Leetcode(python版) 链表中等题

2两数相加

题目：给定两个**非空**链表来表示两个非负整数。位数按照**逆序**方式存储，它们的每个节点只存储单个数字。将两数相加返回一个新的链表。

你可以假设除了数字 0 之外，这两个数字都不会以零开头。

**示例：**

**输入：**(2 -> 4 -> 3) + (5 -> 6 -> 4)

**输出：**7 -> 0 -> 8

**原因：**342 + 465 = 807

思路：创建一个新的链表，两数想加，如果大于9产生进位，直到一个数结束，然后把另一个数的尾部直接加入新链表中

代码：

1. # Definition for singly-linked list.
2. # class ListNode(object):
3. #     def \_\_init\_\_(self, x):
4. #         self.val = x
5. #         self.next = None
7. **class** Solution(object):
8. **def** addTwoNumbers(self, l1, l2):
9. """
10. :type l1: ListNode
11. :type l2: ListNode
12. :rtype: ListNode
13. """
14. p=ListNode(None)
15. carry=0
16. res=p
17. **while** l1 **or** l2 **or** carry:
18. **if** l1:
19. carry+=l1.val
20. l1=l1.next
21. **if** l2:
22. carry+=l2.val
23. l2=l2.next
24. carry,val=divmod(carry,10)
25. p.next=ListNode(val)
26. p=p.next
27. **return** res.next

19删除链表的倒数第N个结点

题目：给定一个链表，删除链表的倒数第 *n*个节点，并且返回链表的头结点。

示例

给定一个链表: 1->2->3->4->5, 和 n = 2.

当删除了倒数第二个节点后，链表变为 1->2->3->5.

进阶：

你能尝试使用一趟扫描实现吗？

思路：快慢指针，快指针先走n步，慢指针再开始走，当快指针走到最后一个结点，慢指针就在倒数第n+1个结点了，

代码：

1. # Definition for singly-linked list.
2. # class ListNode(object):
3. #     def \_\_init\_\_(self, x):
4. #         self.val = x
5. #         self.next = None
7. **class** Solution(object):
8. **def** removeNthFromEnd(self, head, n):
9. """
10. :type head: ListNode
11. :type n: int
12. :rtype: ListNode
13. """
14. dummy=ListNode(None)
15. dummy.next=head
16. p1=p2=dummy
17. **for** i **in** range(n):
18. p1=p1.next
19. **while** p1.next:
20. p1=p1.next
21. p2=p2.next
22. p2.next=p2.next.next
23. **return** dummy.next

24两两交换链表中的节点

题目：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

**示例:**

给定 1->2->3->4, 你应该返回 2->1->4->3.

**说明:**

* 你的算法只能使用常数的额外空间。
* **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

思路：只要找个四个节点交换之间的关系就可以了

代码：

1. # Definition for singly-linked list.
2. # class ListNode(object):
3. #     def \_\_init\_\_(self, x):
4. #         self.val = x
5. #         self.next = None
7. **class** Solution(object):
8. **def** swapPairs(self, head):
9. """
10. :type head: ListNode
11. :rtype: ListNode
12. """
13. **if** **not** head:
14. **return** None
15. cur=ListNode(None)
16. cur.next=head
17. first=cur
18. **while** cur.next **and** cur.next.next:
19. n1=cur.next
20. n2=n1.next
21. nxt=n2.next
22. n1.next=nxt
23. n2.next=n1
24. cur.next=n2
25. cur=n1 //画图可知，此时n1已经是第二值，这样cur.next就是下一组节点
26. **return** first.next

61旋转链表

题目：给定一个链表，旋转链表，将链表每个节点向右移动 *k*个位置，其中 *k*是非负数。

**示例 1:**

**输入:** 1->2->3->4->5->NULL, k = 2

**输出:** 4->5->1->2->3->NULL

**解释:**

向右旋转 1 步: 5->1->2->3->4->NULL

向右旋转 2 步: 4->5->1->2->3->NULL

**示例 2:**

**输入:** 0->1->2->NULL, k = 4

**输出:** 2->0->1->NULL

**解释:**

向右旋转 1 步: 2->0->1->NULL

向右旋转 2 步: 1->2->0->NULL

向右旋转 3 步: 0->1->2->NULL

向右旋转 4 步: 2->0->1->NULL

思路：使用快慢指针，但是在之前要先考虑k的值，从而减少不必要的操作

代码：

1. # Definition for singly-linked list.
2. # class ListNode(object):
3. #     def \_\_init\_\_(self, x):
4. #         self.val = x
5. #         self.next = None
7. **class** Solution(object):
8. **def** rotateRight(self, head, k):
9. """
10. :type head: ListNode
11. :type k: int
12. :rtype: ListNode
13. """
14. **if** **not** head **or** head.next==None **or** k==0:
15. **return** head
16. ListLen = 0
17. p=head
18. **while**(p):
19. ListLen+=1
20. p = p.next
21. k = k%ListLen
22. **if** k==0:
23. **return** head
25. cur=ListNode(None)
26. cur.next=head
27. slow=cur
28. fast=cur
29. **for** i **in** range(k):
30. fast=fast.next
31. **while** fast.next:
32. fast=fast.next
33. slow=slow.next
34. fast.next=head
35. newhead=slow.next
36. slow.next=None
37. **return** newhead

82删除排序链表中的重复元素II

题目：给定一个排序链表，删除所有含有重复数字的节点，只保留原始链表中 *没有重复出现*的数字。

**示例 1:**

**输入:** 1->2->3->3->4->4->5

**输出:** 1->2->5

**示例 2:**

**输入:** 1->1->1->2->3

**输出:** 2->3

思路：保存两次头节点

我们可以很快速的解决这个问题。

h -> 1 -> 2 -> 3 -> 3 -> 4 -> 4 -> 5

pre cur

我们始终要让pre在cur的前面，通过判断cur.val == cur.next.val判断重复元素是否存在。

h -> 1 -> 2 3 3 4 -> 4 -> 5

| |

-----------

pre cur

如果存在重复元素的话，我们pre.next=cur.next;cur=cur.next，如果不存在重复元素，我们pre=cur;cur=cur.next，所以这里我们要通过一个变量标记是否存在重复元素。最后代码如下

！

代码：

1. # Definition for singly-linked list.
2. # class ListNode(object):
3. #     def \_\_init\_\_(self, x):
4. #         self.val = x
5. #         self.next = None
7. **class** Solution(object):
8. **def** deleteDuplicates(self, head):
9. """
10. :type head: ListNode
11. :rtype: ListNode
12. """
13. **if** **not** head **or** head.next==None:
14. **return** head
15. p=ListNode(None)
16. p.next=head
17. pre=p
18. cur=head
19. **while** cur:
20. duplication=False
21. **while** cur.next **and** cur.next.val==cur.val:
22. cur=cur.next
23. duplication=True
24. **if** duplication==False:
25. pre=cur
26. **else**:
27. pre.next=cur.next
28. cur=cur.next
29. **return** p.next

86分隔链表

题目：给定一个链表和一个特定值 *x*，对链表进行分隔，使得所有小于 *x* 的节点都在大于或等于 *x* 的节点之前。

你应当保留两个分区中每个节点的初始相对位置。

**示例:**

**输入:** head = 1->4->3->2->5->2, *x* = 3

**输出:** 1->2->2->4->3->5

思路：构建两个新链表，小于x的放在链表1，大于或等于x的放在链表2，最后将链表1的表尾指向链表2的表头即可

代码：

1. # Definition for singly-linked list.
2. # class ListNode(object):
3. #     def \_\_init\_\_(self, x):
4. #         self.val = x
5. #         self.next = None
7. **class** Solution(object):
8. **def** partition(self, head, x):
9. """
10. :type head: ListNode
11. :type x: int
12. :rtype: ListNode
13. """
14. h1=ListNode(None)
15. p1=h1
16. h2=ListNode(None)
17. p2=h2
18. **if** **not** head **or** head.next==None:
19. **return** head
20. p=head
21. **while** p:
22. **if** p.val<x:
23. p1.next=p
24. p=p.next
25. p1=p1.next
26. p1.next=None
27. **else**:
28. p2.next=p
29. p=p.next
30. p2=p2.next
31. p2.next=None
32. p1.next=h2.next
33. h=h1.next
34. **return** h

92反转链表II

题目：反转从位置 *m* 到 *n* 的链表。请使用一趟扫描完成反转。

思路：首先把所有链表压入一个列表里，然后反转相应的位置。创建新的链表

代码：

1. # Definition for singly-linked list.
2. # class ListNode(object):
3. #     def \_\_init\_\_(self, x):
4. #         self.val = x
5. #         self.next = None
7. **class** Solution(object):
8. **def** reverseBetween(self, head, m, n):
9. """
10. :type head: ListNode
11. :type m: int
12. :type n: int
13. :rtype: ListNode
14. """
15. **if** **not** head:
16. **return**
17. **if** m==n:
18. **return** head
19. stack=[]
20. first=ListNode(None)
21. **while** head:
22. stack.append(head.val)
23. head=head.next
24. stack[m-1:n]=reversed(stack[m-1:n])
25. res=first
26. **while** stack:
27. first.next=ListNode(stack.pop(0))
28. first=first.next
29. **return** res.next

109有序链表转换二叉搜索树

题目：给定一个单链表，其中的元素按升序排序，将其转换为高度平衡的二叉搜索树。

本题中，一个高度平衡二叉树是指一个二叉树*每个节点*的左右两个子树的高度差的绝对值不超过 1。

思路：利用快慢指针找到中点后，要以中点的值建立一个数的根节点，然后需要把原链表断开，分为前后两个链表，都不能包含原中节点，然后再分别对这两个链表递归调用原函数，分别连上左右子节点即可。

代码：

1. # Definition for singly-linked list.
2. # class ListNode(object):
3. #     def \_\_init\_\_(self, x):
4. #         self.val = x
5. #         self.next = None
7. # Definition for a binary tree node.
8. # class TreeNode(object):
9. #     def \_\_init\_\_(self, x):
10. #         self.val = x
11. #         self.left = None
12. #         self.right = None
14. **class** Solution(object):
15. **def** sortedListToBST(self, head):
16. """
17. :type head: ListNode
18. :rtype: TreeNode
19. """
20. **if** **not** head:
21. **return**
22. **if** **not** head.next:
23. **return** TreeNode(head.val)
25. slow, fast = head, head.next.next
26. **while** fast **and** fast.next:
27. fast = fast.next.next
28. slow = slow.next
30. tmp = slow.next
31. slow.next = None
32. root = TreeNode(tmp.val)
33. root.left = self.sortedListToBST(head)
34. root.right = self.sortedListToBST(tmp.next)
35. **return** root

138复制带随机指针的链表

题目：给定一个链表，每个节点包含一个额外增加的随机指针，该指针可以指向链表中的任何节点或空节点。

要求返回这个链表的深度拷贝

思路：*tep1:复制节点，将其插入到原节点的后面//step2:复制random指针//step3:将一个链表拆成两个链表*

代码：

1. # Definition for singly-linked list with a random pointer.
2. # class RandomListNode(object):
3. #     def \_\_init\_\_(self, x):
4. #         self.label = x
5. #         self.next = None
6. #         self.random = None
8. **class** Solution(object):
9. **def** copyRandomList(self, head):
10. """
11. :type head: RandomListNode
12. :rtype: RandomListNode
13. """
14. newhead=RandomListNode(0)
15. l1=head
16. **while** l1:
17. l2=RandomListNode(0)
18. l2.next=l1.next
19. l1.next=l2
20. l1=l2.next
22. newhead.next=head
23. l1=head
24. **while** l1:
25. **if** l1.random:
26. l1.next.random=l1.random.next
27. l1=l1.next.next
28. l1=head
29. **while** l1:
30. l2=l1.next
31. l1.next=l2.next
32. **if** l2.next:
33. l2.next=l2.next.next
34. l1=l1.next
35. **return** newhead.next

142环形链表II

题目：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，则返回 null。

**说明：**不允许修改给定的链表

思路：使用快慢指针，判断是否有环，再让一个指针从原点处发，慢指针原地走，相遇时的那个节点则是入环的第一个节点

代码：

1. # Definition for singly-linked list.
2. # class ListNode(object):
3. #     def \_\_init\_\_(self, x):
4. #         self.val = x
5. #         self.next = None
7. **class** Solution(object):
8. **def** detectCycle(self, head):
9. """
10. :type head: ListNode
11. :rtype: ListNode
12. """
13. **if** **not** head **or** head.next==None:
14. **return** None
15. slow=fast=head
16. **while** fast.next **and** fast.next.next:
17. slow=slow.next
18. fast=fast.next.next
19. **if** slow==fast:
20. p = head
21. **while** slow != p:
22. p = p.next
23. slow = slow.next
24. **return** p
25. **return** None

143重排链表

题目：给定一个单链表 *L*：*L*0→*L*1→…→*Ln*-1→*L*n ，  
将其重新排列后变为： *L*0→*Ln*→*L*1→*Ln*-1→*L*2→*Ln*-2→…

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

给定链表 1->2->3->4, 重新排列为 1->4->2->3.

**示例 2:**

给定链表 1->2->3->4->5, 重新排列为 1->5->2->4->3.

思路：利用快慢指针找到中间节点，将链表分成两个，再将后一个链表反转，最后一一对应

代码：

1. # Definition for singly-linked list.
2. # class ListNode(object):
3. #     def \_\_init\_\_(self, x):
4. #         self.val = x
5. #         self.next = None
7. **class** Solution(object):
8. **def** reorderList(self, head):
9. """
10. :type head: ListNode
11. :rtype: void Do not return anything, modify head in-place instead.
12. """
13. **if** head == None **or** head.next == None:
14. **return**
16. pre = head
17. lat = head.next
18. **while** lat != None **and** lat.next != None:
19. pre = pre.next
20. lat = lat.next.next
22. p = pre.next
23. pre.next = None
24. # reverse
26. cur = None
27. **while** p != None:
28. q = p.next
29. p.next = cur
30. cur = p
31. p = q
33. pre = head
34. **while** pre != None **and** cur != None:
35. tmp = cur.next
36. cur.next = pre.next
37. pre.next = cur
38. pre = pre.next.next
39. cur = tmp

147对链表进行插入排序

题目：对链表进行插入排序。从第一个元素开始，该链表可以被认为已经部分排序。每次迭代时，从输入数据中移除一个元素，并原地将其插入到已排好序的链表中。

**示例 1：**

**输入:** -1->5->3->4->0

**输出:** -1->0->3->4->5

思路：设置一个新的链表。遍历旧链表，每一个节点都作为一个新节点，按序插入新链表中

代码：

1. # Definition for singly-linked list.
2. # class ListNode(object):
3. #     def \_\_init\_\_(self, x):
4. #         self.val = x
5. #         self.next = None
7. **class** Solution(object):
8. **def** insertionSortList(self, head):
9. """
10. :type head: ListNode
11. :rtype: ListNode
12. """
13. p = new\_s = ListNode(None)
14. new\_s.next = head
15. cur = head
16. **while** cur **and** cur.next:
17. val = cur.next.val
18. **if** val > cur.val:
19. cur = cur.next
20. **continue**
21. **if** p.next.val > val:
22. p = new\_s
23. **while** p.next.val < val:
24. p = p.next
25. new = cur.next
26. cur.next = new.next
27. new.next = p.next
28. p.next = new
29. **return** new\_s.next

148排序链表

题目：在 *O*(*n* log *n*) 时间复杂度和常数级空间复杂度下，对链表进行排序。

**示例 1:**

**输入:** 4->2->1->3

**输出:** 1->2->3->4

**示例 2:**

**输入:** -1->5->3->4->0

**输出:** -1->0->3->4->5

思路：这个时间复杂度让我想到归并排序，对于链表来说使用归并排序占用空间为O(1).

代码：

1. # Definition for singly-linked list.
2. # class ListNode(object):
3. #     def \_\_init\_\_(self, x):
4. #         self.val = x
5. #         self.next = None
7. **class** Solution(object):
8. **def** sortList(self, head):
9. """
10. :type head: ListNode
11. :rtype: ListNode
12. """
13. **if** **not** head **or** head.next==None:
14. **return** head
15. mid=self.get\_mid(head)
16. l=head
17. r=mid.next
18. mid.next=None
19. **return** self.merge(self.sortList(l),self.sortList(r))
20. **def** merge(self,p,q):
21. temp=ListNode(0)
22. h=temp
23. **while** p **and** q:
24. **if** p.val<q.val:
25. h.next=p
26. p=p.next
27. **else**:
28. h.next=q
29. q=q.next
30. h=h.next
31. **if** p:
32. h.next=p
33. **if** q:
34. h.next=q
35. **return** temp.next
36. **def** get\_mid(self,node):
37. **if** node **is** None:
38. **return** None
39. fast=slow=node
40. **while** fast.next **and** fast.next.next:
41. fast=fast.next.next
42. slow=slow.next
43. **return** slow

328奇偶链表

题目：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

思路：保存第二节点的地址，分成奇偶两条的链表之后，奇链表的尾部指向第二个节点，偶链表的结尾指向None

代码：

1. # Definition for singly-linked list.
2. # class ListNode(object):
3. #     def \_\_init\_\_(self, x):
4. #         self.val = x
5. #         self.next = None
7. **class** Solution(object):
8. **def** oddEvenList(self, head):
9. """
10. :type head: ListNode
11. :rtype: ListNode
12. """
13. **if** head **is** None:
14. **return** head
15. odd=oddhead=head
16. even=evenhead=head.next
17. **while** even **and** even.next:
18. odd.next=even.next
19. odd=odd.next
20. even.next=odd.next
21. even=even.next
22. odd.next=evenhead
23. **return** oddhead

430扁平化多级双向链表

题目：您将获得一个双向链表，除了下一个和前一个指针之外，它还有一个子指针，可能指向单独的双向链表。这些子列表可能有一个或多个自己的子项，依此类推，生成多级数据结构，如下面的示例所示。

扁平化列表，使所有结点出现在单级双链表中。您将获得列表第一级的头部。

**示例:**

**输入:**

1---2---3---4---5---6--NULL

|

7---8---9---10--NULL

|

11--12--NULL

**输出:**

1-2-3-7-8-11-12-9-10-4-5-6-NULL

思路：看到把子节点插入到后面，就想到了我们应该使用的是DFS，这种搜索方式会让我们提前使用更深层次的节点，当更深层次的搜索结束之后再往上层返回。

现在的思路就是每次遇到child节点，就把这个节点作为当前node的next节点；并且要遍历child节点后面的所有节点，找到child链表最后面的节点，作为要插入的一整段链表最后的节点，即原node.next节点prev节点。

做法需要新定义一个函数，这个函数对每个child链表进行遍历，把整段的child链表插入到原链表中。

思路总结就是：DFS负责查找，新定义的函数负责插入。

代码：

1. """
2. # Definition for a Node.
3. class Node(object):
4. def \_\_init\_\_(self, val, prev, next, child):
5. self.val = val
6. self.prev = prev
7. self.next = next
8. self.child = child
9. """
10. **class** Solution(object):
11. **def** flatten(self, head):
12. """
13. :type head: Node
14. :rtype: Node
15. """
16. **if** **not** head: **return** None
17. node = head
18. **while** node:
19. node\_next = node.next
20. **if** node.child:
21. flattened = self.flatten(node.child)
22. node.child = None
23. nextNode = self.appendToList(node, flattened)
24. node = nextNode
25. **else**:
26. node = node.next
27. **return** head
29. **def** appendToList(self, node, listToAppendHead):
30. next\_node = node.next
31. node.next = listToAppendHead
32. listToAppendHead.prev = node
33. **while** node.next:
34. node = node.next
35. node.next = next\_node
36. **if** next\_node:
37. next\_node.prev = node
38. **return** next\_node

445两数相加II

题目：给定两个**非空**链表来代表两个非负整数。数字最高位位于链表开始位置。它们的每个节点只存储单个数字。将这两数相加会返回一个新的链表。

你可以假设除了数字 0 之外，这两个数字都不会以零开头。

**进阶:**

如果输入链表不能修改该如何处理？换句话说，你不能对列表中的节点进行翻转。

**示例:**

**输入:** (7 -> 2 -> 4 -> 3) + (5 -> 6 -> 4)

**输出:** 7 -> 8 -> 0 -> 7

思路：可以使用一个栈来完成操作，那么加的过程中采用头插法，一直向头部插入新的节点就好了。

代码：

1. # Definition for singly-linked list.
2. # class ListNode(object):
3. #     def \_\_init\_\_(self, x):
4. #         self.val = x
5. #         self.next = None
7. **class** Solution(object):
8. **def** addTwoNumbers(self, l1, l2):
9. """
10. :type l1: ListNode
11. :type l2: ListNode
12. :rtype: ListNode
13. """
14. stack1=[]
15. stack2=[]
16. **while** l1:
17. stack1.append(l1.val)
18. l1=l1.next
19. **while** l2:
20. stack2.append(l2.val)
21. l2=l2.next
22. answer=None
23. carry=0
24. **while** stack1 **and** stack2:
25. add= stack1.pop()+stack2.pop()+carry
26. carry=1 **if** add>=10 **else** 0
27. temp=answer
28. answer=ListNode(add%10)
29. answer.next=temp
30. l=stack1 **if** stack1 **else** stack2
31. **while** l:
32. add=l.pop()+carry
33. carry=1 **if** add>=10 **else** 0
34. temp=answer
35. answer=ListNode(add%10)
36. answer.next=temp
37. **if** carry:
38. temp=answer
39. answer=ListNode(1)
40. answer.next=temp
41. **return** answer

725分割链表

题目：给定一个头结点为 root 的链表, 编写一个函数以将链表分隔为 k 个连续的部分。

每部分的长度应该尽可能的相等: 任意两部分的长度差距不能超过 1，也就是说可能有些部分为 null。

这k个部分应该按照在链表中出现的顺序进行输出，并且排在前面的部分的长度应该大于或等于后面的长度。

返回一个符合上述规则的链表的列表。

举例： 1->2->3->4, k = 5 // 5 结果 [ [1], [2], [3], [4], null ]

**示例 1：**

**输入:**

root = [1, 2, 3], k = 5

**输出:** [[1],[2],[3],[],[]]

**解释:**

输入输出各部分都应该是链表，而不是数组。

例如, 输入的结点 root 的 val= 1, root.next.val = 2, \root.next.next.val = 3, 且 root.next.next.next = null。

第一个输出 output[0] 是 output[0].val = 1, output[0].next = null。

最后一个元素 output[4] 为 null, 它代表了最后一个部分为空链表。

**示例 2：**

**输入:**

root = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], k = 3

**输出:** [[1, 2, 3, 4], [5, 6, 7], [8, 9, 10]]

**解释:**

输入被分成了几个连续的部分，并且每部分的长度相差不超过1.前面部分的长度大于等于后面部分的长度。

思路：第一次遍历单链表，求出链表的长度length，求出平均分成的k个链表中，每个的结点avg，以及还多余的结点rem，第二次遍历输入链表，如果达到avg，且rem存在值，则把本次遍历的结果赋值给结果数组；

代码：

1. # Definition for singly-linked list.
2. # class ListNode(object):
3. #     def \_\_init\_\_(self, x):
4. #         self.val = x
5. #         self.next = None
7. **class** Solution(object):
8. **def** splitListToParts(self, root, k):
9. """
10. :type root: ListNode
11. :type k: int
12. :rtype: List[ListNode]
13. """
14. ret=[None]\*k
15. length=0
16. move=root
17. **while** move:
18. length+=1
19. move=move.next
20. avg=length/k
21. rem=length%k
22. move=root
23. index=0
24. **while** move:
25. temp=move
26. pre=ListNode(0)
27. pre.next=move
28. num=0
29. **while** num<avg:
30. pre=pre.next
31. move=move.next
32. num+=1
33. **if** rem:
34. pre=pre.next
35. move=move.next
36. rem-=1
37. pre.next=None
38. ret[index]=temp
39. index+=1
40. **return** ret

817链表组件

题目：给定一个链表（链表结点包含一个整型值）的头结点 head。

同时给定列表 G，该列表是上述链表中整型值的一个子集。

返回列表 G 中组件的个数，这里对组件的定义为：链表中一段最长连续结点的值（该值必须在列表 G 中）构成的集合。

**示例 1：**

**输入:**

head: 0->1->2->3

G = [0, 1, 3]

**输出:** 2

**解释:**

链表中,0 和 1 是相连接的，且 G 中不包含 2，所以 [0, 1] 是 G 的一个组件，同理 [3] 也是一个组件，故返回 2。

**示例 2：**

**输入:**

head: 0->1->2->3->4

G = [0, 3, 1, 4]

**输出:** 2

**解释:**

链表中，0 和 1 是相连接的，3 和 4 是相连接的，所以 [0, 1] 和 [3, 4] 是两个组件，故返回 2。

**注意:**

* 如果 N 是给定链表 head 的长度，1 <= N <= 10000。
* 链表中每个结点的值所在范围为 [0, N - 1]。
* 1 <= G.length <= 10000
* G 是链表中所有结点的值的一个子集.

思路：从左到右遍历链表依次，每遇到一个节点，就看看这个节点的数值在不在G中，并且这个节点的下一个节点是不是空（末尾），下一个节点值在不在G中。

如果当前的节点值在G中，而下一个节点是空或者节点值不在G中，那么就是一个新的分段呗。

这个题第一次提交的时候超时，超时原因应该在判断一个元素在不在列表中这步比较慢。使用set之后就能提交通过了。。

代码：

1. # Definition for singly-linked list.
2. # class ListNode(object):
3. #     def \_\_init\_\_(self, x):
4. #         self.val = x
5. #         self.next = None
7. **class** Solution(object):
8. **def** numComponents(self, head, G):
9. """
10. :type head: ListNode
11. :type G: List[int]
12. :rtype: int
13. """
14. groups = 0
15. subset = set(G)
16. **while** head:
17. **if** head.val **in** subset **and** (**not** head.next **or** head.next.val **not** **in** subset):
18. groups += 1
19. head = head.next
20. **return** groups

思路2：开辟一个足够大的数组map存储G数组的值，令map[G[i]] = G[i]，每访问一个链表结点可以直接根据map中的值判断是否属于G数组，不需要反复遍历。