203. 移除链表元素

```
class Solution:
    def removeElements(self, head: ListNode, val: int) -> ListNode:
        if not head:
            return head
        p1 = head
        while p1.next:
            if p1.next.val == val:
                p1.next = p1.next.next
            else:
                p1 = p1.next
        if head.val == val:
            return head.next
        return head
```

876. 链表的中间结点

```
class Solution:
    def middleNode(self, head: ListNode) -> ListNode:
        slow = fast = head
        while fast and fast.next:
            fast = fast.next.next
            slow = slow.next
        return slow
```

83. 删除排序链表中的重复元素

```
class Solution:
    def deleteDuplicates(self, head: ListNode) -> ListNode:
        if not head or not head.next:
            return head
        newhead = ListNode(-111)
        tail = newhead
        p1 = head
        while p1:
            tmp = p1.next
            if tail.val != p1.val:
                tail.next = p1
                tail = p1
                p1.next = None
            p1 = tmp
        return newhead.next
```

剑指 Offer 25. 合并两个排序的链表

```
class Solution:
    def mergeTwoLists(self, l1: ListNode, l2: ListNode) -> ListNode:
        if not l1 or not l2:
            return l1 or l2
        head = ListNode(0)
        tail = head
        p1 = l1
        p2 = l2
        while p1 and p2:
```

```
            if p1.val <= p2.val:
                tail.next = p1
                tail = p1
                p1 = p1.next
            else:
                tail.next = p2
                tail = p2
                p2 = p2.next
        if p1 or p2:
            tail.next = p1 or p2
        return head.next
```

2. 两数相加

```
class Solution:
    def addTwoNumbers(self, l1: ListNode, l2: ListNode) -> ListNode:
        carry = 0
        p1 = l1
        p2 = l2
        dummyHead = ListNode(0)
        tail = dummyHead
        while p1 or p2 or carry:
            sum = 0
            if p1:
                sum += p1.val
                p1 = p1.next
            if p2:
                sum += p2.val
                p2 = p2.next
            sum += carry
            carry = sum // 10
            tail.next = ListNode(sum%10)
            tail = tail.next
        if carry > 0:
            tail.next = ListNode(carry)
        return dummyHead.next
```

206. 反转链表

```
class Solution:
    def reverseList(self, head: ListNode) -> ListNode:
        #递归
        if not head or not head.next:
            return head
        p1 = head
        node = self.reverseList(p1.next)
        p1.next.next = p1
        p1.next = None
        return node

    def reverseList(self, head: ListNode) -> ListNode:
        # 循环
        newHead = None
        p1 = head
```

```
        while p1:
            temp = p1.next
            p1.next = newHead
            newHead = p1
            p1 = temp
        return newHead
```

234. 回文链表

```
class Solution:
    def isPalindrome(self, head: ListNode) -> bool:
        if head is None or not head.next:
            return True
        # 找到前半部分链表的尾节点并反转后半部分链表
        midNode = self.findMidNode(head)
        rightHalfNode = self.reverseList(midNode.next)
        # 判断是否回文
        p = head
        q = rightHalfNode
        while q:
            if p.val != q.val:
                return False
            q = q.next
            p = p.next
        return True

    def findMidNode(self, head):
        fast = head
        slow = head
        while fast.next is not None and fast.next.next is not None:
            fast = fast.next.next
            slow = slow.next
        return slow

    def reverseList(self, head):
        if not head:
            return head
        newHead = None
        p = head
        while p is not None:
            tmp = p.next
            p.next = newHead
            newHead = p
            p = tmp
        return newHead
```

328. 奇偶链表

```
class Solution:
  def oddEvenList(self, head: ListNode) -> ListNode:
    if not head:
        return head
    oddHead = ListNode()
    oddTail = oddHead
```

```
    evenHead = ListNode()
    evenTail = evenHead
    p = head
    count = 1
    while p:
        tmp = p.next
        if count % 2 == 1:
            p.next = None
            oddTail.next = p
            oddTail = p
        else:
            p.next = None
            evenTail.next = p
            evenTail = p
        count += 1
        p = tmp
    oddTail.next = evenHead.next
    return oddHead.next
```

25.K 个一组翻转链表

```
class Solution:
  def reverse(self, head: ListNode, tail: ListNode):
    newHead = None
    p = head
    while p != tail:
        tmp = p.next
        p.next = newHead
        newHead = p
        p = tmp
    tail.next = newHead
    return tail, head
  def reverseKGroup(self, head: ListNode, k: int) -> ListNode:
    dummyHead = ListNode()
    tail = dummyHead
    p = head
    while p:
        count = 0
        q = p
        while q:
            count += 1
            if count == k:
                break
            q = q.next
        if not q:
            tail.next = p
            return dummyHead.next
        else:
            tmp = q.next
            nodes = self.reverse(p,q)
            tail.next = nodes[0]
            tail = nodes[1]
            p = tmp
    return dummyHead.next
```

剑指 Offer 22. 链表中倒数第k个节点

```
class Solution:
    def getKthFromEnd(self, head: ListNode, k: int) -> ListNode:
        fast = head
        count = 0
        while fast:
            count += 1
            if count == k:
                break
            fast = fast.next
        if not fast:
            return None
        slow = head
        while fast.next:
            slow = slow.next
            fast = fast.next
        return slow
```

19. 删除链表的倒数第 N 个结点

```
class Solution:
  def removeNthFromEnd(self, head: ListNode, n: int) -> ListNode:
    fast = head
    count = 0
    while fast:
        count += 1
        if count == n:
            break
        fast = fast.next
    if not fast:
        return head
    slow = head
    pre = None
    while fast.next:
        fast = fast.next
        pre = slow
        slow = slow.next
    if not pre:
        head = head.next
    else:
        pre.next = slow.next
    return head
```

20. 相交链表

```
class Solution:
  def getIntersectionNode(self, headA: ListNode, headB: ListNode) -> ListNode:
    na = 0
    pA = headA
    while pA:
        na += 1
        pA = pA.next
    nb = 0
```

```
    pB = headB
    while pB:
        nb += 1
        pB = pB.next
    pA = headA
    pB = headB
    if na >= nb:
        for i in range(na - nb):
            pA = pA.next
    else:
        for i in range(nb - na):
            pB = pB.next
    while pA and pB and pA != pB:
        pA = pA.next
        pB = pB.next
    if not pA or not pB:
        return None
    else:
        return pA
```

141. 环形链表

```
class Solution:
  def hasCycle(self, head: ListNode) -> bool:
    if not head:
        return head
    slow = head
    fast = head.next
    while fast and fast.next and slow != fast:
        fast = fast.next.next
        slow = slow.next
    if slow == fast:
        return True
    return False
```