代码随想录 Chapter2 Linked List

Day3 | 203.移除链表元素, 707.设计链表, 206.反转链表

基础知识

https://programmercarl.com/链表理论基础.html

性质：

- linked list 跟 set 一样，没有 index. 不能通过 index 找 value

  array 中可以通过 index 找 value　　nums[1] = 3

  linked list 没有 index，通过 next，只能看到下一个

- 1个 node 只能引出 1条 next （1个车厢 接 1个车厢）
- 长度未知
- head 初始时指向 头结点 （head node）

  ↑

  地址

性能分析

再把链表的特性和数组的特性进行一个对比，如图所示：

	插入/删除（时间复杂度）	查询（时间复杂度）	适用场景
数组	O(n)	O(1)	数据量固定，频繁查询，较少增删
链表	O(1)	O(n)	数据量不固定，频繁增删，较少查询

数组在定义的时候，长度就是固定的，如果想改动数组的长度，就需要重新定义一个新的数组。

链表的长度可以是不固定的，并且可以动态增删， 适合数据量不固定，频繁增删，较少查询的场景。

相信大家已经对链表足够的了解，后面我会讲解关于链表的高频面试题目，我们下期见！

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

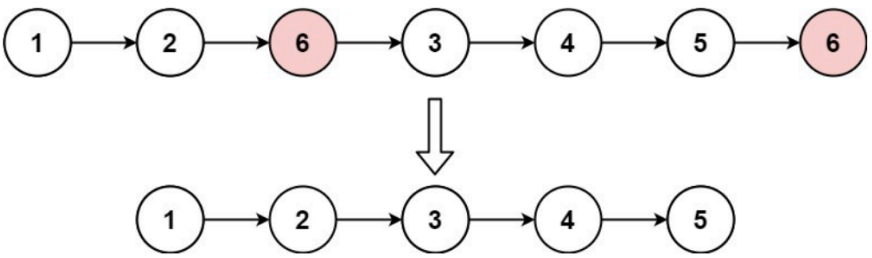
203. Remove Linked List Elements

已解答

简单 相关标签 相关企业 文A

Given the `head` of a linked list and an integer `val`, remove all the nodes of the linked list that has `Node.val == val`, and return *the new head*.

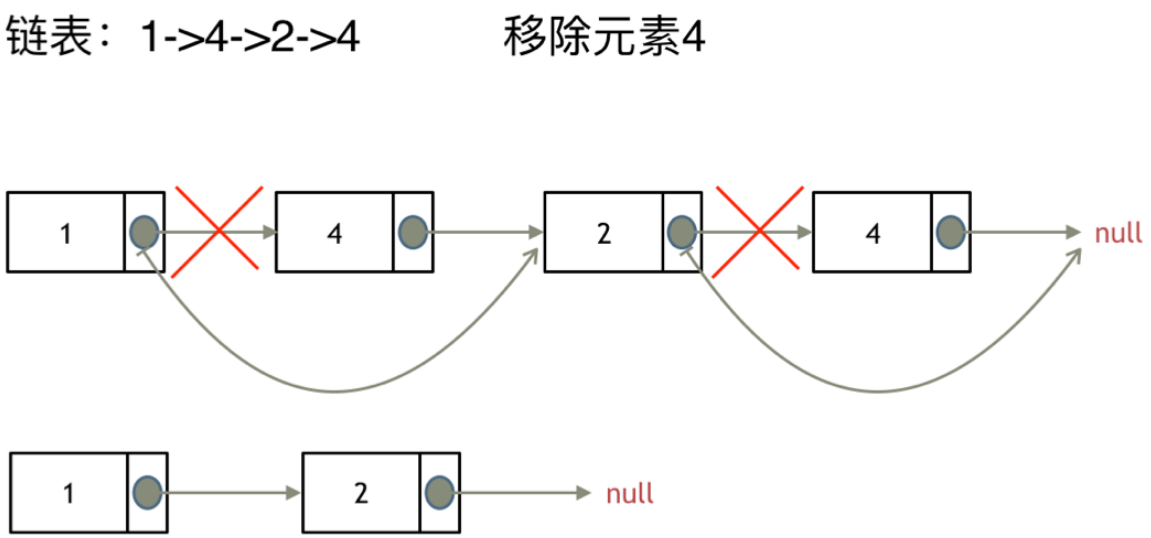
Example 1:



> **Input:** head = [1,2,6,3,4,5,6], val = 6
> **Output:** [1,2,3,4,5]

思路

这里以链表 1 4 2 4 来举例，移除元素4。



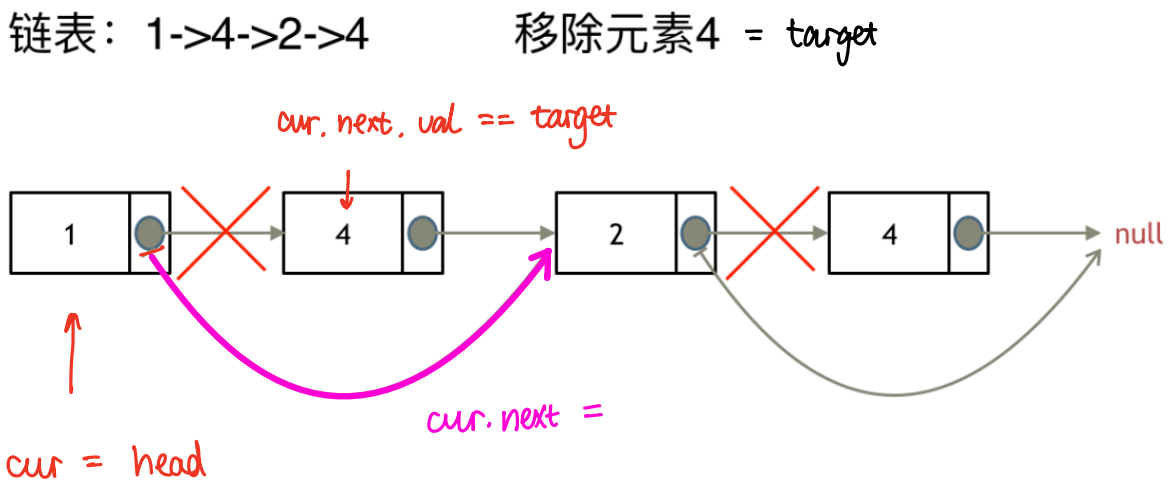
这种情况下的移除操作，就是让节点next指针直接指向下一个节点就可以了，

那么因为单链表的特殊性，只能指向下一个节点，刚刚删除的是链表的中第二个，和第四个节点。

那么如果删除的是头结点又该怎么办呢？这里就涉及如下链表操作的两种方式：

1. 直接使用原来的链表来进行删除操作
2. 设置一个虚拟头结点在进行删除操作

法1: 直接使用原来的链表来进行删除操作




```
# 删除头结点
while head and head.val == target:
    head = head.next          # 改变头结点

# 删除非头结点
cur = head                    # cur 不能是 head.next, 因为我们如果想删除 head.next,
                              我们需要知道它上一个节点的地址
while cur and cur.next:       # 相当于 != None
    if cur.next.val == target:
        cur.next = cur.next.next      # 紫色操作
    else:
        cur = cur.next                # cur 向下走一位
return head
```

注意：

`cur = head`

1. 我们不能改变原来head的值，所以不能用head 遍历，要定义一个临时的cur指针来遍历链表。
2. cur = head 而不是head.next, 因为单链表只能往后找，不能往前找。

`return head`

为什么return head, head到底是什么？
下图是我 print head 出来的东西，可以看到head其实就包含了整个链表的所有信息，而非只是那一个node的信息。

输入

head =

```
[1,2,6,3,4,5,6]
```

val =

```
6
```

标准输出

```
ListNode{val: 1, next: ListNode{val: 2, next: ListNode{val: 6, next: ListNode{
val: 3, next: ListNode{val: 4, next: ListNode{val: 5, next: ListNode{val: 6, n
ext: None}}}}}}
```

每次往后一个node，就减少一个node的信息。比如再print一个head.next:

```
6  class Solution:
7      def removeElements(self, head: Optional[ListNode], val: int) -> Optional
   [ListNode]:
8          cur = head
9          while cur and cur.next:
10             cur = cur.next
11             print(cur)
12             break
```

标准输出

```
ListNode{val: 2, next: ListNode{val: 6, next: ListNode{val: 3, next: ListNode{
val: 4, next: ListNode{val: 5, next: ListNode{val: 6, next: None}}}}}
```

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

class Solution:
    def removeElements(self, head: Optional[ListNode], val: int) -> Optional[ListNode]:
        # 法1: 直接使用原来的链表来进行删除操作
        # 删除head node
        while head and head.val == val:
            head = head.next

        # 删除非head node
        # 对于每一个要删除的节点，都要找它上一个节点的位置，并让上一个节点指向下一个节点
        # 如果cur 从 head.next 开始，那么如果正好要删除这个node，就找不到它的上一个节点了
        cur = head
        while cur and cur.next:
            if cur.next.val == val:
                cur.next = cur.next.next # delete
            else:
                cur = cur.next # 继续往下遍历

        return head
```

[法2]：创建虚拟头结点（建议之后统一使用这个）

好处：在操作链表时，删除和增加 node 的方式都统一（不用判断删的是否是头结点）

```
dummyhead = new node (next = head)
cur = dummyhead          # 与法1同理，不能是 dummyhead.next
while cur and cur.next :          # 不为空
    if cur.next.val == target:
        cur.next = cur.next.next      # 删除
    else :
        cur = cur.next          # 继续遍历
return dummyhead.next          # 新链表的头结点
```

```python
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def removeElements(self, head: Optional[ListNode], val: int) ->
        # 法2: 创建虚拟头结点
        dummyhead = ListNode(val=0, next = head)

        # 统一删除方法
        cur = dummyhead
        while cur and cur.next:
            if cur.next.val == val:
                cur.next = cur.next.next # delete
            else:
                cur = cur.next # continue to traverse

        return dummyhead.next # 这个才是新链表的头结点
```

LC 707 Design Linked List (medium)

707. Design Linked List

中等 相关标签 相关企业

Design your implementation of the linked list. You can choose to use a singly or doubly linked list.
A node in a singly linked list should have two attributes: `val` and `next`. `val` is the value of the current node, and `next` is a pointer/reference to the next node.
If you want to use the doubly linked list, you will need one more attribute `prev` to indicate the previous node in the linked list. Assume all nodes in the linked list are **0-indexed**.

Implement the `MyLinkedList` class:

- `MyLinkedList()` Initializes the `MyLinkedList` object.
- `int get(int index)` Get the value of the `index`th node in the linked list. If the index is invalid, return `-1`.
- `void addAtHead(int val)` Add a node of value `val` before the first element of the linked list. After the insertion, the new node will be the first node of the linked list.
- `void addAtTail(int val)` Append a node of value `val` as the last element of the linked list.
- `void addAtIndex(int index, int val)` Add a node of value `val` before the `index`th node in the linked list. If `index` equals the length of the linked list, the node will be appended to the end of the linked list. If `index` is greater than the length, the node **will not be inserted**.
- `void deleteAtIndex(int index)` Delete the `index`th node in the linked list, if the index is valid.

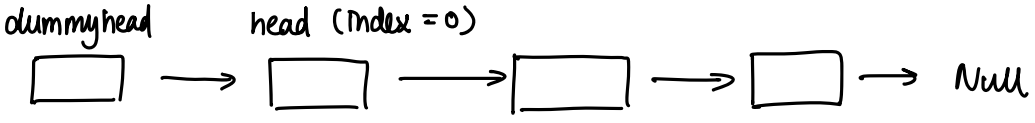
Example 1:

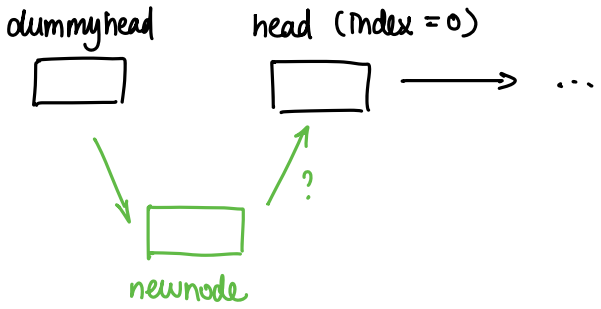
```
Input
["MyLinkedList", "addAtHead", "addAtTail", "addAtIndex", "get", "deleteAtIndex",
"get"]
[[], [1], [3], [1, 2], [1], [1], [1]]
Output
[null, null, null, null, 2, null, 3]

Explanation
MyLinkedList myLinkedList = new MyLinkedList();
myLinkedList.addAtHead(1);
myLinkedList.addAtTail(3);
myLinkedList.addAtIndex(1, 2);    // linked list becomes 1->2->3
myLinkedList.get(1);              // return 2
myLinkedList.deleteAtIndex(1);    // now the linked list is 1->3
myLinkedList.get(1);              // return 3
```

这一道题包括3链表的基本操作！

我的建议是先看 add at head 和 add at tail，不然不知道 self.size 是哪里来的.

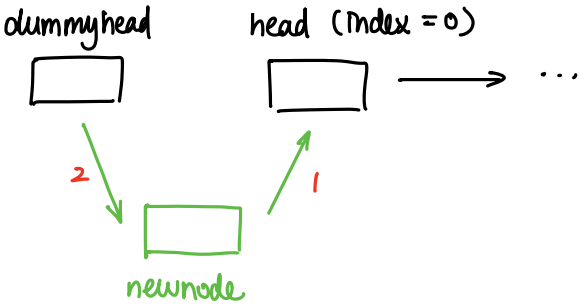
① addAtHead 头部插入节点





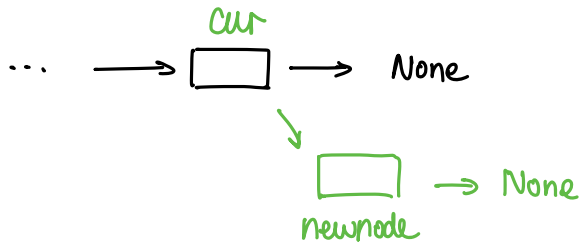

```
newnode = ListNode()
dummyhead.next = newnode
newnode.next = 找不到 head 了   X
```

正确的操作顺序：



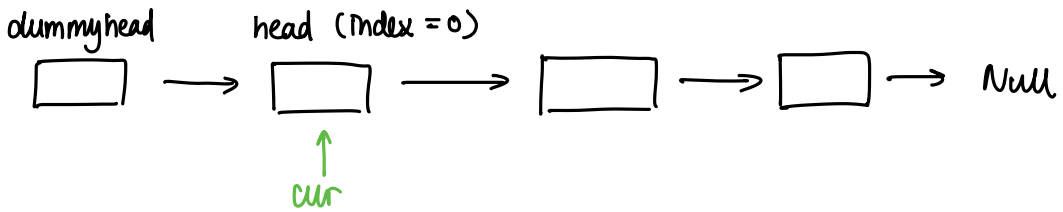

```
newnode = ListNode()
newnode.next = dummyhead.next
dummyhead.next = newnode
```

② Add at Tail 尾部插入节点




```
cur = dummyhead
# 直到 cur.next 为空时，找到 last node
while cur.next != None:
    cur = cur.next
# cur 此时指向原尾部结点
cur.next = newnode
# 我们在定义一个新 node 时默认
node.next = None
```

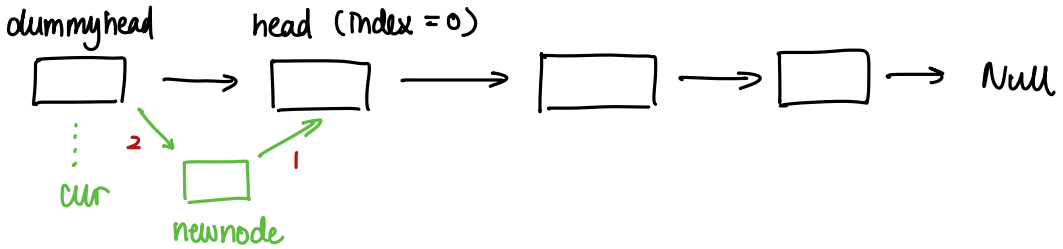
③ get(index) 获取第 n 个节点的值




```
cur = dummyhead.next      # 指向原 head node
                          # 不能改变 head，因为最后要 return head
while index != 0:         # index = 0 不进入循环
    cur = cur.next
```

```
        index -= 1
    return cur.val
```

④ addAtIndex 在第 n 个节点前插入

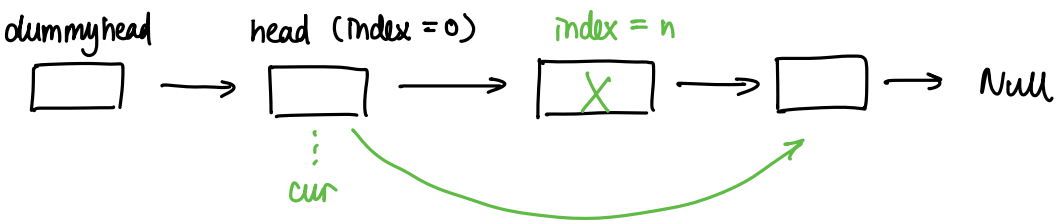



```
cur = dummyhead
```

当 index = 0，则需保证第 0 个 node 是 cur.next
这样才能在 cur 之后插入新节点
总之，保证第 n 个 node 是 cur.next！ ☆

```
# 先找到第 n 个节点
while index != 0:
    cur = cur.next
    index -= 1
# 插入节点的操作与 addAtHead 相同
newnode.next = cur.next
cur.next = newnode
size += 1
```

⑤ deleteAtIndex 删除第 n 个节点



核心：the n^{th} node is cur.next !

```
cur = dummyhead          # n=0 时，index=0 is cur.next
while index != 0:
    cur = cur.next
    index -= 1
# 删除节点
cur.next = cur.next.next
size -= 1
```

总结：对于 singly linked list，只能通过前一个节点找后一个，所以要明确第 n 个 node = cur.next

完整代码：

```python
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

class MyLinkedList:
    # method1: singly linked list
    def __init__(self):
        # 默认val=0, next=None
        self.dummmyhead = ListNode()
        self.size = 0

    def get(self, index: int) -> int:
        if index < 0 or index > self.size - 1:
            # index = self.size - 1 正好是last node
            return -1
        cur = self.dummmyhead.next # the real head
        while index != 0:
            cur = cur.next
            index -= 1
        return cur.val

    def addAtHead(self, val: int) -> None:
        # construct our newnode
        newnode = ListNode(val = val, next = None)
        newnode.next = self.dummmyhead.next
        self.dummmyhead.next = newnode
        self.size += 1

    def addAtTail(self, val: int) -> None:
        newnode = ListNode(val)
        # find the last node
        cur = self.dummmyhead
        while cur.next: #!= None
            cur = cur.next
        # now cur 指向原本的尾部节点
        cur.next = newnode
        self.size += 1
```

```
40      def addAtIndex(self, index: int, val: int) -> None:
41          # check if index is valid
42          if index < 0 or index > self.size:
43              # index = self.size 代表append the node to the end of the list
44              return None
45
46          newnode = ListNode(val)
47          cur = self.dummmyhead
48          # find the n-th node s.t. the n-th node is cur.next
49          while index != 0:
50              cur = cur.next
51              index -= 1
52          # add the newnode
53          newnode.next = cur.next
54          cur.next = newnode
55          self.size += 1
56
57      def deleteAtIndex(self, index: int) -> None:
58          # check if index is valid
59          if index < 0 or index > self.size - 1:
60              return None
61
62          cur = self.dummmyhead
63          # find the index-th node s.t. the index-th node is cur.next
64          while index != 0:
65              cur = cur.next
66              index -= 1
67          # delete the node
68          cur.next = cur.next.next
69          self.size -= 1
```

How to debug ?

```
71      # debug print statement
72      def printList(self) -> None:
73          cur = self.dummmyhead.next
74          while cur:
75              print(cur.val, end = '->')
76              cur = cur.next
77          print('None')
78
79  mylist = MyLinkedList()
80  mylist.addAtHead(1)
81  mylist.addAtTail(3)
82  mylist.printList() # 1->3->None
83  mylist.addAtIndex(1, 2)
84  mylist.printList() # 1->2->3->None
85
```

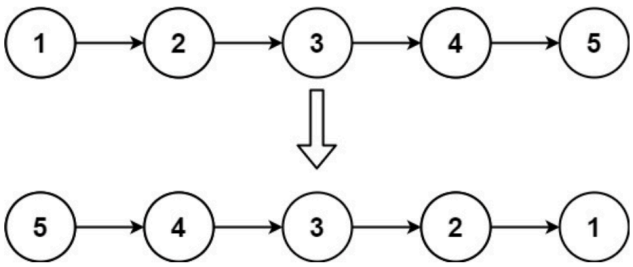
LC 206 Reverse Linked List (easy)

206. Reverse Linked List

简单 相关标签 相关企业

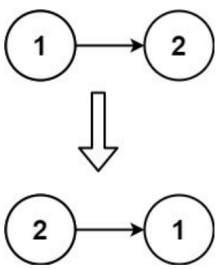
Given the `head` of a singly linked list, reverse the list, and return *the reversed list*.

Example 1:



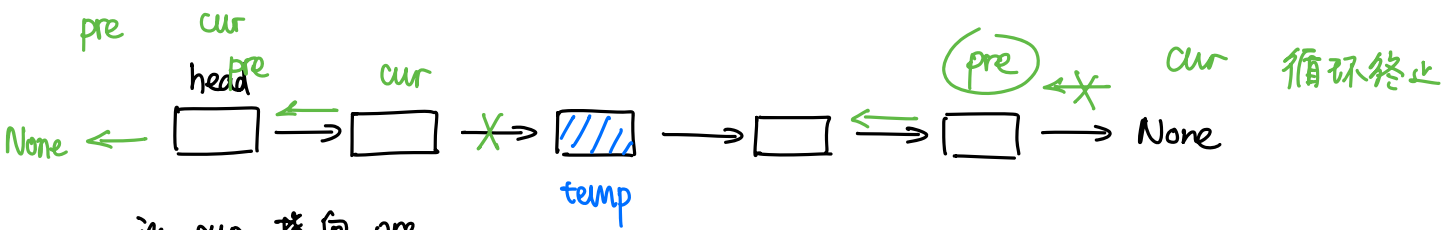

```
Input: head = [1,2,3,4,5]
Output: [5,4,3,2,1]
```

Example 2:




```
Input: head = [1,2]
Output: [2,1]
```

[法1]：双指针（先掌握）



让 cur 指向 pre

pre，cur 同时右移一位

```python
class Solution:
    def reverseList(self, head: Optional[ListNode]) -> Optional[ListNode]:
        # 法1: 双指针
        # initialize
        cur = head
        pre = None
        # while cur is None, the loop ends
        while cur:
            # 在改变cur.next之前，先用temp保存，之后移动指针用
            temp = cur.next
            # 交换
            cur.next = pre
            # 移动指针
            pre = cur
            cur = temp

        # return新链表的头节点
        return pre
```

顺序不能换

时间复杂度：O(N)　N为链表长度

[法2]：递归 recursion

递归的代码是从双指针 -- 对应来的，不要直接写。

```python
def reverseList(self, head: Optio
    # 法1: 双指针
    # initialize
    cur = head
    pre = None
    # while cur is None, the loop
    while cur:
        # 在改变cur.next之前，先用te
        temp = cur.next
        # 交换
        cur.next = pre
        # 移动指针
        pre = cur
        cur = temp

    # return新链表的头节点
    return pre
```

```python
def reverseList(self, head: Optional[ListNode])

    return self.reverse(head, None)
                        cur   pre

def reverse(cur, pre):
    # 当 cur 为空，结束，return pre
    if cur == None: return pre
    temp = cur.next
    cur.next = pre
    # 这里进入下一层 recursion
    return reverse(temp, cur)
```

```python
# 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: Optional[ListNode]) -> Optional[ListNode]:
        return self.reverse(head, None)

    def reverse(self, cur: ListNode, pre: ListNode) -> Optional[ListNode]:
        if cur == None:
            return pre

        temp = cur.next
        cur.next = pre
        # 进入下一层recursion
        return self.reverse(temp, cur)

```

注：这是 reverse 是整个 class solution 的一个 method，如果要 call，一定要加 "self. reverse (cur, pre)"

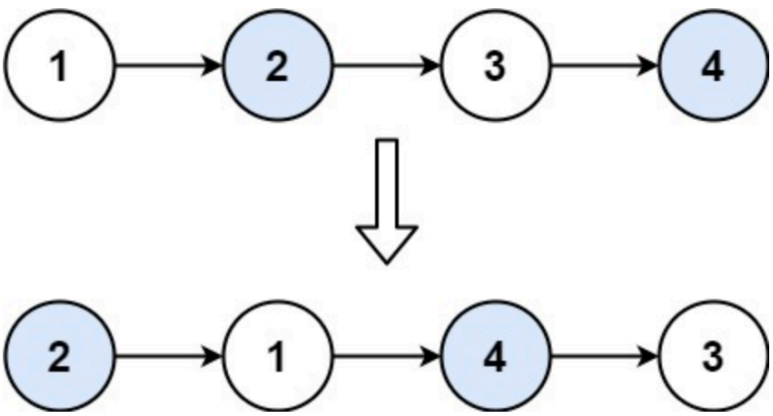
Day 4 LinkedList 02 | 24. 两两交换，19. 删除倒数第 n 个结点，160. 链表相交，142. 环形链表 II

24. Swap Nodes in Pairs

中等 相关标签 相关企业 文A

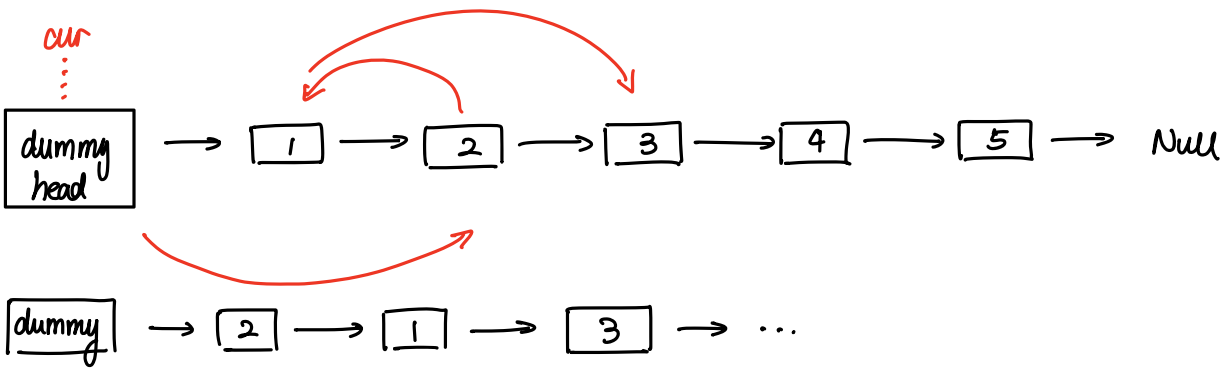
Given a linked list, swap every two adjacent nodes and return its head. You must solve the problem without modifying the values in the list's nodes (i.e., only nodes themselves may be changed.)

Example 1:




```
Input: head = [1,2,3,4]
Output: [2,1,4,3]
```

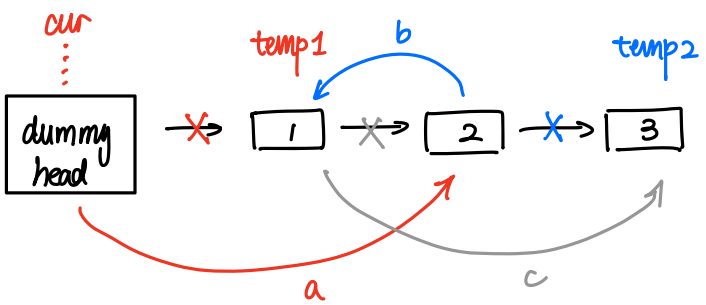
- 思路：cur 指针法



核心 1：cur 指针要指向要反转的两个 node 的前一个 node！

```
#### 思路
dummyhead = ListNode(val=0, next=head)
cur = dummyhead

# 遍历的终止条件:
# 如果链表节点是奇数，则最后一个不做交换，即: cur.next.next = None 遍历结束
# 如果链表节点是偶数，则最后一个做交换，即: cur.next = None 遍历结束 （也包括链表size = 0，空链表）
while cur.next and cur.next.next:
    # 先检查cur.next再检查cur.next.next

    # 保存
    temp1 = cur.next
    temp2 = cur.next.next.next
    # 交换
    cur.next = cur.next.next    #(a)
    cur.next.next = temp1       #(b)
    cur.next.next.next = temp2  #(c)
```

核心2：用 temp 指针保存结果

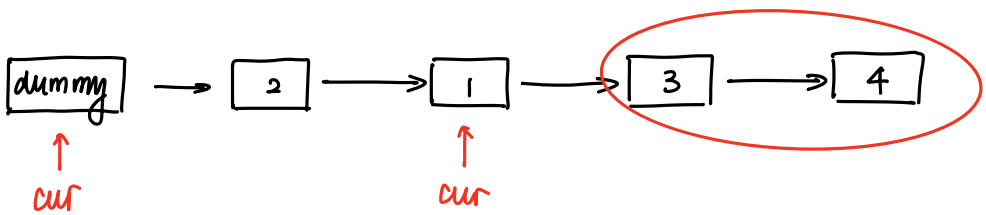


① 改变 dummyhead.next 指向之前：
保存 node 1

② 改变 node2.next 指向之前：
保存 node 3
↑
保存原指向

（如果可以获取到节点位置，则不用 temp 指针保存）

```
# 移动指针
cur = cur.next.next
```



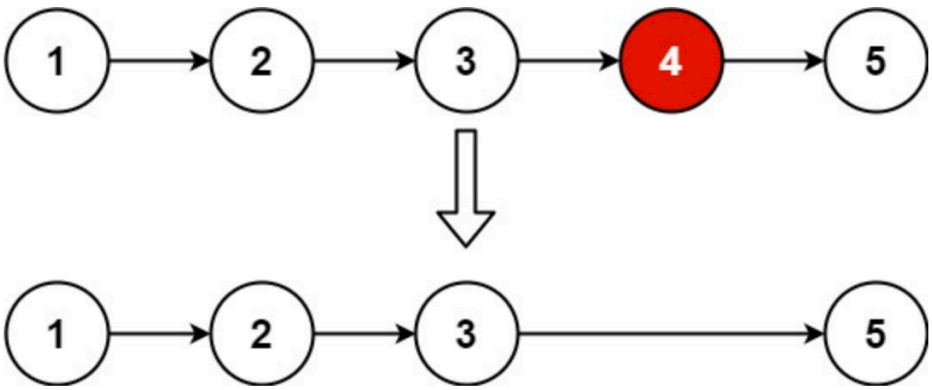
```python
class Solution:
    def swapPairs(self, head: Optional[ListNode]) -> Optional[ListNode]:
        ## current 指针法
        dummyhead = ListNode(val=0, next=head)
        cur = dummyhead

        # 遍历的终止条件:
        # 如果链表节点是奇数，则最后一个不做交换，即：cur.next.next = None 遍历结束
        # 如果链表节点是偶数，则最后一个做交换，即：cur.next = None 遍历结束 （也包括链表size = 0，空链表）
        while cur.next and cur.next.next:
            # 先检查cur.next再检查cur.next.next
            # 保存
            temp1 = cur.next
            temp2 = cur.next.next.next
            # 交换
            cur.next = cur.next.next    #(a)
            cur.next.next = temp1       #(b)
            cur.next.next.next = temp2  #(c)
            # 移动指针
            cur = cur.next.next

        # return 新链表的头节点
        return dummyhead.next
```

19. Remove Nth Node From End of List

中等 相关标签 相关企业 提示 文A

Given the `head` of a linked list, remove the `n^th` node from the end of the list and return its head.

Example 1:



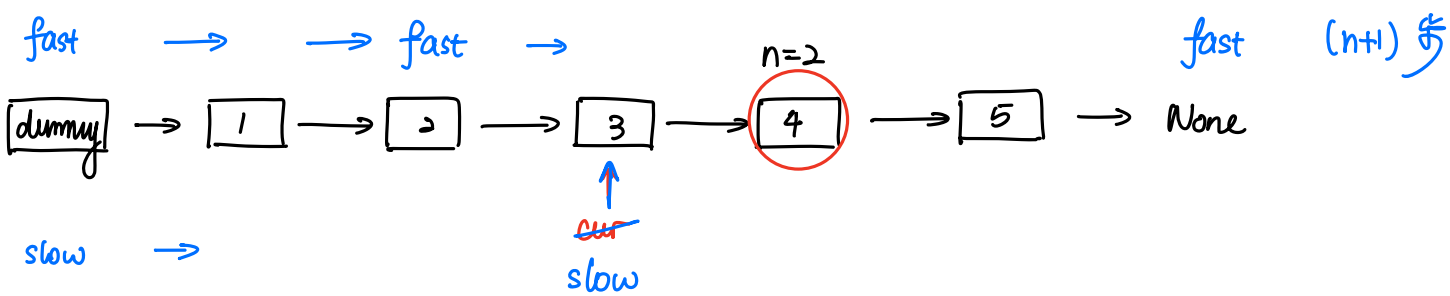

```
Input: head = [1,2,3,4,5], n = 2
Output: [1,2,3,5]
```

Example 2:

```
Input: head = [1], n = 1
Output: []
```

Comment: 删除操作已讲过，本题关键是如何找到倒数第 n 个节点！

- 思路



核心1：之前反复提过，操作指针要在被操作节点的上一个节点！
(因为是 singly linked list)

核心2：如何找到倒数第 n 个节点？
→ fast, slow 双指针

- fast 先向前走 n 步 (n+1) → 保证 slow 在被操作节点的上一个节点
- 然后 fast，slow 同时往前走，直到 fast 指向 None 为止

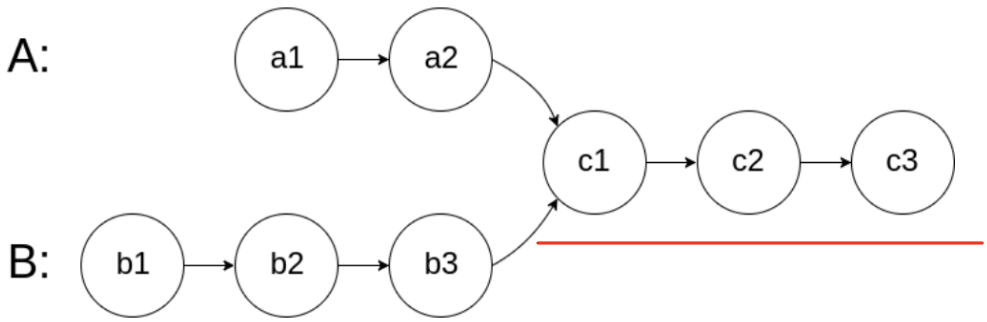
```python
class Solution:
    def removeNthFromEnd(self, head: Optional[ListNode], n: int) -> Optional[ListNode]:
        # 创建一个虚拟节点，并将其下一个指针设置为链表的head
        dummyhead = ListNode(val = 0, next = head)

        # 创建两个指针，慢指针和快指针，并将它们初始化为虚拟节点
        slow = fast = dummyhead

        # fast指针比slow指针快 n+1 步
        for i in range(n + 1):
            fast = fast.next

        # 移动两个指针，直到快速指针到达链表的末尾
        while fast:
            fast = fast.next
            slow = slow.next

        # 现在 slow 指向被操作节点的上一个节点
        # 执行删除操作
        slow.next = slow.next.next

        # 返回新链表的头节点
        return dummyhead.next
```

160. Intersection of Two Linked Lists

简单 相关标签 相关企业

Given the heads of two singly linked-lists `headA` and `headB`, return *the node at which the two lists intersect*. If the two linked lists have no intersection at all, return `null`.

For example, the following two linked lists begin to intersect at node `c1`:

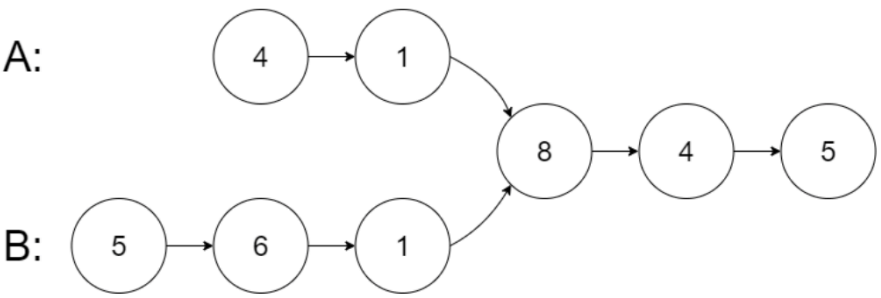


是后面的整个链表都一样

The test cases are generated such that there are no cycles anywhere in the entire linked structure.

Note that the linked lists must **retain their original structure** after the function returns.

Example 1:

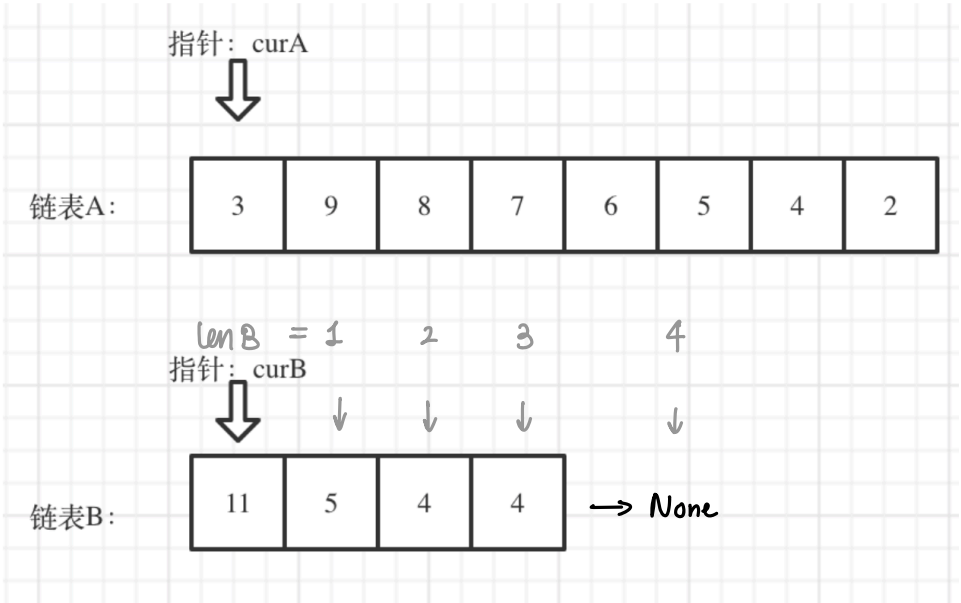



```
Input: intersectVal = 8, listA = [4,1,8,4,5], listB = [5,6,1,8,4,5], skipA = 2, skipB = 3
Output: Intersected at '8'
Explanation: The intersected node's value is 8 (note that this must not be 0 if the two
lists intersect).
From the head of A, it reads as [4,1,8,4,5]. From the head of B, it reads as
[5,6,1,8,4,5]. There are 2 nodes before the intersected node in A; There are 3 nodes
before the intersected node in B.
- Note that the intersected node's value is not 1 because the nodes with value 1 in A and
B (2nd node in A and 3rd node in B) are different node references. In other words, they
point to two different locations in memory, while the nodes with value 8 in A and B (3rd
node in A and 4th node in B) point to the same location in memory.
```

但我们不用 worry about this

简单来说，就是求两个链表相交节点的指针（address）。注意：交点不是 value 相等，而是 address 相等。

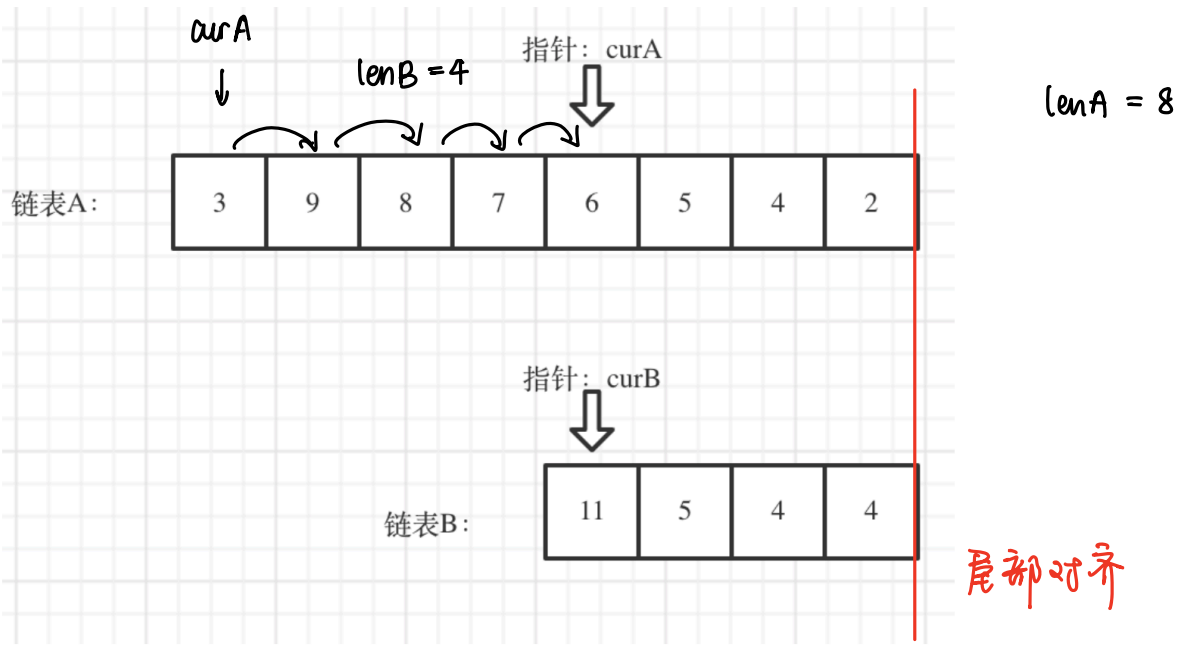
看如下两个链表，目前curA指向链表A的头结点，curB指向链表B的头结点：



思路：

step 1 我们求出两个链表的长度，并求出两个链表长度的差值，然后让curA移动到，和curB 末尾对齐的位置，如图：

step 2：



step 3：此时我们就可以比较curA和curB是否相同，如果不相同，同时向后移动curA和curB，如果遇到curA == curB，则找到交点。

否则循环退出返回空指针。

```python
class Solution:
    def getIntersectionNode(self, headA: ListNode, headB: ListNode)

        lenA, lenB = 0, 0
        # 求链表A的长度
        cur = headA
        while cur:
            cur = cur.next
            lenA += 1
        # 求链表B的长度
        cur = headB
        while cur:
            cur = cur.next
            lenB += 1

        ### curA, curB 指针
        curA, curB = headA, headB
        if lenA < lenB:    # 让curA为最长链表的头，lenA为其长度
            curA, curB = curB, curA
            lenA, lenB = lenB, lenA

        # 让curA指针向前移动lenB steps，使得两个链表尾部对齐
        for _ in range(lenA - lenB):
            curA = curA.next

        # 遍历curA 和 curB，遇到相同则直接返回
        while curA and curB:
            if curA == curB:
                return curA
            else:
                curA = curA.next
                curB = curB.next

        # 如果没找到相同，返回None
        return None
```

Step 1:

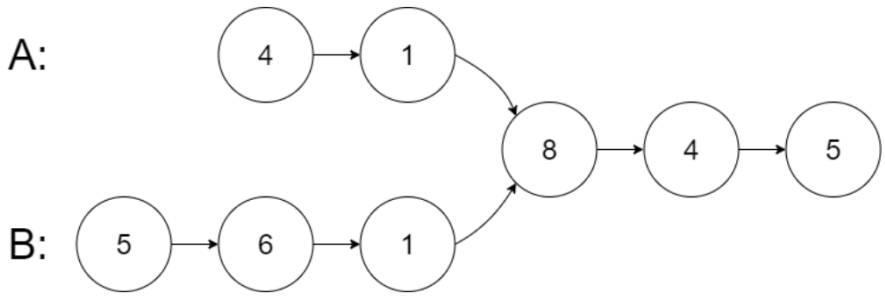
Step 2

Step 3

注：要搞清楚什么叫 curA == curB，不是 value 相等，而是由 curA，curB 开头的后面的整个链表都相同！

比如：

Example 1:



If I print them out，能看出 curA，curB 代表它们以及它们之后的整个链表.

```
27
28          # 让curA指针向前移动lenB steps，使得两个链表尾部对齐
29          for _ in range(lenA - lenB):
30              curA = curA.next
31          print(curA)
32
33          # 遍历curA 和 curB，遇到相同则直接返回
34          while curA and curB:
35              if curA == curB:
36                  return curA
37              else:
38                  curA = curA.next
39                  curB = curB.next
40                  print(curA)
41                  print(curB)
42
```

已从云端恢复　　行 30，列

测试用例 | >_ 测试结果

3

标准输出

curA → ListNode{val: 6, next: ListNode{val: 1, next: ListNode{val: 8, next: ListNode{val: 4, next: ListNode{val: 5, next: None}}}}}
curA → ListNode{val: 1, next: ListNode{val: 8, next: ListNode{val: 4, next: ListNode{val: 5, next: None}}}}
curB → ListNode{val: 1, next: ListNode{val: 8, next: ListNode{val: 4, next: ListNode{val: 5, next: None}}}}
curA → ListNode{val: 8, next: ListNode{val: 4, next: ListNode{val: 5, next: None}}}
curB → ListNode{val: 8, next: ListNode{val: 4, next: ListNode{val: 5, next: None}}}

输出

142. Linked List Cycle II

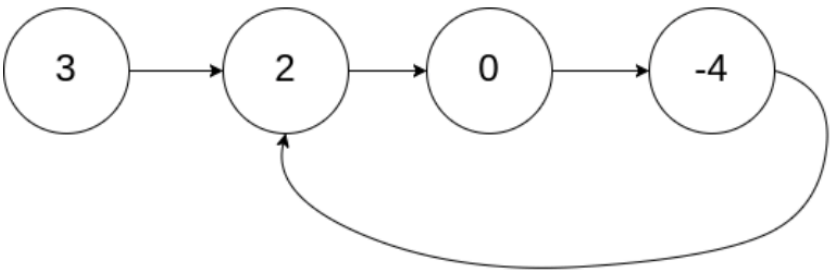
中等 相关标签 相关企业

Given the `head` of a linked list, return *the node where the cycle begins. If there is no cycle, return* `null`.

There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the `next` pointer. Internally, `pos` is used to denote the index of the node that tail's `next` pointer is connected to (**0-indexed**). It is `-1` if there is no cycle. **Note that** `pos` **is not passed as a parameter**.

Do not modify the linked list.

Example 1:




```
Input: head = [3,2,0,-4], pos = 1
Output: tail connects to node index 1
Explanation: There is a cycle in the linked list, where tail connects to the second node.
```