

- 1.两两交换链表中的节点
- 2.翻转链表
- 3.重排链表
- 4.翻转链表
- 5.K个一组翻转链表
- 6.环形链表
- 7.环形链表2
- 8.翻转链表2
- 9.删除链表的倒数第K个节点
- 10.两个链表的第一个公共节点

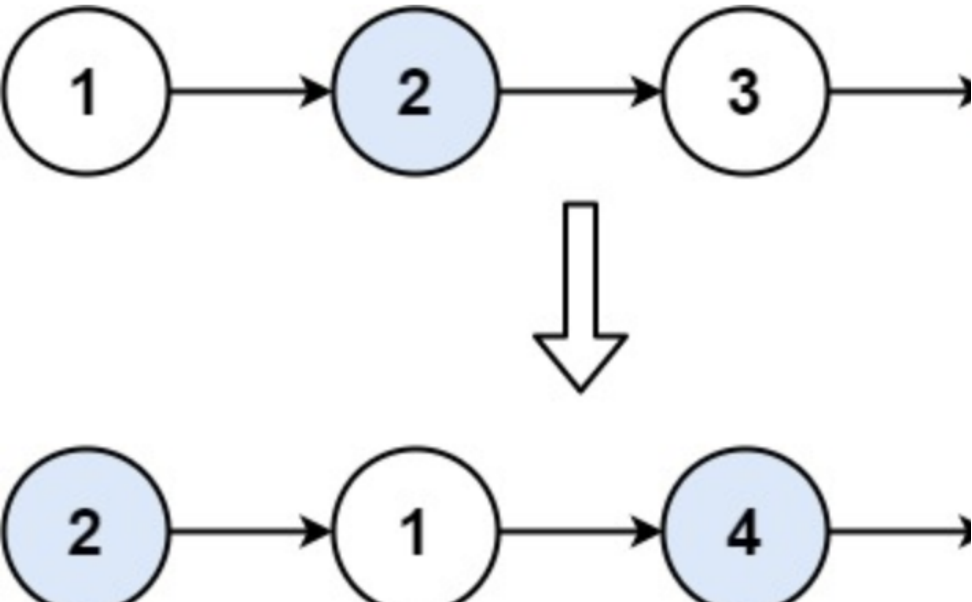
24. 两两交换链表中的节点

难度 中等 997 收藏 分享 切换为

给定一个链表，两两交换其中相邻的节点，并返回交换后的链表。

你不能只是单纯的改变节点内部的值，而是需要实际的进行节点交换。

示例 1:



输入: head = [1,2,3,4]
输出: [2,1,4,3]

示例 2:

输入: head = []
输出: []

class Solution:

```

def swapPairs(self, head: ListNode) -> Lis
    if not head:
        return None
    dummy = ListNode(-1)
    dummy.next = head
    pre, tail = dummy, dummy
    while True:
        count = 2
        while count and tail:
            tail = tail.next
            count -= 1
        if not tail:
            break
        tmp = pre.next
        while pre.next != tail:
            cur = pre.next
            pre.next = cur.next
            cur.next = tail.next
            tail.next = cur
        pre = tmp
        tail = tmp
    return dummy.next

```

206. 反转链表

难度 简单

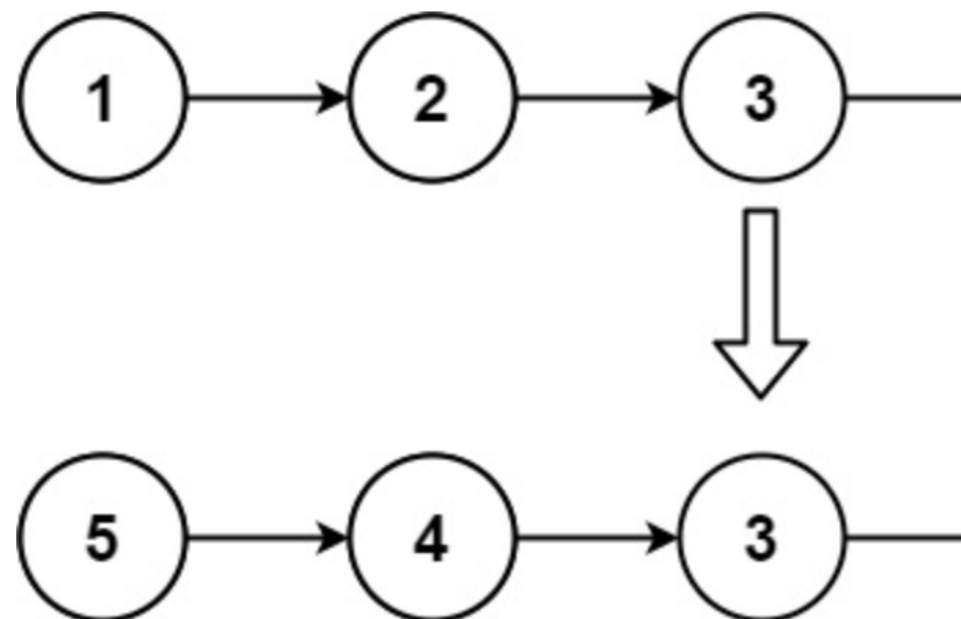
👍 1887

☆ 收藏

📄 分享

🔗 切换语言

给你单链表的头节点 `head`，请你反转链表，并返回反转后的头节点。

示例 1:

输入: `head = [1,2,3,4,5]`

输出: `[5,4,3,2,1]`

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next

class Solution:
    def reverseList(self, head: ListNode) -> ListNode:
        pre = None
        cur = head
        while cur:
            tmp = cur.next
            cur.next = pre
            pre = cur
            cur = tmp
        return pre
```

143. 重排链表

难度 中等

👍 639

☆ 收藏

📄 分享

🌐 切换为英

给定一个单链表 L 的头节点 $head$ ，单链表 L 表示为：

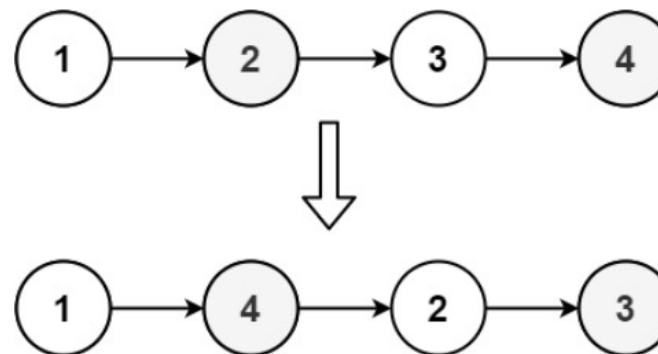
$$L_0 \rightarrow L_1 \rightarrow \dots \rightarrow L_{n-1} \rightarrow L_n$$

请将其重新排列后变为：

$$L_0 \rightarrow L_n \rightarrow L_1 \rightarrow L_{n-1} \rightarrow L_2 \rightarrow L_{n-2} \rightarrow \dots$$

不能只是单纯的改变节点内部的值，而是需要实际的进行节点交

示例 1:



输入: $head = [1, 2, 3, 4]$

输出: $[1, 4, 2, 3]$

```
class Solution:
```

```
    def reorderList(self, head: ListNode) -> None:
```

```
        11 = head
```

```

l1 = head
mid = self.findmin(l1)
l2 = mid.next
mid.next = None
l2 = self.reverse(l2)
self.merge(l1, l2)

```

```

def findmin(self, head):
    slow, fast = head, head
    while fast.next and fast.next.next:
        slow = slow.next
        fast = fast.next.next
    return slow

```

```

def reverse(self, head):
    pre = None
    cur = head
    while cur:
        tmp = cur.next
        cur.next = pre
        pre = cur
        cur = tmp
    return pre

```

```

def merge(self, l1, l2):
    while l1 and l2:
        l1_tmp = l1.next
        l2_tmp = l2.next

```


```

l2_tmp = l2.next
l1.next = l2
l1 = l1_tmp
l2.next = l1
l2 = l2_tmp

```

K个一组翻转链表

25. K 个一组翻转链表

难度 **困难** 1235

☆ 收藏

 分享

🌐 切换为英

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

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
- **你不能只是单纯的改变节点内部的值，而是需要实际进行**

示例 1：

输入：head = [1,2,3,4,5], $k = 2$

输出：[2,1,4,3,5]

示例 2：

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

示例 3:

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

class Solution:

```
def reverseKGroup(self, head: ListNode, k: int) -> ListNode:
    dummy = ListNode(-1)
    dummy.next = head
    pre = dummy
    tail = dummy
    while True:
        count = k
        while count and tail:
            tail = tail.next
            count -= 1
        if not tail:
            break
        tmp = pre.next
        while pre.next != tail:
            cur = pre.next
            pre.next = cur.next
            cur.next = tail.next
            tail.next = cur
        pre = tmp
        tail = tmp
    return dummy.next
```

141. 环形链表

难度 简单

👍 1163

☆ 收藏

📄 分享

🔗 切换为

给定一个链表，判断链表中是否有环。

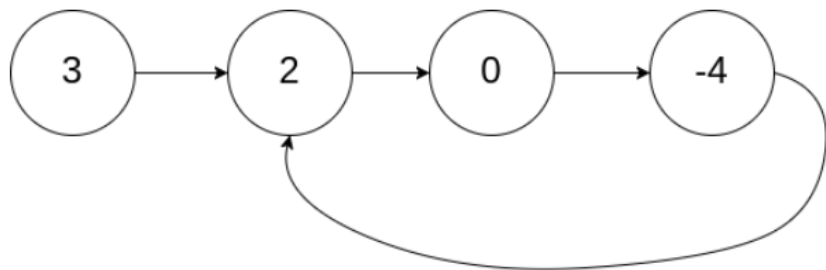
如果链表中有某个节点，可以通过连续跟踪 `next` 指针再次到表中的环，我们使用整数 `pos` 来表示链表尾连接到链表中的位置 `-1`，则在该链表中没有环。**注意：** `pos` 不作为参数进行传递

如果链表中存在环，则返回 `true`。否则，返回 `false`。

进阶：

你能用 $O(1)$ （即，常量）内存解决此问题吗？

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。

```

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

1.为什么slow走一步，fast走两步，他们一定会在环里相遇，会不会永远追不上，请证明！

不会，假设slow进环的时候，fast跟slow的距离是 N ，紧接着追击过程中，fast往前走2步，slow走1步，每走一次，他们之间距离缩小1.....，缩小到0时相遇。

2.slow走一步，fast走3步？走4步？走 x 步行不行？为什么？请证明！

假设slow进环的时候，fast跟slow的距离是 N ，紧接着追击过程中，fast往前走3步，slow走1步，每走一次，他们之间距离缩小2

若 N 为偶数： $N, N-2, N-4, \dots, 2, 0$ （能追上）

若 N 为奇数： $N, N-2, N-4, \dots, 1, -1$ （差距是 $C-1$, C 是环的长度，如果 $C-1$ 恰好为奇数，那么永远追不上）

结论：如果slow进环时，slow跟fast的差距 N 是奇数，且环的长度是偶数，那么他们两在环里面就会一直打圈，不会相遇。

142. 环形链表 II

难度 中等 1101 收藏 分享 切换为:

给定一个链表，返回链表开始入环的第一个节点。如果链表无环，返回 null。

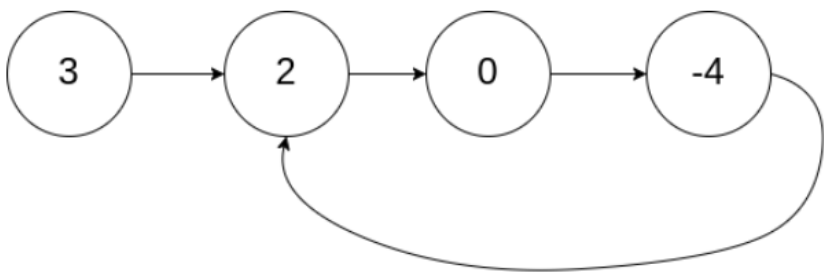
为了表示给定链表中的环，我们使用整数 `pos` 来表示链表尾节点连接到列表中的哪个下标。如果 `pos` 是 `-1`，则在该链表中没有环。**注意，`pos` 仅仅是为了测试用例中，它只存在于函数参数中。**

说明：不允许修改给定的链表。

进阶：

- 你是否可以使用 $O(1)$ 空间解决此题？

示例 1：



输入: `head = [3,2,0,-4]`, `pos = 1`

输出: 返回索引为 1 的链表节点

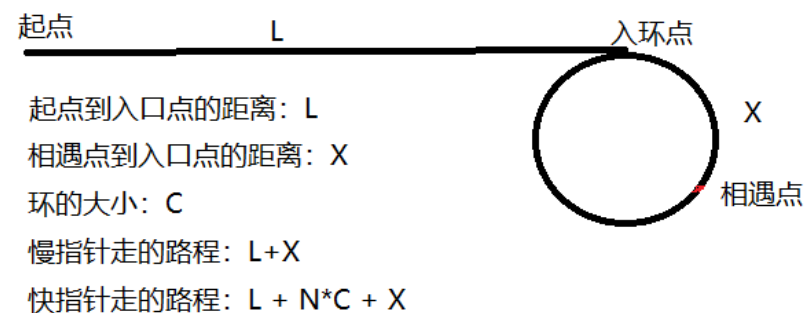
解释: 链表中有一个环，其尾部连接到第二个节点。

class Solution:

```

def detectCycle(self, head: ListNode) -> ListNode:
    slow, fast = head, head
    while True:
        if not fast or not fast.next:
            return None
        slow = slow.next
        fast = fast.next.next
        if slow == fast:
            break
    fast = head
    while fast != slow:
        slow = slow.next
        fast = fast.next
    return fast

```



$$L + N \cdot C + X = 2(L + X)$$

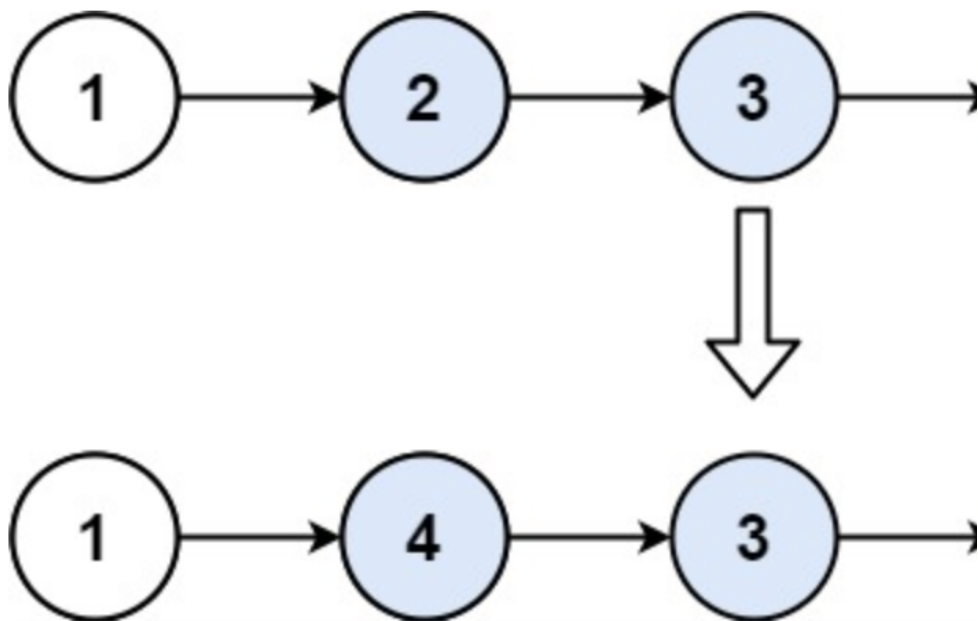
https://blog.csdn.net/weixin_41446512

92. 反转链表 II

难度 中等 984 ☆ 收藏 分享 切换为

给你单链表的头指针 `head` 和两个整数 `left` 和 `right`，置 `left` 到位置 `right` 的链表节点，返回 **反转后的链表**。

示例 1:



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

示例 2:

输入: `head = [5]`, `left = 1`, `right = 1`
输出: `[5]`

class Solution:

```
def reverseBetween(self, head: ListNode, left: int, right: int):
    if left == right:
        return head
    dummy = ListNode(-1)
    dummy.next = head
    pre, tail = dummy, dummy
    for i in range(left - 1):
        pre = pre.next
    for i in range(right):
        tail = tail.next
    # tmp = pre.next
    while pre.next != tail:
        cur = pre.next
        pre.next = cur.next
        cur.next = tail.next
        tail.next = cur
    return dummy.next
```

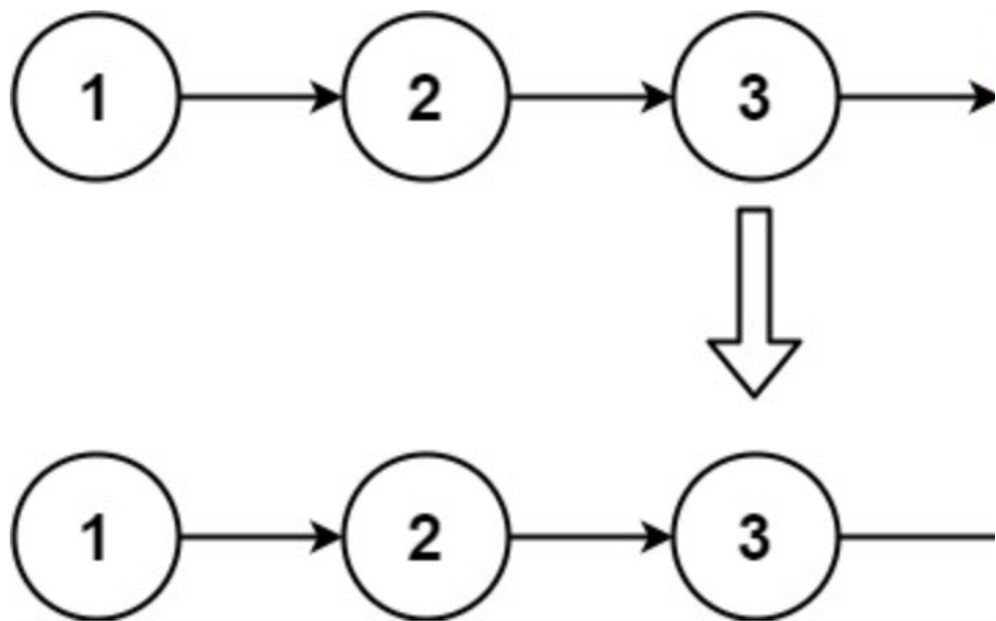
19. 删除链表的倒数第 N 个结点

难度 中等 1501 ☆ 收藏 分享 切换为

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

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]

示例 2:

输入: head = [1], n = 1

输出: []

```
class Solution:
```

```
def removeNthFromEnd(self, head: ListNode,
    dummy = ListNode(-1)
    dummy.next = head
    slow = dummy
    fast = head
    for i in range(n):
        fast = fast.next
    while fast:
        fast = fast.next
        slow = slow.next
    cur = slow.next
    slow.next = cur.next
    return dummy.next
```

NC66 两个链表的第一个公共结点

简单

通过率: 35.62%

时间限制: 1秒

空间限制:

知识点: 链表

[题目](#)
[题解\(105\)](#)
[讨论\(1k\)](#)
[排行](#)

描述

输入两个无环的单链表，找出它们的第一个公共结点。（注：其他方式显示的，保证传入数据是正确的）

示例1

输入: {1,2,3},{4,5},{6,7}

返回值: {6,7}

说明: 第一个参数{1,2,3}代表是第一个链表非公共部分, 最后的{6,7}表示的是2个链表的公共部分
这3个参数最后在后台会组装成为2个两个无环的

示例2

输入: {1},{2,3},{}

返回值: {}

说明: 2个链表没有公共节点, 返回null, 后台打印{

```
class Solution:
    def FindFirstCommonNode(self, pHead1,
        # write code here
        if not pHead1 or not pHead2:
            return
        p1 = pHead1
        p2 = pHead2
        while p1 != p2:
            p1 = p1.next
            p2 = p2.next
            if p1 != p2:
                if not p1:
                    p1 = pHead2
                if not p2:
                    p2 = pHead1
        return p1
```

合并两个有序链表

21. 合并两个有序链表

难度 **简单**

🔖 1848

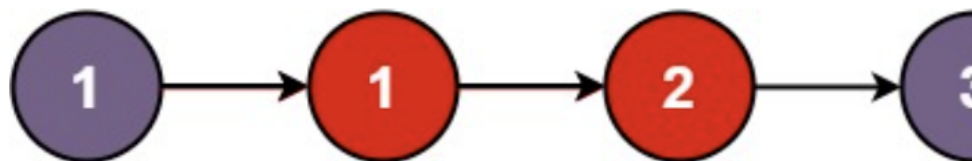
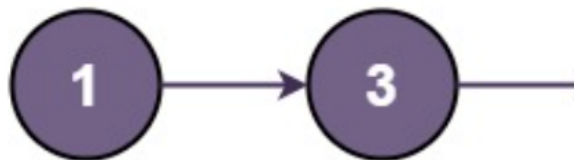
☆ 收藏

📄 分享

🔗 切换为

将两个升序链表合并为一个新的 **升序** 链表并返回。新链表是新的。

示例 1:



输入: l1 = [1,2,4], l2 = [1,3,4]

输出: [1,1,2,3,4,4]

```
class Solution:
```

```
    def mergeTwoLists(self, l1: ListNode, l2:
```

```
        dummy = cur = ListNode(0)
```

```
        while l1 and l2:
```

```
            if l1.val < l2.val:
```

```
                cur.next = l1
```

```
                l1 = l1.next
```

```
            else:
```

```
                cur.next = l2
```

```
                l2 = l2.next
```

```
            cur = cur.next
```

```
        cur.next = l1 or l2
```

```
        return dummy.next
```