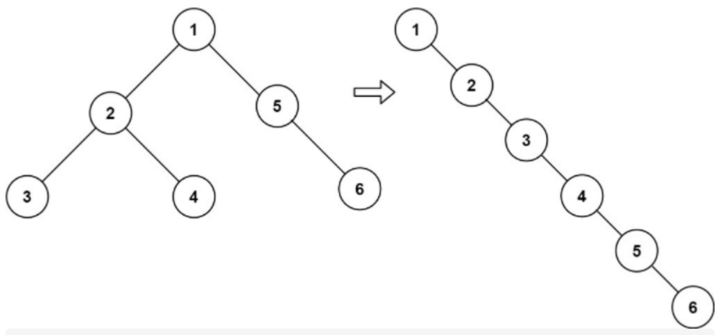
114 二叉树展开为链表

```
Label：二叉树
给你二叉树的根结点 root ，请你将它展开为一个单链表：
展开后的单链表应该同样使用 TreeNode ，其中 right 子指针指向链表中下一个结点，而左子指针始终为
null 。
展开后的单链表应该与二叉树 先序遍历 顺序相同。
```



```
输入：root = [1,2,5,3,4,null,6]
输出：[1,null,2,null,3,null,4,null,5,null,6]
```

- 先遍历 再构造 递归

```
class Solution {
    public void flatten(TreeNode root) {
        if (root == null || (root.left==null && root.right == null))
            return;
        List<Integer> list = new ArrayList<>();
        flat(root, list);
        root.left = null;
        root.right = new TreeNode(0,null,null);
        TreeNode curr = root.right;
        // 构造新 Tree
        for (int i = 1; i < list.size(); i++) {
            curr.val = list.get(i);
            curr.left = null;

            if (i == list.size() -1 ){
                curr.right = null;
            } else {
                curr.right = new TreeNode(0,null,null);
            }
            curr = curr.right;
        }
    }

    private void flat(TreeNode root, List<Integer> list) {
        if (root == null) return;
        list.add(root.val);
        flat(root.left, list);
        flat(root.right, list);
    }
}
```

- 先遍历 后构造 迭代

```
class Solution {
    public void flatten(TreeNode root) {
        List<TreeNode> list = new ArrayList<TreeNode>();
        Deque<TreeNode> stack = new LinkedList<TreeNode>();
        TreeNode node = root;
        while (node != null || !stack.isEmpty()) {
            while (node != null) {
                list.add(node);
                stack.push(node);
                node = node.left;
            }
            node = stack.pop();
            node = node.right;
        }
        int size = list.size();
        for (int i = 1; i < size; i++) {
            TreeNode prev = list.get(i - 1), curr = list.get(i);
            prev.left = null;
            prev.right = curr;
        }
    }
}
```

- 前序与遍历同时展开

```
class Solution {
    public void flatten(TreeNode root) {
        if (root == null) return;

        Stack<TreeNode> stack = new Stack<TreeNode>();
        stack.push(root);
        TreeNode prev = null;
        while (!stack.isEmpty()) {
            TreeNode curr = stack.pop();
            if (prev != null) {
                prev.left = null;
                prev.right = curr;  // 因为已经保存了，所以这里直接可以对树进行修改
            }

            TreeNode left = curr.left, right = curr.right;
            if (right != null) {
                stack.push(right);
            }
            if (left != null) {
                stack.push(left);
            }
            prev = curr;
        }
    }
}
```

- 寻找前驱节点 (结合图才能理解)

```java
class Solution {
    public void flatten(TreeNode root) {
        TreeNode curr = root;
        while (curr != null) {
            if (curr.left != null) {
                TreeNode next = curr.left;
                TreeNode predecessor = next;

                while (predecessor.right != null) {
                    predecessor = predecessor.right;
                }
                predecessor.right = curr.right;
                curr.left = null;
                curr.right = next;
            }
            curr = curr.right;
        }
    }
}
```