王争的算法训练营

习题课：链表

习题课：链表

配套习题（13）：

203. 移除链表元素 （简单）**(已讲)**
876. 链表的中间结点（简单）**(已讲)**

83. 删除排序链表中的重复元素（简单）
剑指 Offer 25. 合并两个排序的链表 （中等）**(已讲)**
2. 两数相加 （中等）
206. 反转链表 （中等）
234. 回文链表 （中等，蚂蚁金服社招，我当年被Google同事面到的）
328. 奇偶链表（中等）
25. K 个一组翻转链表（困难）

剑指 Offer 22. 链表中倒数第k个节点 （简单）
19. 删除链表的倒数第 N 个结点 （中等）
160. 相交链表（简单）
141. 环形链表（简单） 判断链表中是否存在环

题型说明

很常考，是重中之重

题型有限，代码实现难

这类题目考察的不是算法，不像动态规划，题型很多，他纯粹考察的是候选人的编程能力。所以，面试中基本上都是原题或者在原题上稍加改造。

如何准备这类题型?

虽然代码实现难，很容易写出bug，但这类题目不难准备面试。

只要把这节课布置的13道题都写熟练，基本上链表的题目就没有问题了。

解题技巧

链表相关的问题都会涉及“遍历”，核心是通过“画图举例”确定遍历的“三要素”：

1）遍历的结束条件：p==null or p.next==null …

2）指针的初始值：p=head or ….

3）遍历的核心逻辑：…（视题目要求而定）

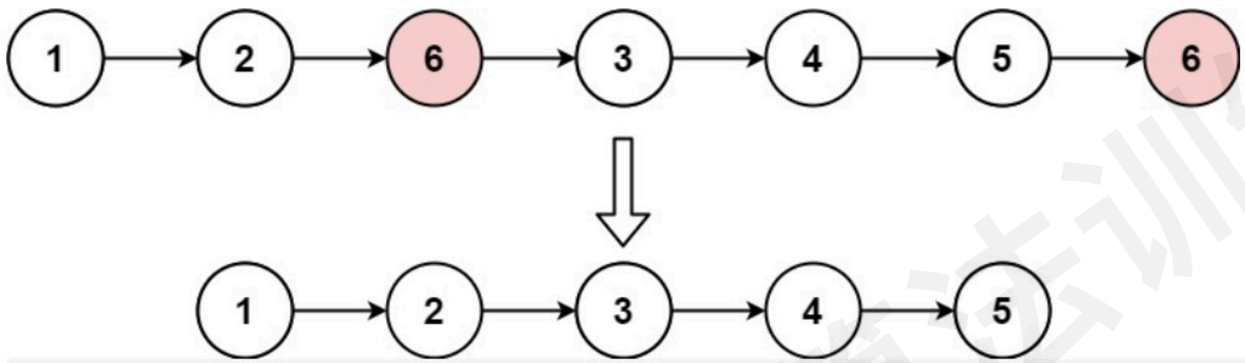
特殊情况处理：是否需要对头节点、尾节点、空链表等做特殊处理?

引入虚拟节点：是否可以通过添加虚拟节点简化编程?

203. 移除链表元素 （简单）(已讲)

给你一个链表的头节点 `head` 和一个整数 `val` ，请你删除链表中所有满足 `Node.val == val` 的节点，并返回 **新的头节点** 。

示例 1：




```
输入：head = [1,2,6,3,4,5,6], val = 6
输出：[1,2,3,4,5]
```

示例 2：

```
输入：head = [], val = 1
输出：[]
```

示例 3：

```
输入：head = [7,7,7,7], val = 7
输出：[]
```

- 改变链表的万能写法

```
class Solution {
    public ListNode removeElements(ListNode head, int val) {
        if (head == null) return null;
        ListNode newHead = new ListNode();
        ListNode tail = newHead;
        ListNode p = head;
        while (p != null) {
            ListNode tmp = p.next;
            if (p.val != val) {
                p.next = null;
                tail.next = p;
                tail = p;
            }
            p = tmp;
        }
        return newHead.next;
    }
}
```

时间复杂度O(n)，空间复杂度O(1)

习题课：链表配套习题讲解

876. 链表的中间结点（简单）(已讲)

给定一个头结点为 `head` 的非空单链表，返回链表的中间结点。

如果有两个中间结点，则返回第二个中间结点。

示例 1：

```
输入：[1,2,3,4,5]
输出：此列表中的结点 3 (序列化形式：[3,4,5])
返回的结点值为 3 。 (测评系统对该结点序列化表述是 [3,4,5])。
注意，我们返回了一个 ListNode 类型的对象 ans，这样：
ans.val = 3, ans.next.val = 4, ans.next.next.val = 5, 以及 ans.next.next.next
= NULL.
```

习题课：链表配套习题讲解

876. 链表的中间结点（简单）**(已讲)**

画图举例确定遍历三要素：

- 指针初始值：fast=head, slow=head
- 遍历的结束条件：fast==null || fast.next==null
- 遍历的核心逻辑：slow =slow.next;fast=fast.next.next;

特殊情况处理：不需要特殊处理头节点、尾节点、空链表

是否需要虚拟头节点：不需要

习题课：链表配套习题讲解

876. 链表的中间结点（简单）**(已讲)**

```
class Solution {
    public ListNode middleNode(ListNode head) {
        ListNode slow = head;
        ListNode fast = head;
        while (fast != null && fast.next != null) {
            slow = slow.next;
            fast = fast.next.next;
        }
        return slow;
    }
}
```

- 指针初始值：fast=head, slow=head
- 遍历的结束条件：fast==null ll fast.next==null
- 遍历的核心逻辑：slow =slow.next;fast=fast.next.next;

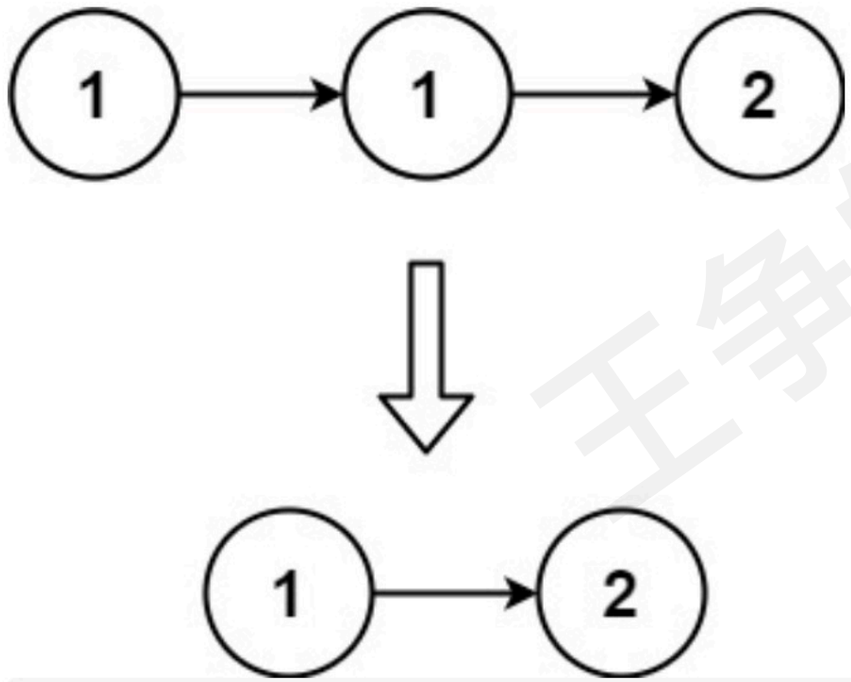
时间复杂度O(n)，空间复杂度O(1)

83. 删除排序链表中的重复元素（简单）

存在一个按升序排列的链表，给你这个链表的头节点 `head` ，请你删除所有重复的元素，使每个元素 **只出现一次** 。

返回同样按升序排列的结果链表。

示例 1：




```
输入：head = [1,1,2]
输出：[1,2]
```

数组中删除重复元素：

1. 利用栈
2. 利用数组
3. 在原数组上处理

1->1->1->2->2->3

习题课：链表

```
class Solution {
    public ListNode deleteDuplicates(ListNode head) {
        if (head == null) return head;
        ListNode newHead = new ListNode(-111, null);//虚拟头节点
        ListNode tail = newHead;
        ListNode p = head;
        while (p != null) {
            ListNode tmp = p.next;
            if (p.val != tail.val) {
                tail.next = p;
                tail = p;
                p.next = null;
            }
            p = tmp;
        }
        return newHead.next;
    }
}
```

时间复杂度O(n)，空间复杂度O(1)

1/第一行?

2/-111?

3/空间复杂度?

习题课：链表

剑指 Offer 25. 合并两个排序的链表 （中等）已讲

剑指 Offer 25. 合并两个排序的链表

难度 简单　113　收藏　分享　切换为英文　接收动态　反馈

输入两个递增排序的链表，合并这两个链表并使新链表中的节点仍然是递增排序的。

合并两个有序数组

示例1：

```
输入：1->2->4, 1->3->4
输出：1->1->2->3->4->4
```

限制：

```
0 <= 链表长度 <= 1000
```

```java
class Solution {
    public ListNode mergeTwoLists(ListNode l1, ListNode l2) {
        if (l1 == null) return l2;
        if (l2 == null) return l1;
        ListNode p1 = l1;
        ListNode p2 = l2;
        ListNode head = new ListNode();//虚拟头节点
        ListNode tail = head;
        while (p1 != null && p2 != null) {
            if (p1.val <= p2.val) {
                tail.next = p1;
                tail = p1;
                p1 = p1.next;
            } else {
                tail.next = p2;
                tail = p2;
                p2 = p2.next;
            }
        }
        // 如果p1还没处理完，就把剩下的直接接到tail后面
        if (p1 != null) tail.next = p1;
        // 同理
        if (p2 != null) tail.next = p2;
        return head.next;
    }
}
```

时间复杂度O(m+n)，空间复杂度O(1)

2. 两数相加 （中等）

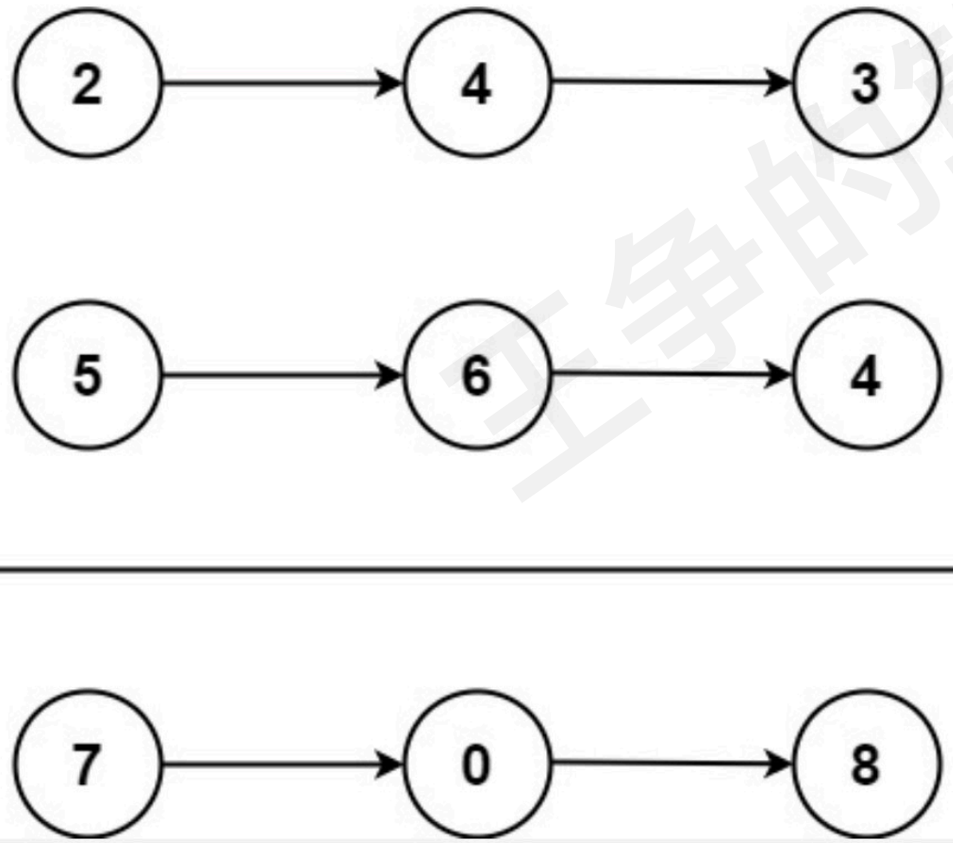
给你两个 **非空** 的链表，表示两个非负的整数。它们每位数字都是按照 **逆序** 的方式存储的，并且每个节点只能存储 **一位** 数字。

请你将两个数相加，并以相同形式返回一个表示和的链表。

你可以假设除了数字 0 之外，这两个数都不会以 0 开头。

大数加法：数存储在数组中

示例 1：



```java
class Solution {
    public ListNode addTwoNumbers(ListNode l1, ListNode l2) {
        ListNode p1 = l1;
        ListNode p2 = l2;
        ListNode dummyHead = new ListNode();//虚拟节点
        ListNode tail = dummyHead;
        int carry = 0;
        while (p1 != null || p2 != null) {
            int sum = 0;
            if (p1 != null) {
                sum += p1.val;
                p1 = p1.next;
            }
            if (p2 != null) {
                sum += p2.val;
                p2 = p2.next;
            }
            if (carry != 0) {
                sum += carry;
            }
            tail.next = new ListNode(sum%10);
            carry = sum/10;
            tail = tail.next;
        }
        if (carry != 0) {
             tail.next = new ListNode(carry);
        }
        return dummyHead.next;
    }
}
```

时间复杂度O(n)，空间复杂度O(n)，n表示两个链表的最大长度

习题课：链表

206. 反转链表 （中等）

非递归解法

难度 简单 1667 收藏 分享 切换为英文 接收动态 反馈

反转一个单链表。

示例:

```
输入: 1->2->3->4->5->NULL
输出: 5->4->3->2->1->NULL
```

进阶:
你可以迭代或递归地反转链表。你能否用两种方法解决这道题?

206. 反转链表 （中等）

```
class Solution {
    public ListNode reverseList(ListNode head) {
        ListNode newHead = null;
        ListNode p = head;
        while (p != null) {
            ListNode tmp = p.next;
            p.next = newHead;
            newHead = p;
            p = tmp;
        }
        return newHead;
    }
}
```

时间复杂度O(n)，空间复杂度O(1)

234. 回文链表 （中等，蚂蚁金服社招，我当年被Google同事面到的）

请判断一个链表是否为回文链表。

示例 1:

```
输入: 1->2
输出: false
```

示例 2:

```
输入: 1->2->2->1
输出: true
```

```java
public boolean isPalindrome(ListNode head) {
    if (head == null || head.next == null) return true;
    ListNode midNode = findMidNode(head);
    ListNode rightHalfHead = reverseList(midNode.next);
    ListNode p = head;
    ListNode q = rightHalfHead;
    while (q != null) {
        if (p.val != q.val) return false;
        p = p.next;
        q = q.next;
    }
    return true;
}
```

```java
private ListNode findMidNode(ListNode head) {
    ListNode slow = head;
    ListNode fast = head;
    while (fast.next != null && fast.next.next != null) {
        slow = slow.next;
        fast = fast.next.next;
    }
    return slow;
}

private ListNode reverseList(ListNode head) {
    if (head == null) return null;
    ListNode newHead = null;
    ListNode p = head;
    while (p != null) {
        ListNode tmp = p.next;
        p.next = newHead;
        newHead = p;
        p = tmp;
    }
    return newHead;
}
```

时间复杂度O(n)，空间复杂度O(1)

328. 奇偶链表（中等）

给定一个单链表，把所有的奇数节点和偶数节点分别排在一起。请注意，这里的奇数节点和偶数节点指的是节点编号的奇偶性，而不是节点的值的奇偶性。

请尝试使用原地算法完成。你的算法的空间复杂度应为 O(1)，时间复杂度应为 O(nodes)，nodes 为节点总数。

示例 1:

```
输入: 1->2->3->4->5->NULL
输出: 1->3->5->2->4->NULL
```

示例 2:

```
输入: 2->1->3->5->6->4->7->NULL
输出: 2->3->6->7->1->5->4->NULL
```

```java
class Solution {
    public ListNode oddEvenList(ListNode head) {
        if (head == null) return null;
        ListNode oddHead = new ListNode();
        ListNode oddTail = oddHead;
        ListNode evenHead = new ListNode();
        ListNode evenTail = evenHead;
        ListNode p = head;
        int count = 1;
        while (p != null) {
            ListNode tmp = p.next;
            if (count % 2 == 1) { // 奇数
                p.next = null;
                oddTail.next = p;
                oddTail = p;
            } else { // 偶数
                p.next = null;
                evenTail.next = p;
                evenTail = p;
            }
            count++;
            p = tmp;
        }
        oddTail.next = evenHead.next;
        return oddHead.next;
    }
}
```

时间复杂度O(n)，空间复杂度O(1)

25. K 个一组翻转链表（困难）

给你一个链表，每 *k* 个节点一组进行翻转，请你返回翻转后的链表。

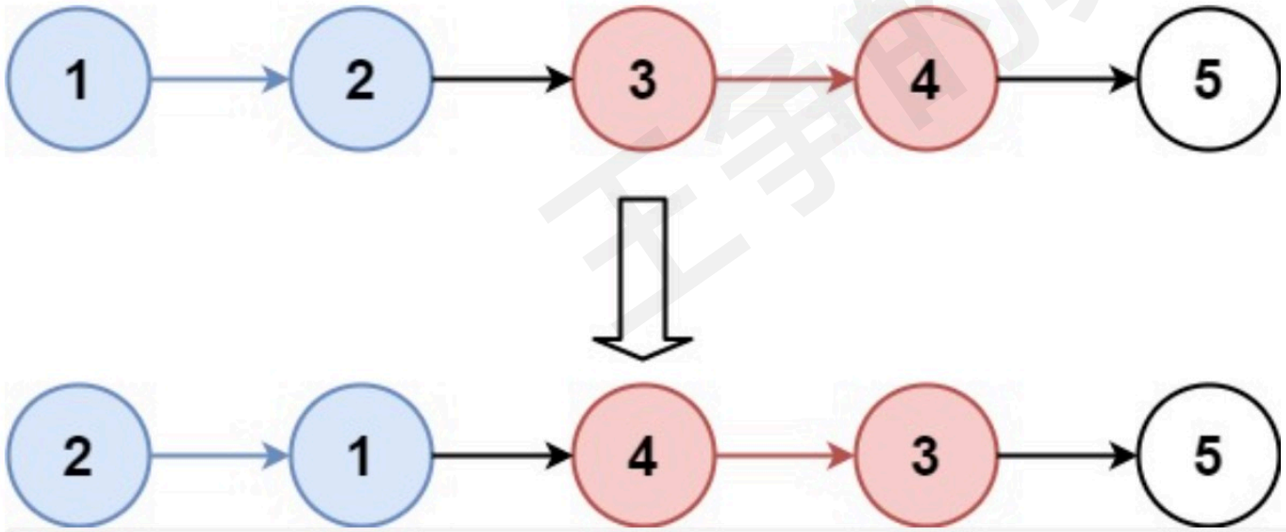
k 是一个正整数，它的值小于或等于链表的长度。

如果节点总数不是 *k* 的整数倍，那么请将最后剩余的节点保持原有顺序。

进阶：

- 你可以设计一个只使用常数额外空间的算法来解决此问题吗？
- **你不能只是单纯的改变节点内部的值**，而是需要实际进行节点交换。

示例 1：




```
输入：head = [1,2,3,4,5], k = 2
输出：[2,1,4,3,5]
```

```java
class Solution {
    public ListNode reverseKGroup(ListNode head, int k) {
        ListNode dummyHead = new ListNode();
        ListNode tail = dummyHead;

        ListNode p = head;
        while (p != null) {
            int count = 0;
            ListNode q = p;
            while (q != null) {
                count++;
                if (count == k) {
                    break;
                }
                q = q.next;
            }
            if (q == null) {
                tail.next = p;
                return dummyHead.next;
            } else {
                ListNode tmp = q.next;
                ListNode[] nodes = reverse(p, q);
                tail.next = nodes[0];
                tail = nodes[1];
                p = tmp;
            }
        }
        return dummyHead.next;
    }
}
```

```java
private ListNode[] reverse(ListNode head, ListNode tail) {
    ListNode newHead = null;
    ListNode p = head;
    while (p != tail) {
        ListNode tmp = p.next;
        p.next = newHead;
        newHead = p;
        p = tmp;
    }
    tail.next = newHead;
    newHead = tail;

    return new ListNode[]{tail, head};
}
```

时间复杂度O(n)，空间复杂度O(1)

剑指 Offer 22. 链表中倒数第k个节点 （简单）

输入一个链表，输出该链表中倒数第k个节点。为了符合大多数人的习惯，本题从1开始计数，即链表的尾节点是倒数第1个节点。

例如，一个链表有 `6` 个节点，从头节点开始，它们的值依次是 `1、2、3、4、5、6`。这个链表的倒数第 `3` 个节点是值为 `4` 的节点。

示例：

给定一个链表: **1->2->3->4->5**, 和 ***k* = 2.**

返回链表 **4->5.**

习题课：链表

```
class Solution {
    public ListNode getKthFromEnd(ListNode head, int k) {
        // 遍历1
        ListNode fast = head;
        int count = 0;
        while (fast != null) {
            count++;
            if (count == k) break;
            fast = fast.next;
        }
        if (fast == null) { // 链表节点不够k
            return null;
        }
        // 遍历2
        ListNode slow = head;
        while (fast.next != null) {
            slow  = slow.next;
            fast = fast.next;
        }
        return slow;
    }
}
```

时间复杂度O(n)，空间复杂度O(1)

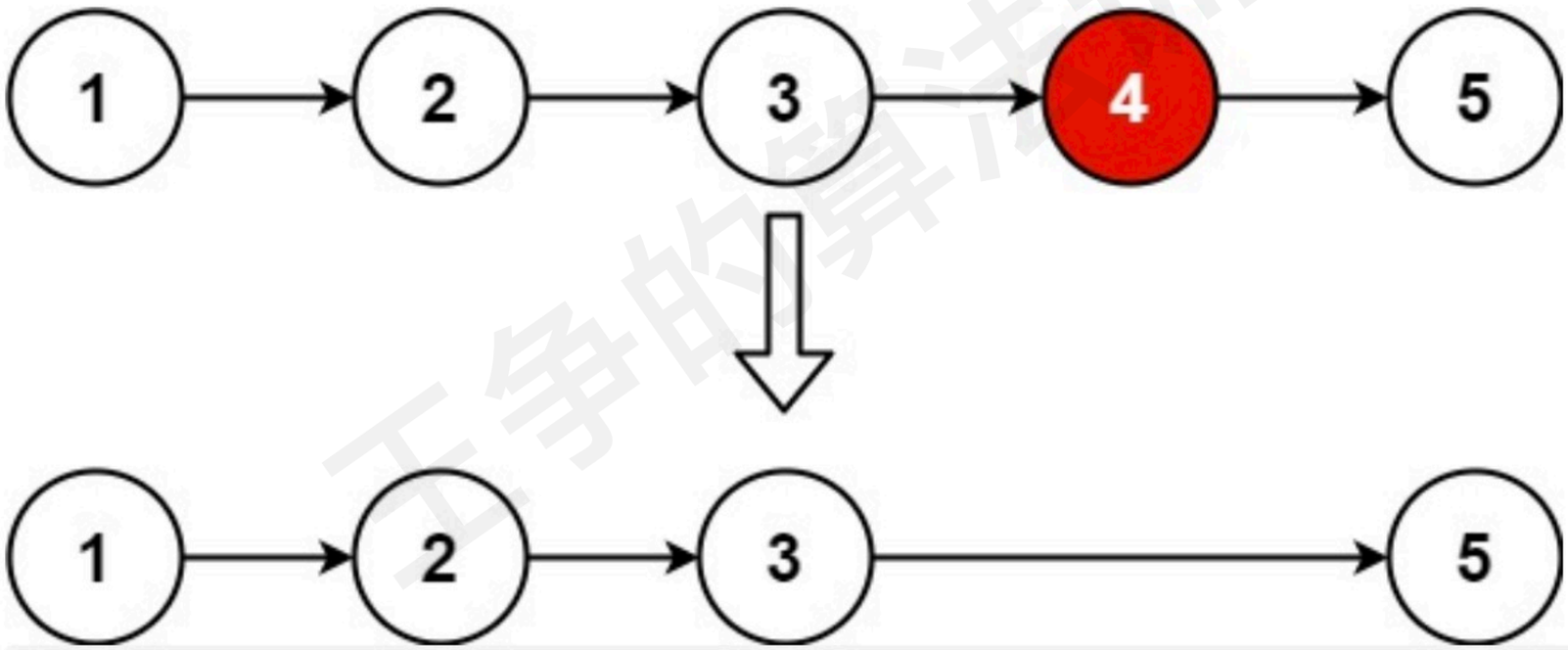
习题课：链表

19. 删除链表的倒数第 N 个结点 （中等）

给你一个链表，删除链表的倒数第 `n` 个结点，并且返回链表的头结点。

进阶：你能尝试使用一趟扫描实现吗?

示例 1：




```
输入：head = [1,2,3,4,5], n = 2
输出：[1,2,3,5]
```

```java
class Solution {
    public ListNode removeNthFromEnd(ListNode head, int n) {
        // 遍历1：fast先到第n个节点处
        ListNode fast = head;
        int count = 0;
        while (fast != null) {
            count++;
            if (count == n) {
                break;
            }
            fast = fast.next;
        }
        if (fast == null) { // 不够k个
            return head;
        }
        // 遍历2：查找pre
        ListNode slow = head;
        ListNode pre = null;
        while (fast.next != null) {
            fast = fast.next;
            pre = slow; // 加了这一行
            slow = slow.next;
        }
        // 删除倒数第n个节点
        if (pre == null) { // 头节点是倒数第n个节点
            head = head.next;
        } else {
            pre.next = slow.next;
        }
        return head;
    }
}
```

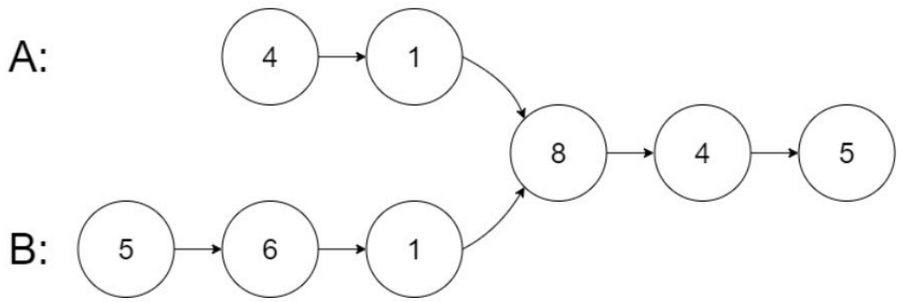
特殊情况处理：头节点、尾节点、空链表

时间复杂度O(n)，空间复杂度O(1)

160. 相交链表（简单）

给你两个单链表的头节点 `headA` 和 `headB` ，请你找出并返回两个单链表相交的起始节点。如果两个链表不存在相交节点，返回 `null` 。

示例 1：




```
输入：intersectVal = 8, listA = [4,1,8,4,5], listB = [5,6,1,8,4,5], skipA = 2, skipB = 3
输出：Intersected at '8'
解释：相交节点的值为 8 （注意，如果两个链表相交则不能为 0）。
从各自的表头开始算起，链表 A 为 [4,1,8,4,5]，链表 B 为 [5,6,1,8,4,5]。
在 A 中，相交节点前有 2 个节点；在 B 中，相交节点前有 3 个节点。
```

```java
public class Solution {
    public ListNode getIntersectionNode(ListNode headA, ListNode headB) {
        // 求链表A的长度na
        int na = 0;
        ListNode pA = headA;
        while (pA != null) {
            na++;
            pA = pA.next;
        }
        // 求链表B的长度nb
        int nb = 0;
        ListNode pB = headB;
        while (pB != null) {
            nb++;
            pB = pB.next;
        }
        // 先让指向长链表的指针多走na-nb或nb-na步
        pA = headA;
        pB = headB;
        if (na >= nb) {
            for (int i = 0; i < na-nb; ++i) {
                pA = pA.next;
            }
        } else {
            for (int i = 0; i < nb-na; ++i) {
                pB = pB.next;
            }
        }
        // 让pA和pB同步前进
        while (pA != null && pB != null && pA != pB) {
            pA = pA.next;
            pB = pB.next;
        }
        if (pA == null || pB == null) return null;
        else return pA;
    }
}
```

时间复杂度O(m+n)，空间复杂度O(1)

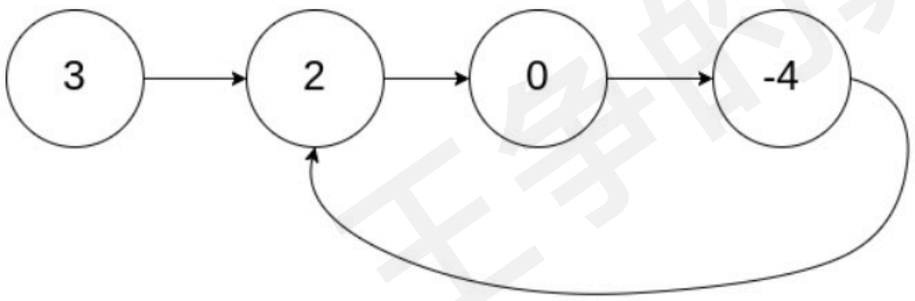
141. 环形链表（简单） 判断链表中是否存在环

给你一个链表的头节点 `head` ，判断链表中是否有环。

如果链表中有某个节点，可以通过连续跟踪 `next` 指针再次到达，则链表中存在环。 为了表示给定链表中的环，评测系统内部使用整数 `pos` 来表示链表尾连接到链表中的位置（索引从 0 开始）。如果 `pos` 是 `-1` ，则在该链表中没有环。**注意：** **`pos` 不作为参数进行传递**，仅仅是为了标识链表的实际情况。

如果链表中存在环，则返回 `true` 。 否则，返回 `false` 。

示例 1：




```
输入：head = [3,2,0,-4], pos = 1
输出：true
解释：链表中有一个环，其尾部连接到第二个节点。
```

141. 环形链表（简单） 判断链表中是否存在环

```
public class Solution {
    public boolean hasCycle(ListNode head) {
        if (head == null) return false;
        ListNode slow = head;
        ListNode fast = head.next;
        while (fast != null && fast.next != null && slow != fast) {
            slow = slow.next;
            fast = fast.next.next;
        }
        if (slow == fast) return true;
        return false;
    }
}
```

时间复杂度O(n)，空间复杂度O(1)

关注微信公众号“**小争哥**”，

后台回复“**PDF**”获取独家算法资料