# 链表

## 2. 两数相加(中等)

给出两个非空的链表用来表示两个非负的整数。其中，它们各自的位数是按照逆序的方式存储的，并且它们的每个节点只能存储一位数字。

·如果，我们将这两个数相加起来，则会返回一个新的链表来表示它们的和。

·您可以假设除了数字0之外，这两个数都不会以0开头。

示例：

输入：(2 -> 4 -> 3) + (5 -> 6 -> 4)

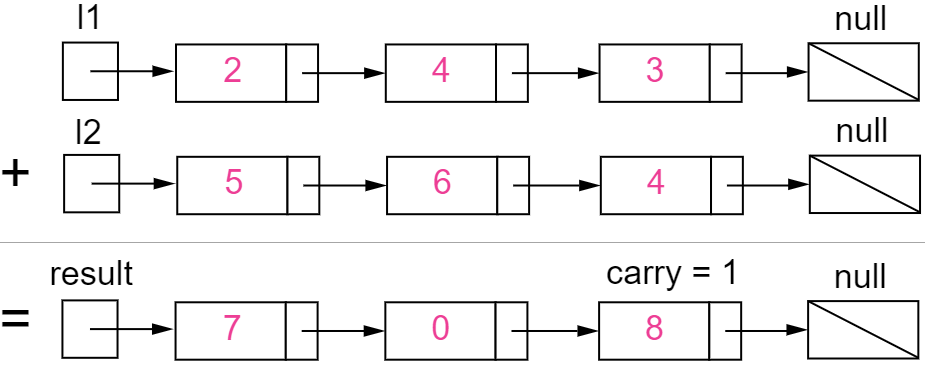
输出：7 -> 0 -> 8

原因：342 + 465 = 807

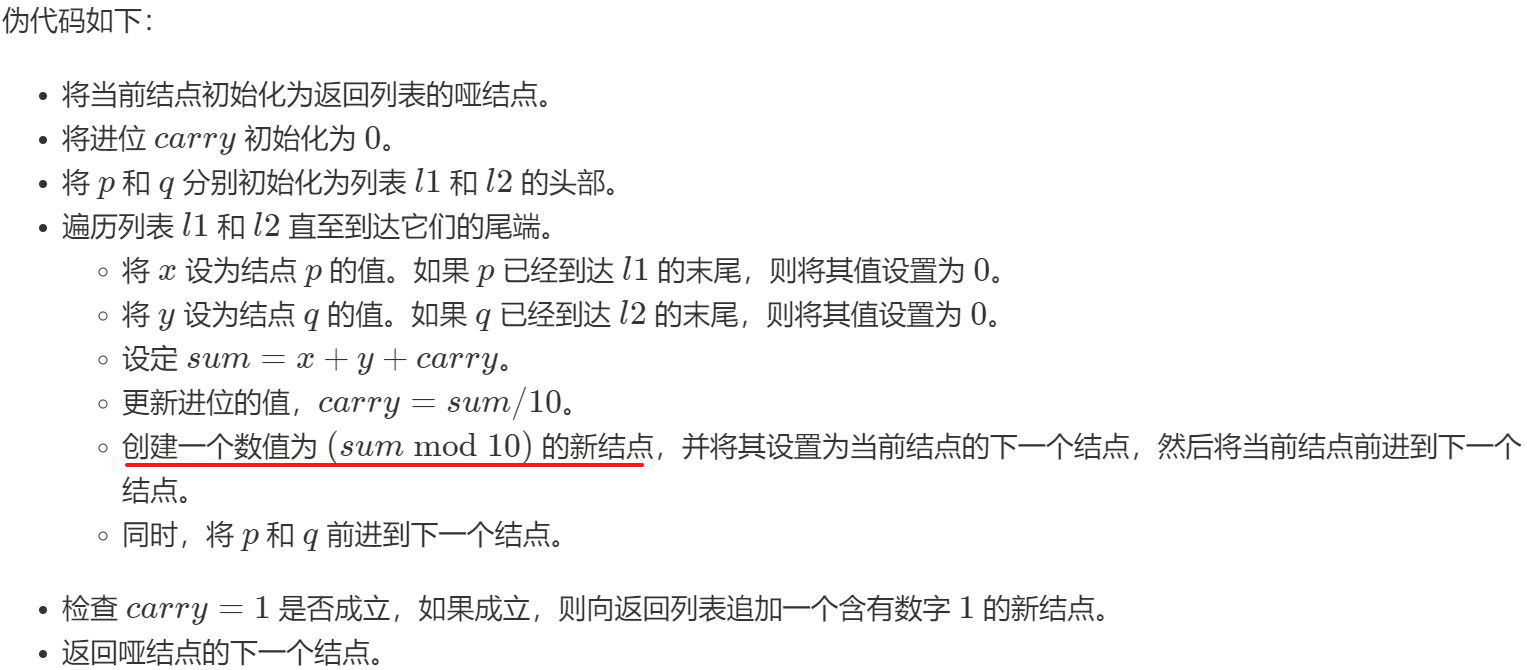
来源：力扣（LeetCode）

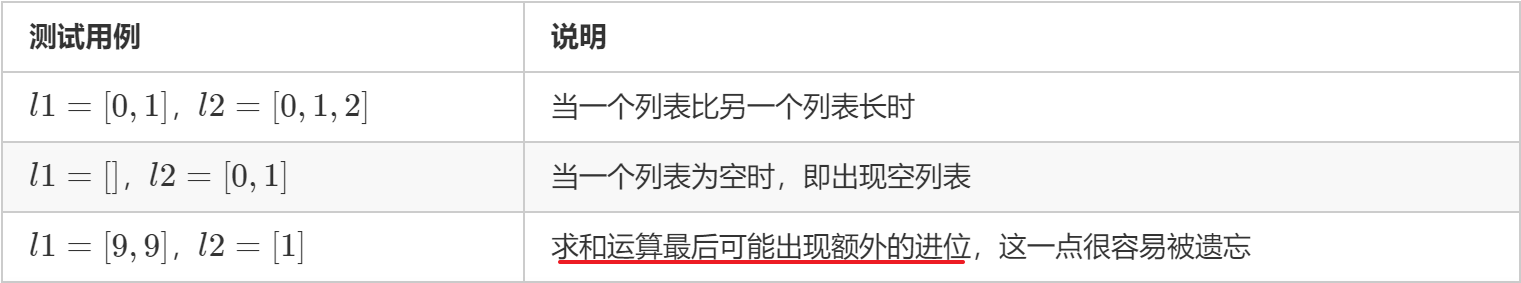
链接：<https://leetcode-cn.com/problems/add-two-numbers>

注意点：关于进位的问题如何处理？

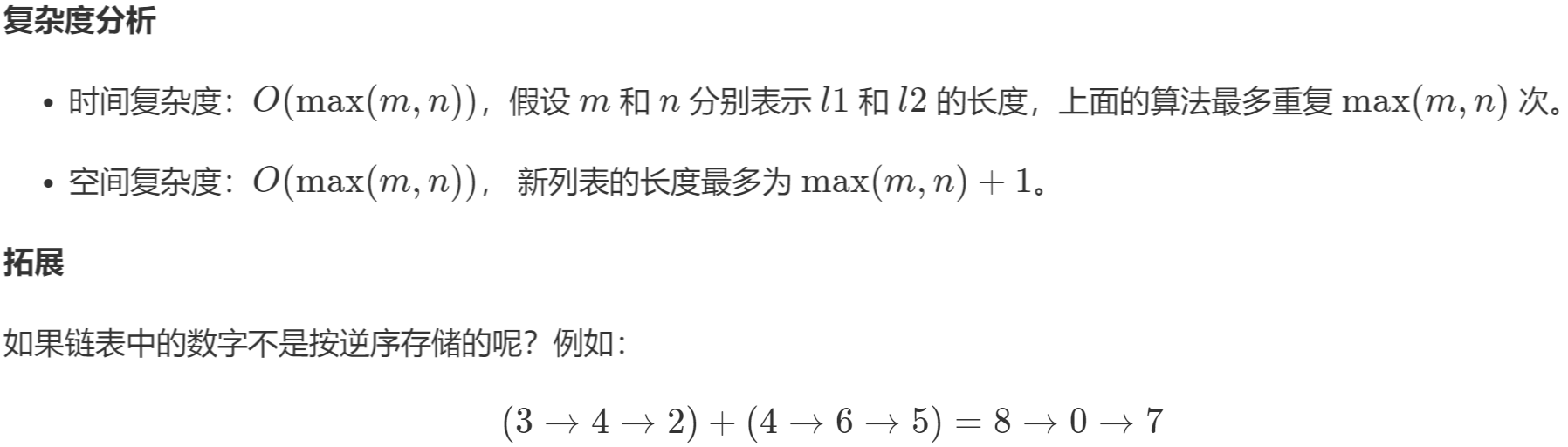


就像你在纸上计算两个数字的和那样，我们首先从最低有效位也就是列表l1和 l2的表头开始相加。由于每位数字都应当处于0…9的范围内，我们计算两个数字的和时可能会出现“溢出”。例如，5 + 7 = 12。在这种情况下，我们会将当前位的数值设置为 2，并将进位carry = 1带入下一次迭代。进位 carry必定是 0或1。





**package** LinkedList;  
  
/\*\*  
 \* Definition for singly-linked list.  
 \* public class ListNode {  
 \* int val;  
 \* ListNode next; // 初始化为null  
 \* ListNode(int x) { val = x; }  
 \* }  
 \*/  
**public class** AddTwoNumbers  
{  
 **public** ListNode addTwoNumbers(ListNode listNode1, ListNode listNode2)  
 {  
 ListNode dummyHead = **new** ListNode(0); // 定义一个哑结点  
 ListNode p = listNode1, q = listNode2, curr = dummyHead;  
 **int** carry = 0; // 进位  
  
 **while** (p != **null** || q != **null**)  
 {  
 **int** x = (p != **null**) ? p.**val** : 0;  
 **int** y = (q != **null**) ? q.**val** : 0;  
 **int** sum = carry + x + y;  
  
 carry = sum / 10;  
 curr.**next** = **new** ListNode(sum % 10);  
 curr = curr.**next**;  
  
 **if** (p != **null**)  
 p = p.**next**;  
 **if** (q != **null**)  
 q = q.**next**;  
 }  
  
 // 防止求和最后出现进位 99 + 1 -> 001  
 **if** (carry > 0)  
 curr.**next** = **new** ListNode(carry);  
  
 **return** dummyHead.**next**;  
 }  
}



## 19. 删除链表的倒数第N个节点(中等)

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.

说明：

给定的n保证是有效的。

进阶：

你能尝试使用一趟扫描实现吗？

来源：力扣（LeetCode）

链接：<https://leetcode-cn.com/problems/remove-nth-node-from-end-of-list>

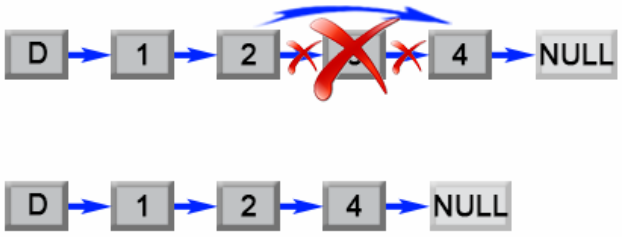
### 方法一：两次遍历算法

思路

我们注意到这个问题可以容易地简化成另一个问题：删除从列表开头数起的第(L - n + 1)个结点，其中L是列表的长度。只要我们找到列表的长度L，这个问题就很容易解决。

算法

首先我们将添加一个哑结点作为辅助，该结点位于列表头部。哑结点用来简化某些极端情况，例如列表中只含有一个结点，或需要删除列表的头部。在第一次遍历中，我们找出列表的长度L。然后设置一个指向哑结点的指针，并移动它遍历列表，直至它到达第(L - n)个结点那里。我们把第(L - n)个结点的 next指针重新链接至第(L−n+2)个结点，完成这个算法。



/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode(int x) { val = x; }

\* }

\*/

class Solution

{

public ListNode removeNthFromEnd(ListNode head, int n)

{

ListNode dummyNode = new ListNode(0);

dummyNode.next = head;

int length = 0;

ListNode curr = head;

while (curr != null)

{

length++;

curr = curr.next;

}

length -= n;

curr = dummyNode;

// 找到欲删除结点的前一个结点

while (length > 0)

{

length -= 1;

curr = curr.next;

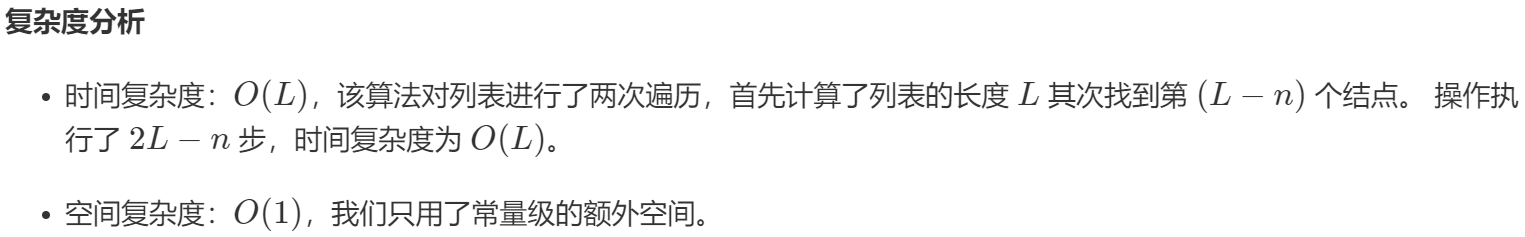
}

curr.next = curr.next.next;

return dummyNode.next;

}

}



### 方法二：一次遍历算法

算法

上述算法可以优化为只使用一次遍历。我们可以使用两个指针而不是一个指针。第一个指针从列表的开头向前移动n+1步，而第二个指针将从列表的开头出发。现在，这两个指针被n个结点分开。我们通过同时移动两个指针向前来保持这个恒定的间隔，直到第一个指针到达最后一个结点。此时第二个指针将指向从最后一个结点数起的第n个结点。我们重新链接第二个指针所引用的结点的next指针指向该结点的下下个结点。

