

令 c 等于 stack 的栈顶节点——节点 1，此时步骤 2 的条件③命中，将节点 1 从 stack 中弹出并打印，h 变为节点 1，stack 为空。

过程结束。

只用一个栈实现后序遍历的全部过程请参看如下代码中的 posOrderUnRecur2 方法。

```
public void posOrderUnRecur2(Node h) {
        System.out.print("pos-order: ");
        if (h != null) {
                Stack<Node> stack = new Stack<Node>();
                stack.push(h);
                Node c = null;
                while (!stack.isEmpty()) {
                        c = stack.peek();
                        if (c.left != null && h != c.left && h != c.right) {
                                stack.push(c.left);
                        } else if (c.right != null && h != c.right) {
                                stack.push(c.right);
                        } else {
                                System.out.print(stack.pop().value + " ");
                                h = c;
                        }
                }
        }
        System.out.println();
}
```

打印二叉树的边界节点

【题目】

给定一棵二叉树的头节点 head，按照如下两种标准分别实现二叉树边界节点的逆时针打印。

标准一：

1．头节点为边界节点。

2．叶节点为边界节点。

3．如果节点在其所在的层中是最左或最右的，那么也是边界节点。

标准二：

1．头节点为边界节点。

2．叶节点为边界节点。

3．树左边界延伸下去的路径为边界节点。

4．树右边界延伸下去的路径为边界节点。

例如，如图 3-2 所示的树。

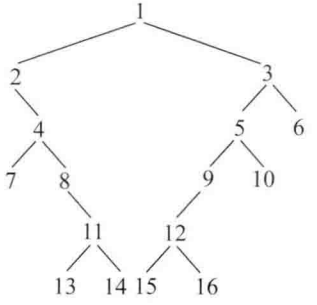

图 3-2

按标准一的打印结果为：1，2，4，7，11，13，14，15，16，12，10，6，3

按标准二的打印结果为：1，2，4，7，13，14，15，16，10，6，3

【要求】

1．如果节点数为 N，两种标准实现的时间复杂度要求都为 $O(N)$，额外空间复杂度要求都为 $O(h)$，h 为二叉树的高度。

2．两种标准都要求逆时针顺序且不重复打印所有的边界节点。

【难度】

尉 ★★☆☆

【解答】

按照标准一的要求实现打印的具体过程如下：

1．得到二叉树每一层上最左和最右的节点。以题目的例子来说，这个记录如下：

	最左节点	最右节点
第一层	1	1
第二层	2	3
第三层	4	6
第四层	7	10
第五层	11	12
第六层	13	16

2．从上到下打印所有层中的最左节点。对题目的例子来说，即打印：1，2，4，7，11，13。

3．先序遍历二叉树，打印那些不属于某一层最左或最右的节点，但同时又是叶节点的节点。对题目的例子来说，即打印：14，15。

4．从下到上打印所有层中的最右节点，但节点不能既是最左节点，又是最右节点。对题目的例子来说，即打印：16，12，10，6，3。

按标准一打印的全部过程请参看如下代码中的 printEdge1 方法。

```
public class Node {
        public int value;
        public Node left;
        public Node right;

        public Node(int data) {
                this.value = data;
        }
}

public void printEdge1(Node head) {
        if (head == null) {
                return;
        }
        int height = getHeight(head, 0);
        Node[][] edgeMap = new Node[height][2];
        setEdgeMap(head, 0, edgeMap);
        // 打印左边界
        for (int i = 0; i != edgeMap.length; i++) {
                System.out.print(edgeMap[i][0].value + " ");
        }
        // 打印既不是左边界，也不是右边界的叶子节点
        printLeafNotInMap(head, 0, edgeMap);
        // 打印右边界，但不是左边界的节点
        for (int i = edgeMap.length - 1; i != -1; i--) {
                if (edgeMap[i][0] != edgeMap[i][1]) {
                        System.out.print(edgeMap[i][1].value + " ");
                }
        }
```

```
        System.out.println();
    }

    public int getHeight(Node h, int l) {
        if (h == null) {
            return l;
        }
        return Math.max(getHeight(h.left, l + 1), getHeight(h.right, l + 1));
    }

    public void setEdgeMap(Node h, int l, Node[][] edgeMap) {
        if (h == null) {
            return;
        }
        edgeMap[l][0] = edgeMap[l][0] == null ? h : edgeMap[l][0];
        edgeMap[l][1] = h;
        setEdgeMap(h.left, l + 1, edgeMap);
        setEdgeMap(h.right, l + 1, edgeMap);
    }

    public void printLeafNotInMap(Node h, int l, Node[][] m) {
        if (h == null) {
            return;
        }
        if (h.left == null && h.right == null && h != m[l][0] && h != m[l][1]) {
            System.out.print(h.value + " ");
        }
        printLeafNotInMap(h.left, l + 1, m);
        printLeafNotInMap(h.right, l + 1, m);
    }
```

获取高度

获取边界

打印不在map中的叶子节点

按照标准二的要求实现打印的具体过程如下：

1．从头节点开始往下寻找，只要找到第一个既有左孩子，又有右孩子的节点，记为 h，则进入步骤 2。在这个过程中，找过的节点都打印。对题目的例子来说，即打印：1，因为头节点直接符合要求，所以打印后没有后续的寻找过程，直接进入步骤 2。但如果二叉树如图 3-3 所示，此时则打印：1，2，3。节点 3 是从头节点开始往下第一个符合要求的。如果二叉树从上到下一直找到叶节点也不存在符合要求的节点，说明二叉树是棒状结构，那么打印找过的节点后直接返回即可。

2．h 的左子树先进入步骤 3 的打印过程；h 的右子树再进入步骤 4 的打印过程；最后返回。

3．打印左边界的延伸路径以及 h 左子树上所有的叶节点，具体请参看 printLeftEdge 方法。

4．打印右边界的延伸路径以及 h 右子树上所有的叶节点，具体请参看 printRightEdge 方法。

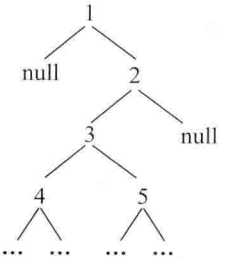

图 3-3

按标准二打印的全部过程请参看如下代码中的 printEdge2 方法。

```
public void printEdge2(Node head) {
        if (head == null) {
                return;
        }
        System.out.print(head.value + " ");
        if (head.left != null && head.right != null) {
                printLeftEdge(head.left, true);
                printRightEdge(head.right, true);
        } else {
                printEdge2(head.left != null ? head.left : head.right);
        }
        System.out.println();
}

public void printLeftEdge(Node h, boolean print) {
        if (h == null) {
                return;
        }
        if (print || (h.left == null && h.right == null)) {
                System.out.print(h.value + " ");
        }
        printLeftEdge(h.left, print);
        printLeftEdge(h.right, print && h.left == null ? true : false);
}

public void printRightEdge(Node h, boolean print) {
        if (h == null) {
                return;
        }
        printRightEdge(h.left, print && h.right == null ? true : false);
        printRightEdge(h.right, print);
        if (print || (h.left == null && h.right == null)) {
                System.out.print(h.value + " ");
        }
}
```