将单链表的每 K 个节点之间逆序

【题目】

给定一个单链表的头节点 head，实现一个调整单链表的函数，使得每 K 个节点之间逆序，如果最后不够 K 个节点一组，则不调整最后几个节点。

例如：

链表：1->2->3->4->5->6->7->8->null，K=3。

调整后为：3->2->1->6->5->4->7->8->null。其中 7、8 不调整，因为不够一组。

【难度】

尉 ★★☆☆

【解答】

首先，如果 K 的值小于 2，不用进行任何调整。因为 K<1 没有意义，K==1 时，代表每 1 个节点为 1 组进行逆序，原链表也没有任何变化。接下来介绍两种方法，如果链表长度为 N，方法一的时间复杂度为 O(N)，额外空间复杂度为 O(K)。方法二的时间复杂度为 O(N)，额外空间复杂度为 O(1)。本题考查面试者代码实现不出错的能力。

方法一：利用栈结构的解法。

1．从左到右遍历链表，如果栈的大小不等于 K，就将节点不断压入栈中。

2．当栈的大小第一次到达 K 时，说明第一次凑齐了 K 个节点进行逆序，从栈中依次弹出这些节点，并根据弹出的顺序重新连接，这一组逆序完成后，需要记录一下新的头部，同时第一组的最后一个节点（原来是头节点）应该连接下一个节点。

例如：链表 1->2->3->4->5->6->7->8->null，K = 3。第一组节点进入栈，从栈顶到栈底依次为 3，2，1。逆序重连之后为 3->2->1->...，然后节点 1 去连接节点 4，链表变为 3->2->1->4->5->6->7->8->null，之后从节点 4 开始不断处理 K 个节点为一组的后续情况，也就是步骤 3，并且需要记录节点 3，因为链表的头部已经改变，整个过程结束后需要返回这个新的头节点，记为 newHead。

3．步骤 2 之后，当栈的大小每次到达 K 时，说明又凑齐了一组应该进行逆序的节点，从栈中依次弹出这些节点，并根据弹出的顺序重新连接。这一组逆序完成后，该组的第一

个节点（原来是该组最后一个节点）应该被上一组的最后一个节点连接上，这一组的最后一个节点（原来是该组第一个节点）应该连接下一个节点。然后继续去凑下一组，直到链表都被遍历完。

例如：链表 3->2->1->4->5->6->7->8->null，$K=3$，第一组已经处理完。第二组从栈顶到栈底依次为 6，5，4。逆序重连之后为 6->5->4，然后节点 6 应该被节点 1 连接，节点 4 应该连接节点 7，链表变为 3->2->1->6->5->4->7->8->null。然后继续从节点 7 往下遍历。

4．最后应该返回 newHead，作为链表新的头节点。

方法一的具体实现请参看如下代码中的 reverseKNodes1 方法。

```
public class Node {
        public int value;
        public Node next;

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

public Node reverseKNodes1(Node head, int K) {
        if (K < 2) {
                return head;
        }
        Stack<Node> stack = new Stack<Node>();
        Node newHead = head;
        Node cur = head;
        Node pre = null;
        Node next = null;
        while (cur != null) {
                next = cur.next;
                stack.push(cur);
                if (stack.size() == K) {
                        pre = resign1(stack, pre, next);
                        newHead = newHead == head ? cur : newHead;
                }
                cur = next;
        }
        return newHead;
}

public Node resign1(Stack<Node> stack, Node left, Node right) {
        Node cur = stack.pop();
        if (left != null) {
                left.next = cur;
        }
        Node next = null;
        while (!stack.isEmpty()) {
                next = stack.pop();
```

```
                cur.next = next;
                cur = next;
        }
        cur.next = right;
        return cur;
}
```

方法二：不需要栈结构，在原链表中直接调整。

用变量记录每一组开始的第一个节点和最后一个节点，然后直接逆序调整，把这一组的节点都逆序。和方法一一样，同样需要注意第一组节点的特殊处理，以及之后的每个组在逆序重连之后，需要让该组的第一个节点（原来是最后一个节点）被之前组的最后一个节点连接上，将该组的最后一个节点（原来是第一个节点）连接下一个节点。

方法二的具体实现请参看如下代码中的 reverseKNodes2 方法。

```
public Node reverseKNodes2(Node head, int K) {
        if (K < 2) {
                return head;
        }
        Node cur = head;
        Node start = null;
        Node pre = null;
        Node next = null;
        int count = 1;
        while (cur != null) {
                next = cur.next;
                if (count == K) {
                        start = pre == null ? head : pre.next;
                        head = pre == null ? cur : head;
                        resign2(pre, start, cur, next);
                        pre = start;
                        count = 0;
                }
                count++;
                cur = next;
        }
        return head;
}

public void resign2(Node left, Node start, Node end, Node right) {
        Node pre = start;
        Node cur = start.next;
        Node next = null;
        while (cur != right) {
                next = cur.next;
                cur.next = pre;
                pre = cur;
                cur = next;
        }
```

```
        if (left != null) {
                left.next = end;
        }
        start.next = right;
}
```

删除无序单链表中值重复出现的节点

【题目】

给定一个无序单链表的头节点 head，删除其中值重复出现的节点。

例如：1->2->3->3->4->4->2->1->1->null，删除值重复的节点之后为 1->2->3->4->null。

请按以下要求实现两种方法。

方法 1：如果链表长度为 N，时间复杂度达到 $O(N)$。

方法 2：额外空间复杂度为 $O(1)$。

【难度】

士 ★☆☆☆

【解答】

方法一：利用哈希表。时间复杂度为 $O(N)$，额外空间复杂度为 $O(N)$。

具体过程如下：

1．生成一个哈希表，因为头节点是不用删除的节点，所以首先将头节点的值放入哈希表。

2．从头节点的下一个节点开始往后遍历节点，假设当前遍历到 cur 节点，先检查 cur 的值是否在哈希表中，如果在，则说明 cur 节点的值是之前出现过的，就将 cur 节点删除，删除的方式是将最近一个没有被删除的节点 pre 连接到 cur 的下一个节点，即 pre.next=cur.next。如果不在，将 cur 节点的值加入哈希表，同时令 pre=cur，即更新最近一个没有被删除的节点。

方法一的具体实现请参看如下代码中的 removeRep1 方法。

```
public Node {
        public int value;
        public Node next;

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