82 删除排序链表中的重复元素II

```
Label：链表 双指针 递归

给定一个排序链表，删除所有含有重复数字的节点，只保留原始链表中 没有重复出现 的数字。
    输入: 1->2->3->3->4->4->5
    输出: 1->2->5
    输入: 1->1->1->2->3
    输出: 2->3
```

- 迭代（速度最快、还没看）

```
class Solution {
    public ListNode deleteDuplicates(ListNode head) {
        ListNode newHead = new ListNode();
        ListNode cuNewHead = newHead;
        while (head != null) {
            if (head.next != null && head.next.val == head.val) {
                while(head.next != null && head.next.val == head.val) {
                    head.next = head.next.next;
                }
                head = head.next;
                continue;
            }
            cuNewHead.next = head;
            cuNewHead = cuNewHead.next;
            head = head.next;
        }
        cuNewHead.next = null;
        return newHead.next;
    }
}
```

- 递归

```
class Solution {
    public ListNode deleteDuplicates(ListNode head) {
        if (head == null || head.next == null)
            return head;
        // 当前节点值和下一个节点值相等，直接跳过相等的节点即可
        if (head.val == head.next.val) {
            while (head.next != null && head.val == head.next.val)
                head = head.next;
            return deleteDuplicates(head.next);
        } else {
            head.next = deleteDuplicates(head.next);
            return head;
        }
    }
}
```

- 遍历（双向链表）

```
class Solution {
    public ListNode deleteDuplicates(ListNode head) {
        if (head == null || head.next == null) {
            return head;
        }

        Deque<Integer> queue = new LinkedList<Integer>();
        ListNode curr = head.next;
        queue.addFirst(head.val);

        while (curr != null) {
            if (!queue.isEmpty() && curr.val == queue.peekLast()) {
                int pre = queue.removeLast();
                while (curr != null && curr.next != null && curr.next.val ==
pre) {
                    curr = curr.next;  // 防止连续出现多个
                }
            }else {
                queue.addLast(curr.val);
            }
            curr = curr.next;
        }

        // 构建新的链表
        if (queue.isEmpty()) {
            return null;
        }
        ListNode newHead = new ListNode(queue.removeFirst());
        curr = newHead;
        while (!queue.isEmpty()) {
            curr.next = new ListNode(queue.removeFirst());
            curr = curr.next;
        }
        return newHead;
    }
}
```

- 双指针

```
class Solution {
    public ListNode deleteDuplicates(ListNode head) {
        if (head == null || head.next == null)
            return head;

        ListNode dummy = new ListNode(-1);  // 虚拟头结点
        ListNode tail = dummy;  // 定义一个尾巴，用于尾插法。

        dummy.next = head;
        ListNode pre = dummy;
        ListNode curr = pre.next;
        while (curr != null && curr.next != null) {
            boolean is = false;
            // 去除所有重复点 剩1个
            while (curr.next != null && curr.val == curr.next.val) {
                curr.next = curr.next.next;
                is = true;
            }

            if (is) {
                // 跨过那一个重复点
                pre.next = curr.next;
            } else {
                // 无重复点
                pre = pre.next;

            }
              curr = curr.next;
        }
        return dummy.next;
    }
}
```