力扣刷题讲义2-链表

链表特点

特点分析应该对比数组

1. 链表插入删除快；而且PositionalList可以提供位置信息

插入删除分别是\_delete\_node, \_insert\_between

1. 为操作提供最坏情况的时间界限（而动态数组的摊销是实时系统操作不可接受的）

数组

1. 访问快速，但是插入删除时间复杂度不高效（要循环替换后续所有元素）
2. 相对于链式结构存储更小

双向链表实现的双端队列

\_insert\_between

\_delete\_node

基于上述功能，可以实现，\_delete\_first, \_delete\_last, \_insert\_first, \_insert\_last

Positional list:

Element()

First()

Last()

Add\_first, add\_last,

Add\_before(p,e) , add\_after(p,e)

Delete(p)

Iter(L)

After(p)

Before(p)

Replace(p,e)

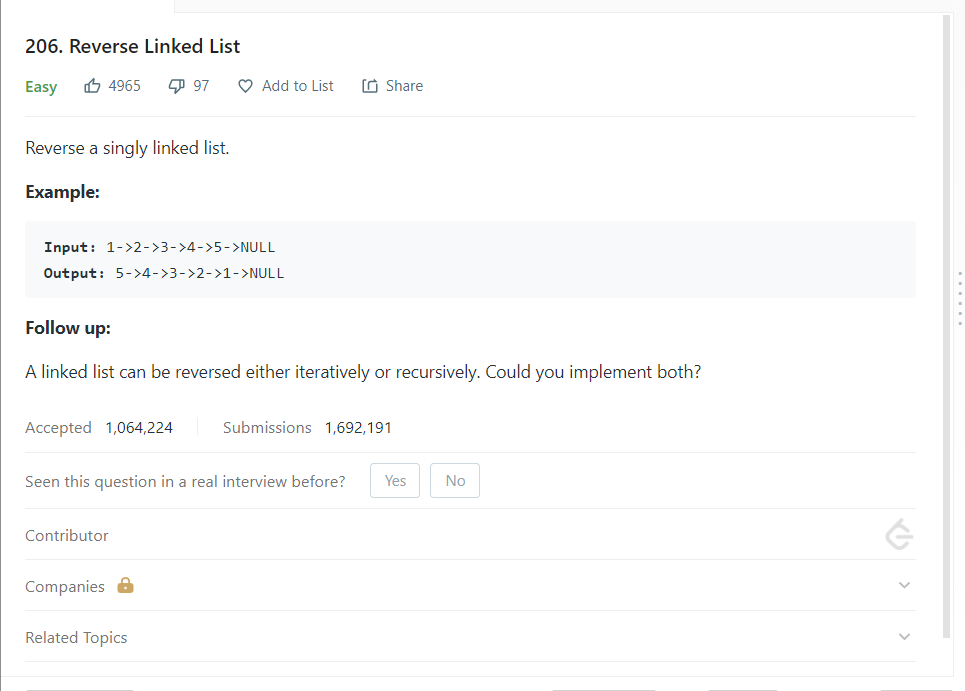
题型一：

关键词：设置新节点，head的使用

游标的移动，手术怎么做

**1.**

**Reverse linked list**



解析：一直感觉链表是一种形象的结构，需要画图去理解，感觉像做医生，通过一些基本技术，结合思考，去接节点。

这题比较简单，先尝试了一下盲打代码

我发现自己失败了，原因很简单，因为已经不记得基本的思考

以下是思考的总结，很简单的思维

#curr紧跟着head去移动，而head也是在不停移动

#curr总是往回指，指完之后很重要，prev要变为curr

那么代码就是

Def reverseLinkedList():  
prev=None

While head:

Curr=head

Head=head.next

Curr.next=prev

Prev=curr

Return prev

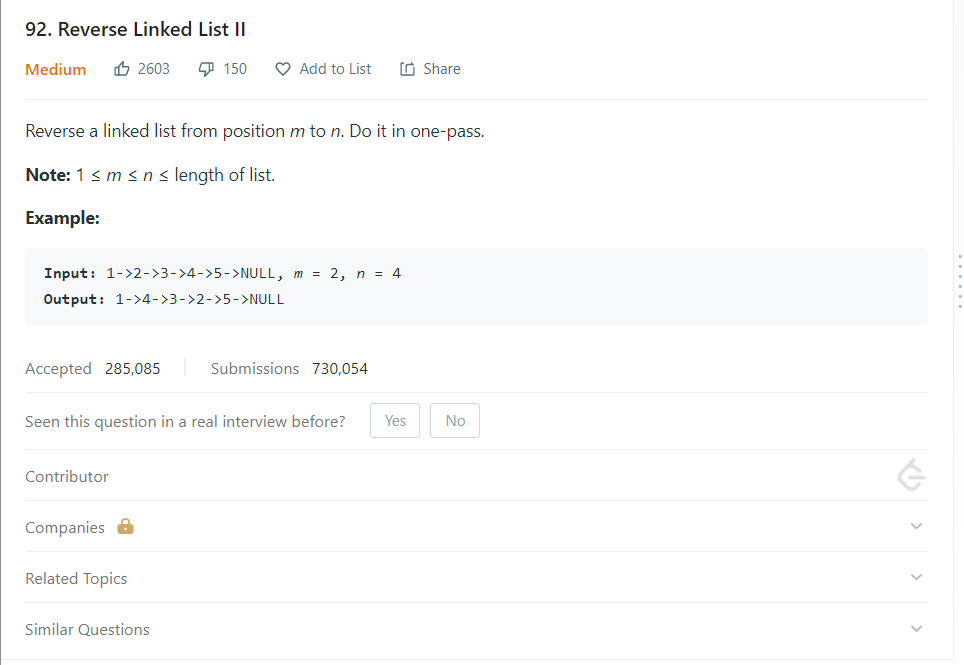
如果知道curr是紧跟head，head老大不停往下走，curr往回指，小弟又紧跟着，就不会有问题，那么思维是很简单的。

所谓的链表逆置，不用创建一个新节点，只要拿着节点head，去搞出来curr，prev，head=head.next往下动就可以了。

要素：1.prev，curr

1. head=head.next

**2.进阶版reverse linked list II**



if m==n:

return head

dummyNode=ListNode(0)

dummyNode.next=head

pre=dummyNode

for i in range(m-1):

pre=pre.next

reverse=None

cur=pre.next

for i in range(n-m+1):

next=cur.next

cur.next=reverse

reverse=cur

cur=next

#这里用的就不是head了，因为这次不能简单地用head了（定位到某一段才能开始操作）

#通过保存next=cur.next,cur=next,来进行移动

#不过reverse依旧是prev的作用，小弟跟班效果

#之所以是n-m+1,是为了最后一次弄到片段的后一个节点，便于之后的手术

pre.next.next=cur

pre.next=reverse

#（这个时候pre与它们的关系还没改变，在做最后的手术）

#这两行代码就是为了进行最后的连接

#很精妙，这里跟reverse linked list相比，逆置的过程相似，差别在于“移动”方式，而精妙之处在于最后的手术

return dummyNode.next

对比与精华：

所以这两题，主要相似之处在于

第一题里面

Prev=None

While head:

Curr=head

Head=head.next

Curr.next=prev

Prev=curr

Return prev

第二题

Prev=None

Cur=pre.next

For i in range(n-m+1):

Next=cur.next

Cur.next=prev

Prev=cur

Cur=next(

#cur.next貌似是变掉了其实已经

Pre.next.next=cur

Pre.next=prev

Return dummyNode.next

相似之处在于移动，而最后的不同在于，第一题可以直接用head

而第二题还是用next去保存好

而且第二题还有一个最后的手术步骤

**Head在第二题也不是不用，只不过是先用掉了**，创建新节点时，把新节点指向head，就可以进行进一步的设置pre和cur

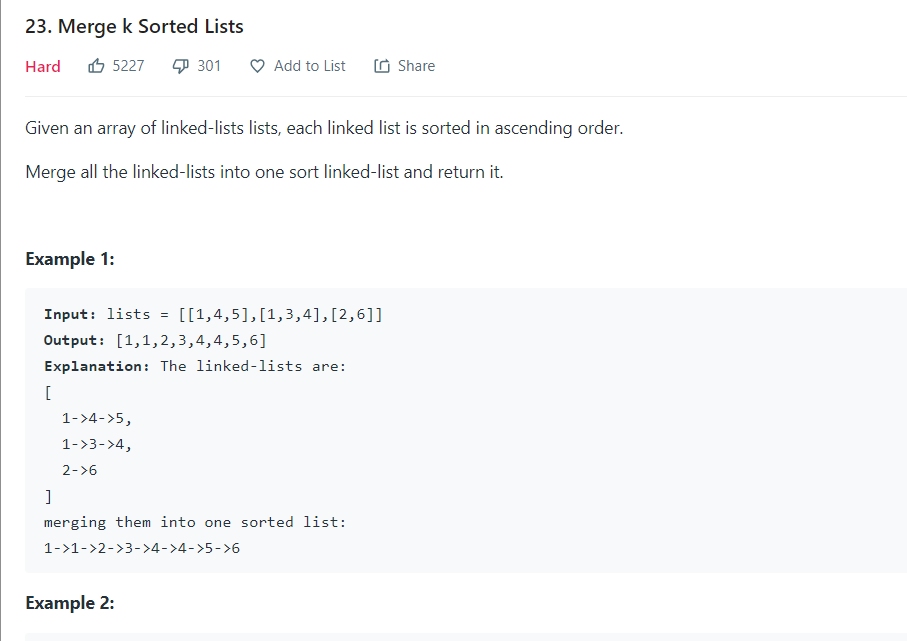
后续复习感想

9.2 reverse linked list II(对于这题而言，思路依旧记得，依旧顺利的打完了代码，记得技巧，手上也还熟悉，但是有一点就是，dummyNode是和pre是不一样的）

思路顺一遍：先建立新的节点dummyNode，然后把pre沿着新节点，新节点接好head，然后pre向下遍历设置好prev，cur

接着就开始逆置，依旧使用for循环，进行逆置，最后一次循环设置好了cur和prev，便于之后最后的一个小手术，prev是后来新设置的，用来做小跟班做逆置，而pre是之前最初的存储的位置定位，当初的pre.next接上最后的结尾，而pre用来接后来设置好的prev

**3.归并排序链表题**



思考：觉得应该，这个点在于，和数据结构starting point不一样

目前想法，把数据存储到一个地方，排序，然后再组合起来

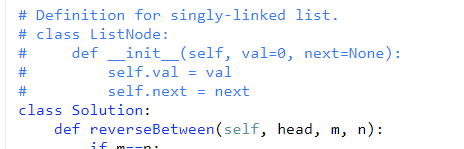
遇到error:listnode has no len

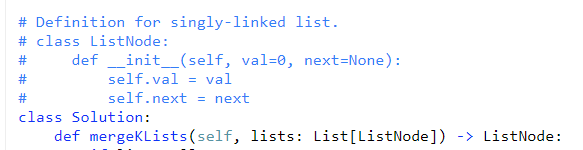
我想是我没掌握leetcode的格式书写

可以回到之前一个题目看一眼，模仿一下

Head怎么去接呢？而且这么多个head，该怎么去弄呢

不行了，感觉什么也不是很会，放弃了





个人突然觉得，其实应该根据leetcode的提示去做题，这里的self，lists，很值得注意，之前那题其实也还没注意已经给出的head，m，n

突然觉得好像装东西的这个大的lists是可以遍历去对每个进行操作的，但是没有head去管啊，而且如果去操作，接下来怎么用val去弄呢。

又觉得要看答案了

9.4 灵感：归并排序，merge sort可以复习一下其实

代码：

def mergeKLists(self,lists):

if not lists:

return

if len(lists)==1:

return lists[0]

mid=len(lists)//2

l=self.mergeKLists(lists[mid:])

r=self.mergeKLists(lists[:mid])

return self.merge(l,r)

def merge(self,l,r):

dummy=cur=ListNode(0)

while l and r:

if l.val<r.val:

cur.next=l

l=l.next

else:

cur.next=r

r=r.next

cur=cur.next

cur.next=l or r

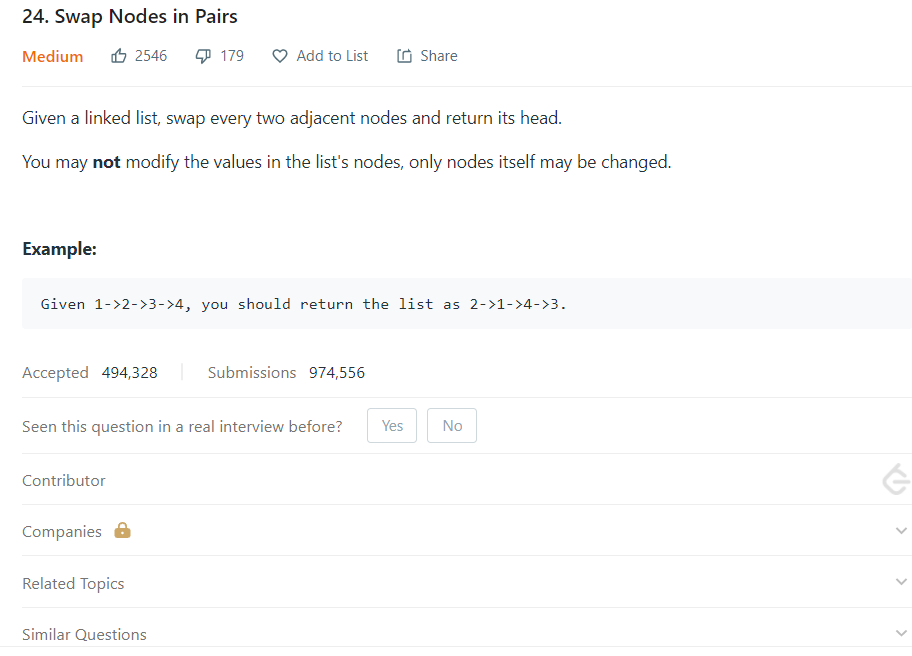
return dummy.next

**解析：**

#这题我们其实是通过递归获取了类似于head这种东西

#链表题：找head，找工具人dummy，cur,

**4.swapPairs利用节点**



def swapPairs(self,head):

If not head or not head.next: return head

dummy=ListNode(0)

dummy.next=head

cur=dummy

while cur.next and cur.next.next:

first=cur.next

sec=cur.next.next

cur.next=sec

first.next=sec.next

sec.next=first

cur=cur.next.next

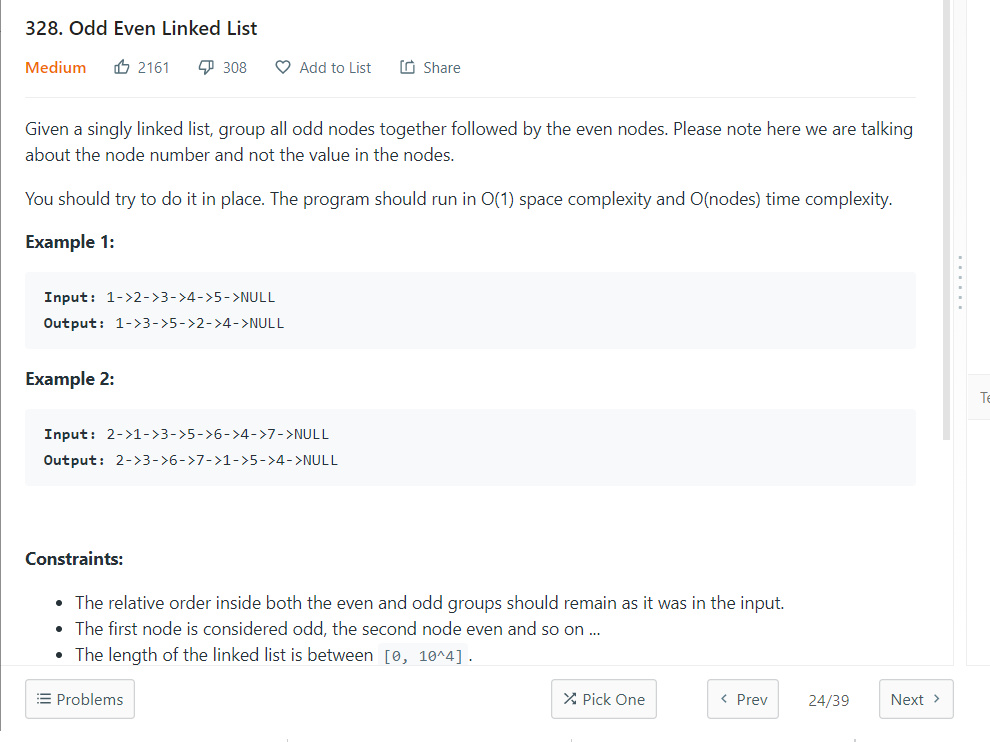
return dummy.next

**解析**：最主要的，这个swapPairs的题其实还是节点的一个手术，关键在于什么呢，一个是这个while循环整个架构要想好怎么去构造，一个是要把这个头节点的移动搞清楚，比如那行代码cur=cur.next.next

和人交流感想：主要是，head不动，每次总有一个新的节点先跟head产生关系，然后作为工具人一直被使用

9.15坐在校车上的体会：穿针引线，就可以做好这道题，穿针引线，按照线的路线一路往下移动head和prevnode

**5.odd even linked list**



思考：一开始尝试了一些利用节点的死做方法，已经很接近了，但是最后还是没接清楚，主要是基于swapPairs的一些技巧，但是可能一些地方不对，导致了error-cycle found in listnode这样的错误