目录

[1.反转链表（基础->递归 1](#_Toc10961)

[2.删除链表倒数第n个节点 3](#_Toc31322)

[3.如何检测链表中是否有环 4](#_Toc26495)

[4.如何找到链表中环的入口点 5](#_Toc5952)

[5.找出两个链表的第一个相交节点 8](#_Toc17181)

[6.合并两个有序链表 9](#_Toc27672)

[7.单链表升序排序（递归，传换成题6 10](#_Toc4508)

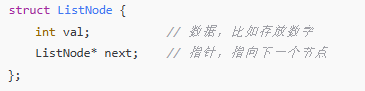
[8.链表的中间节点 13](#_Toc3363)

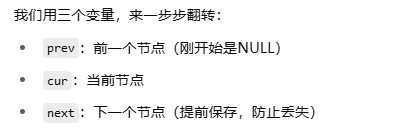
[9.链表分割（使得所有小于x的节点都位于大于或等于x的节点之前 13](#_Toc11783)

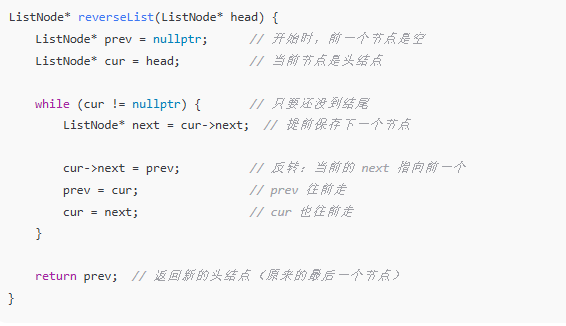
[10.删除排序链表中的重复元素 13](#_Toc18911)

[11.判断回文链表[1,2,3,4,4,3,2,1] 13](#_Toc4092)

### 1.反转链表（基础->递归







个人理解：

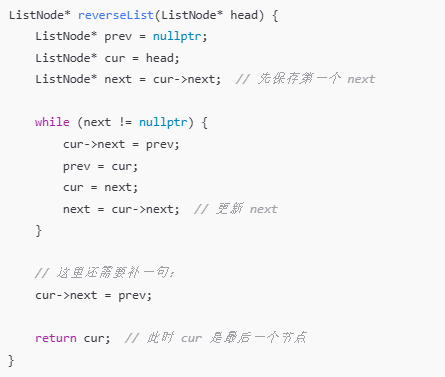
①首先做好准备工作，因为输入的是头节点head，所以我们先让头节点指向NULL，将当前值cur设为head头节点，准备将他指向NULL。前一个节点自然就是prev=NULL

②可以将prev、cur、next三个节点整体向右移动了。先保存next的节点，也就是当前节点cur的->next。

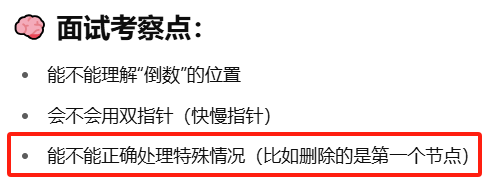
然后开始移动，prev=cur；cur=next；

③返回新的头节点。

自己改的一个版本：（先初始化三个值，再逐个移动）（原来是初始化两个，next在循环中赋值）

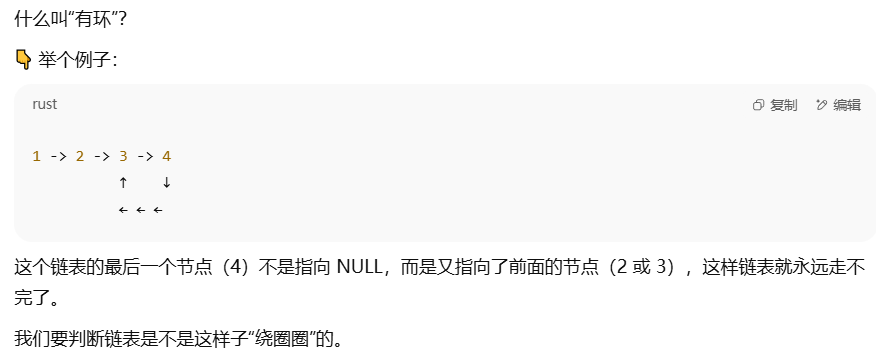


### 2.删除链表倒数第n个节点

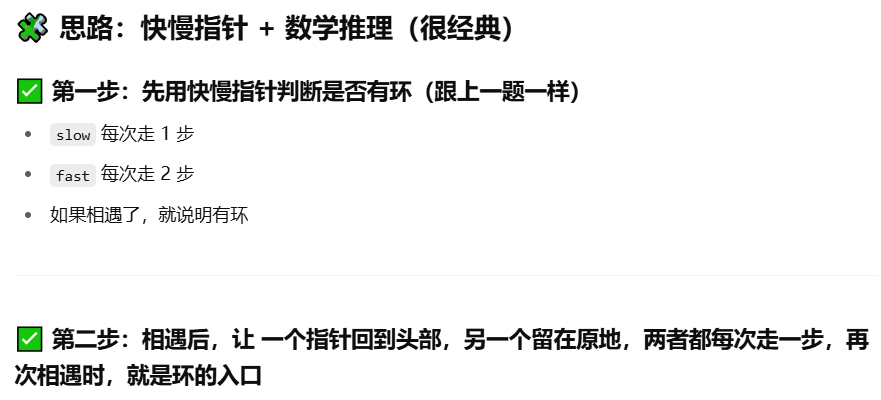


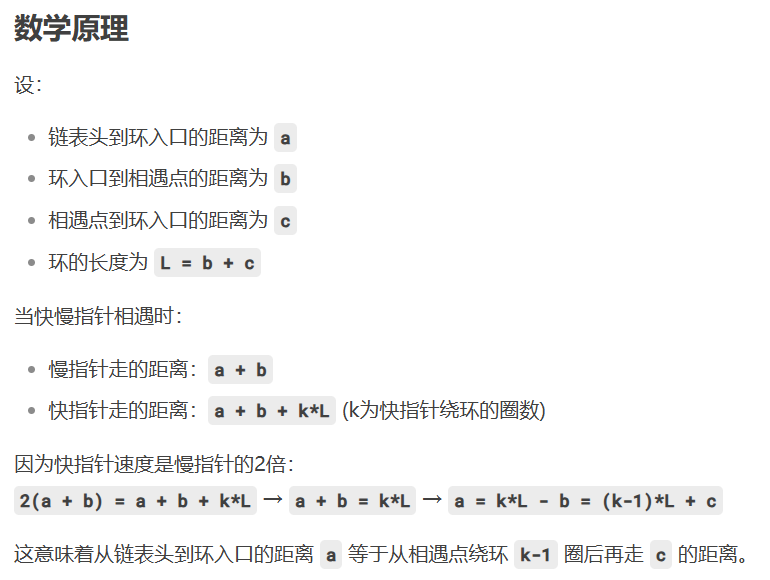
**C++版本**

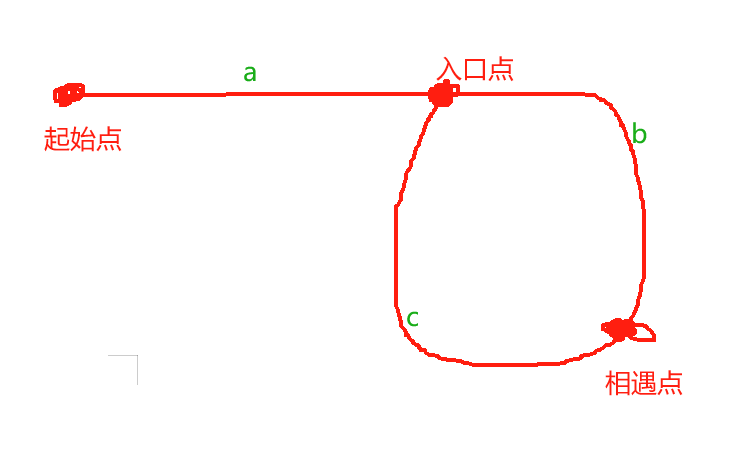


3.如何检测链表中是否有环  
  


### 4.如何找到链表中环的入口点



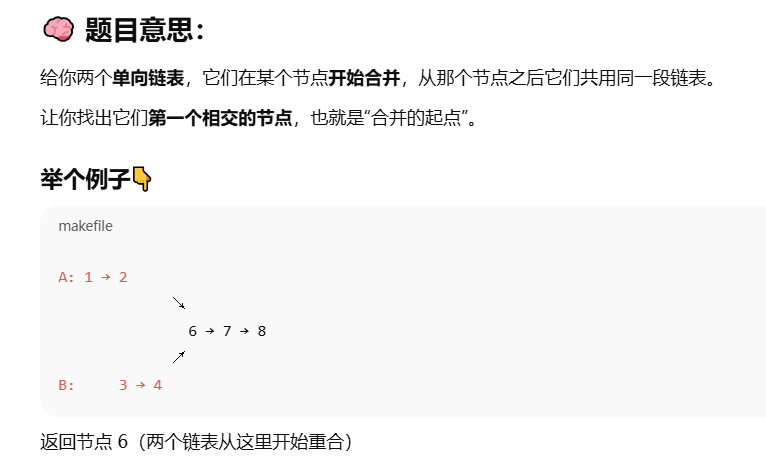


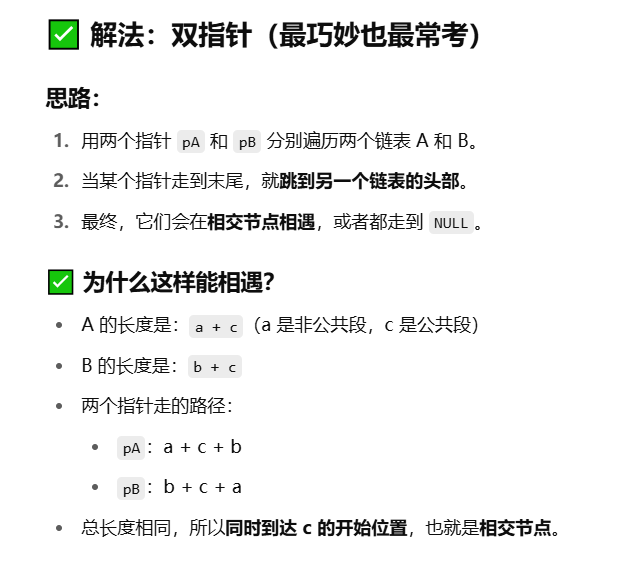


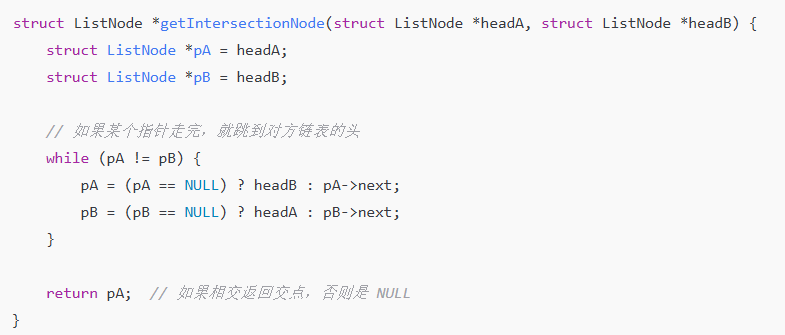
**纯数学经典原理，这样做就是能找到入口**

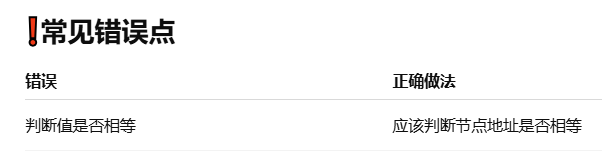


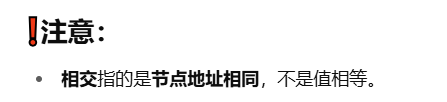
### 5.找出两个链表的第一个相交节点



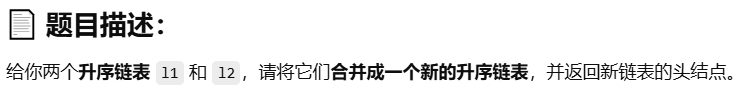


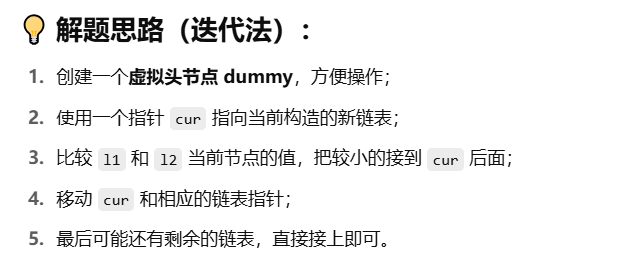


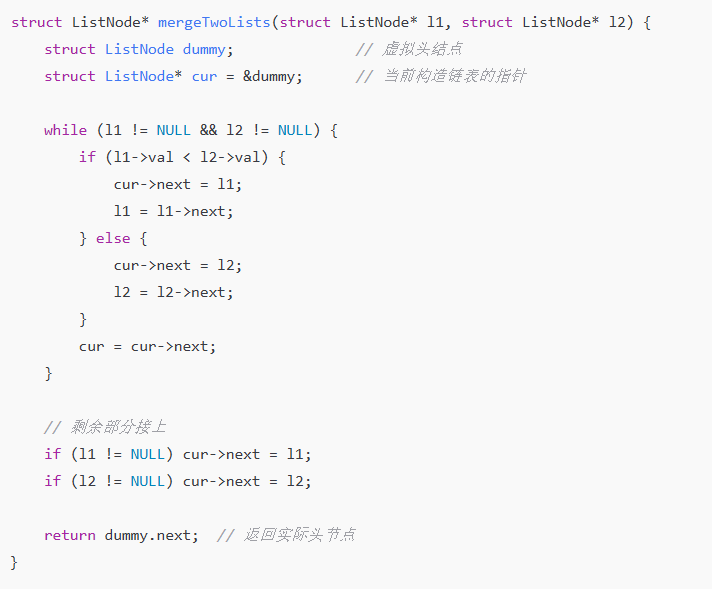


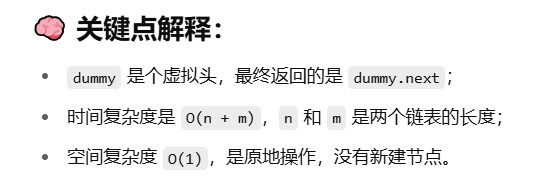


### 6.合并两个有序链表

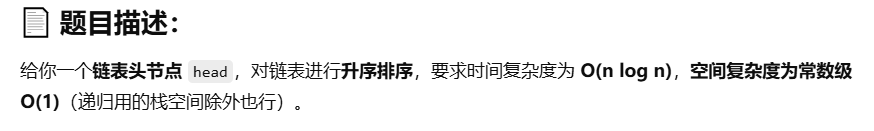






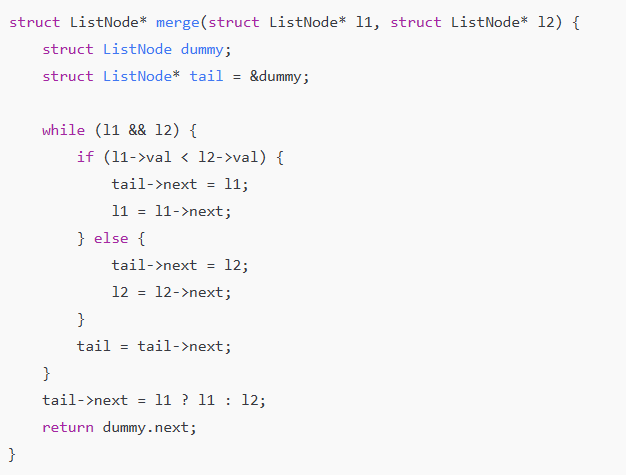


### 7.单链表升序排序（递归，传换成题6





上一题的合并两个升序链表代码：

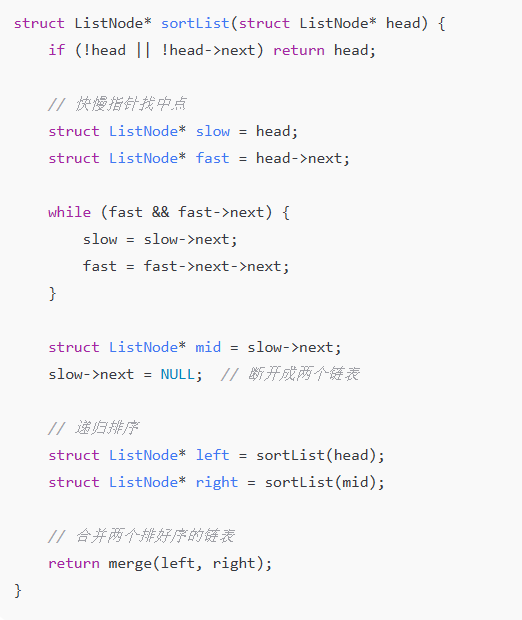


使用递归调用将长链表分解为两个升序链表：

①先将链表从中间节点分解为两个链表

②因为不满足**if (!head || !head->next) return head;**条件，所以sortList函数会一直递归调用。直到将两段链表都分解为单个节点的链表。

③单个节点是最小的，然后逐级merge函数合并，最终合并为左右两个升序链表，再合并成一个升序链表。



### 链表的中间节点

**链表奇数：**

**返回正中间的那个节点**

**链表偶数：**

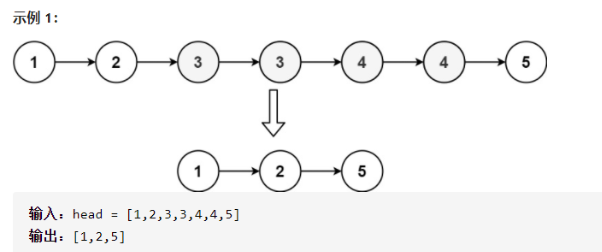
**返回右半边链表的第一个节点**

注意：红框里的条件不同，return返回的指向也不同，如果使用

while (fast->next != NULL && fast->next->next != NULL) {，返回的是中点前一个

### 链表分割（使得所有小于x的节点都位于大于或等于x的节点之前

### 10.删除排序链表中的重复元素



### 11.判断回文链表[1,2,3,4,4,3,2,1]



用短的那段链表进行比较

代码中找中间节点使用的是：  
while (fast->next != NULL && fast->next->next != NULL) {

会返回 奇数：中间节点

偶数：中左节点，

所以我们使用 returm -->next，翻转右半边链表，再进行比较。

如果链表为奇数，分开后两边链表长度会不一样，按照短的进行比较就行。

可以使用while(p1 != NULL && p2! = NULL),可以兼容前半部分短或后半部分短的问题。