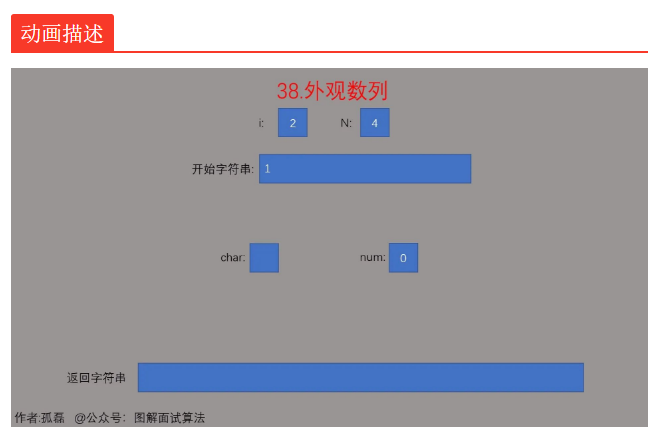
# 简单

## 38\_外观数列

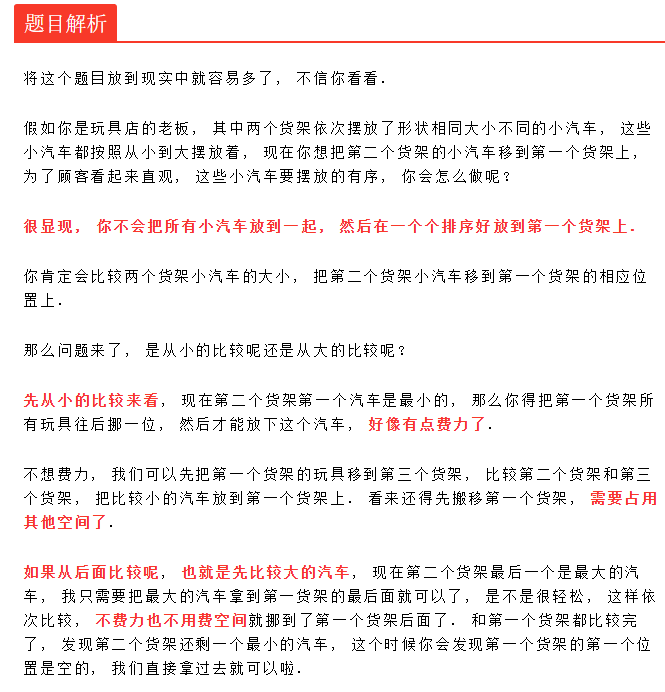


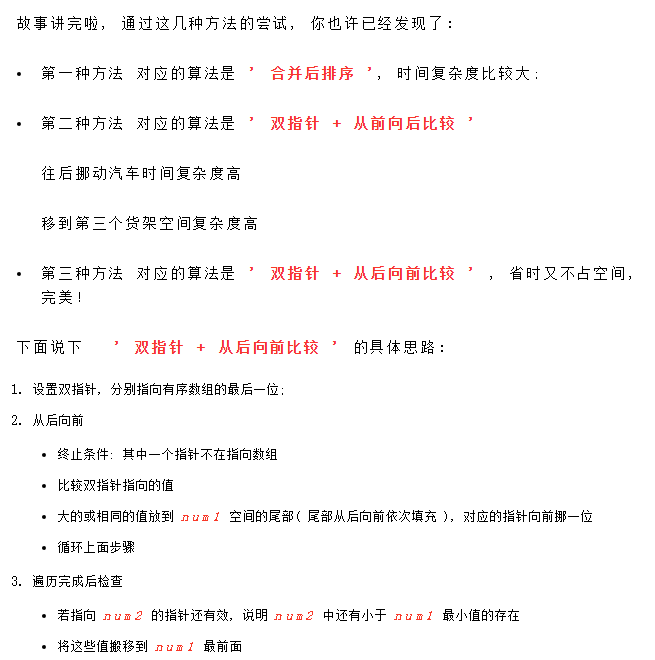


## 53\_最大子序和

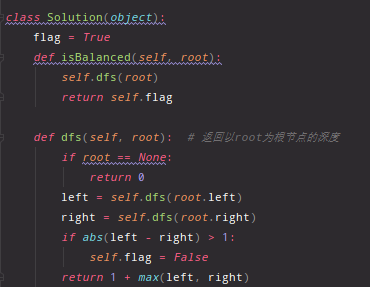
1. maxSum = -2e32
2. Dp[i]表示以nums[i]结尾的最大连续子数组和
3. 记得使用空间优化

## 88\_合并两个有序数组





110\_平衡二叉树



## 198\_打家劫舍

Dp[i]表明打劫前i家商店的最高金额，但是第i家商店的金额是nums[i-1]

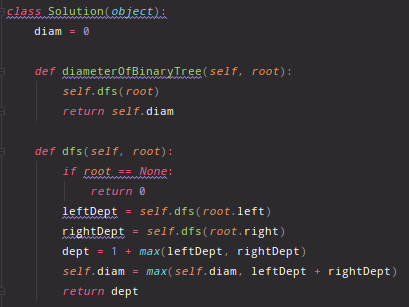
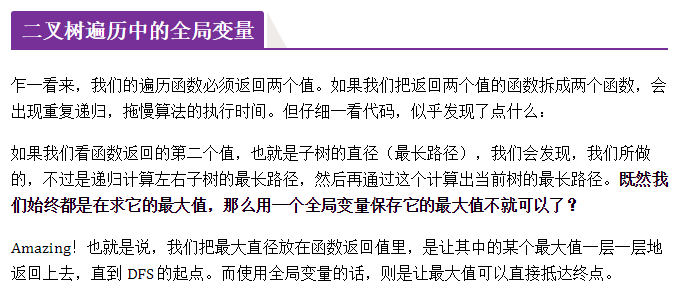
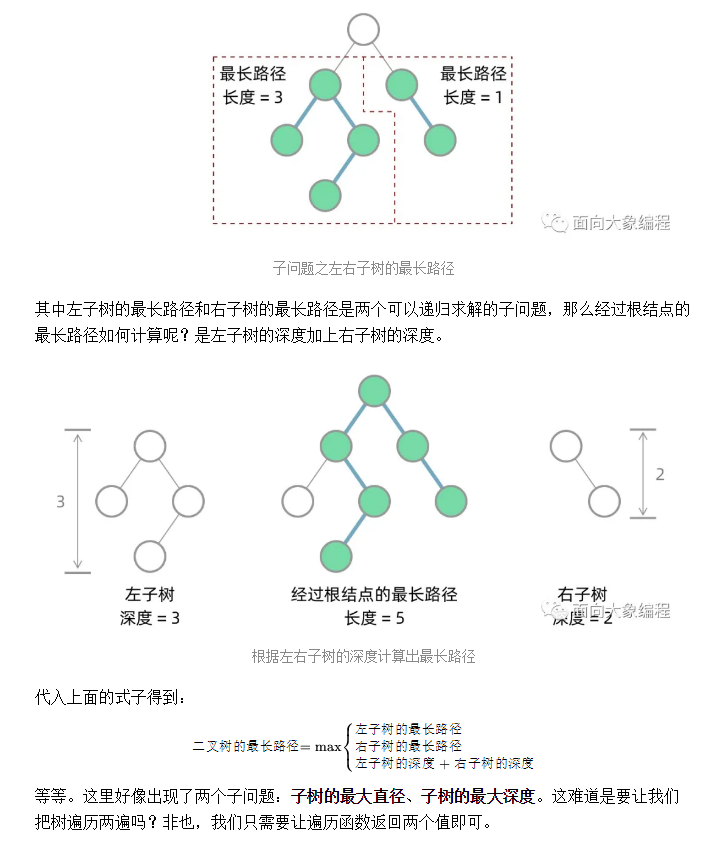
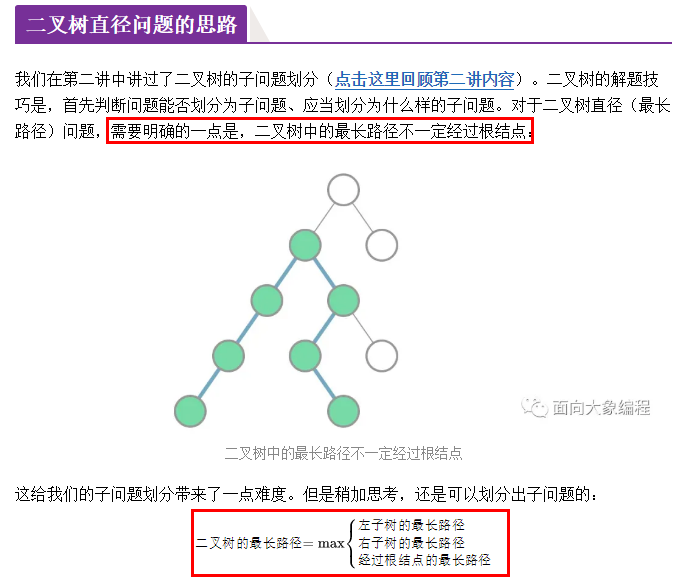
## 290\_单词规律



改进：不需要hashset，只需要讲Map中的values值取出比较即可。



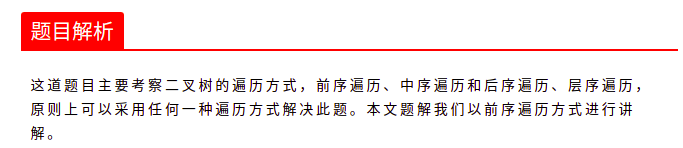
## 543\_二叉树的直径

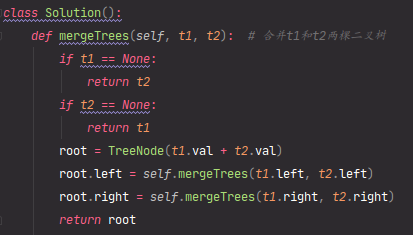


## 563\_二叉树的坡度



## 617\_合并二叉树



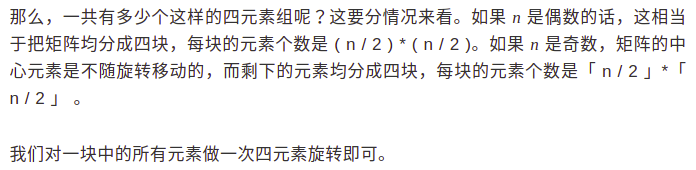


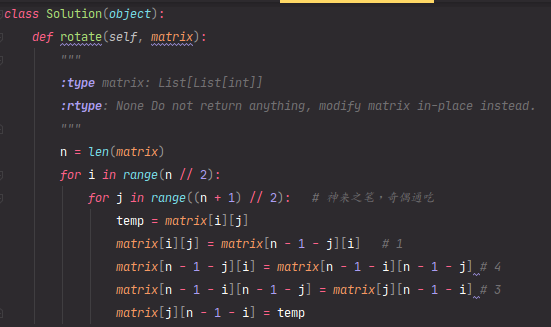
# 中等

## 48\_旋转图像









## 152\_最大乘积子数组

1. 数组中有负数，所以maxProduct = -2e32
2. 相比于最大连续和子数组来说，这里因为是乘积，所以要维持两个数，一个最大乘积，一个最小乘积，前者一定为正，后者一定为负，当遇到负数时，交换两者。这是精髓，一定要记住。

## 213\_打家劫舍2

在打家劫舍的基础上分为两种情况：

1、# 第一间房屋必偷， 那么就要从第三间房屋开始偷起， 偷到第N - 1间房屋为止

2、# 第一间不偷，偷到最后一间房屋为止

重用代码。

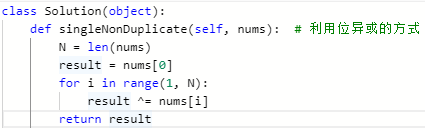
## 337\_打家劫舍3

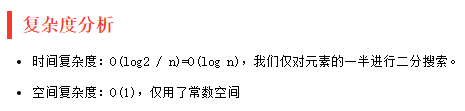
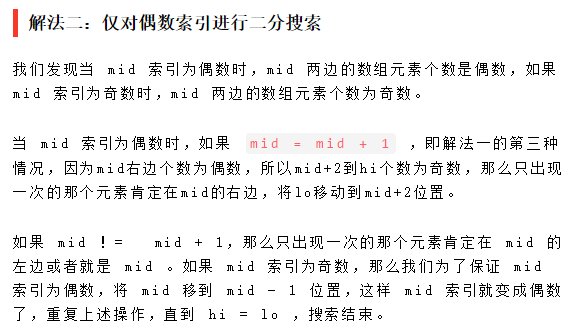
递归函数表示# 从root节点进入 偷root节点和不偷root节点返回的最大金额

注意：兄弟节点不算相邻

## 540\_有序数组中的单一元素

解法一：





二分了N次，直到只剩下一个元素，就是我们要找的那个数:

2 ^ N = n ----> N = logn

## 718\_最长重复子数组

# dp[i][j] 表示A以第i个字符结尾和B前j个字符结尾的最长重复子数组长度



空间优化：无法空间优化，因为递推关系式既设计上一行也涉及上一列

## 978\_最长湍流子数组

1. dp从1开始计数，因为刚开始和当两数相等时，从新开始寻找湍流子数组时，本身计为1，所以从第二个开始循环，相等之后，也相当与从相等之后的下一个数开始循环
2. 设置flag，flag为1表示前面为上升趋势，为0为下降趋势。

## 1372\_二叉树中的最长交错路径

题目不难，难得是如何只进行一次深度遍历，解决方法是：

当该往左节点时，那么就不会往右节点了，那么这个时候右节点就可以重新计算最长交错路径，达到一次遍历，两次运用的目的。

## 1143\_最长公共子序列

参考718题

# 困难

## LCP 14\_切分数组

用gcd（a, b）> 1来判断太过耗时间，只需求出b的所有质数u[]，看a%u[i] == 0是否满足，满足就表明最大公约数大于1，这是题目设计好了的，就是要你用质数筛。

## 23\_合并k个排序链表

结合归并排序，转化为合并两个排序链表的算法。

## 124\_二叉树的最大路径和



应该划分为：（倒数第三句）

1. root.val + 左子树的最大路径和（因为右子树为负）
2. root.val + 右子树的最大路径和（因为左子树为负）
3. Root.val + 左右最大路径和（左右子树都非负）

## [1373\_二叉搜索子树的最大键值和](https://leetcode-cn.com/problems/maximum-sum-bst-in-binary-tree/)

我的思路：取一个全局变量，从上往下记录最大键值和self.sum，当不满足二叉树的性质时，self.sum清0，从不满足二叉树性质的那个节点所在层从新计算。

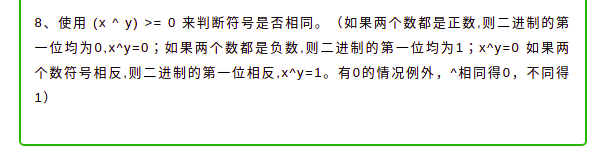
阶梯思路：上面的思路错误，我的想法是，从上往下，这样是大错特错的，因为加入底下不是二叉搜索树，那上面更不会是，所以应该采用的是自底向上，这里的思路很巧，返回三个值，以root为根节点的二叉搜索树的路径和，该二叉搜索树的最小值和最大值，这时应熟练处理，当root为None和当root为根节点的二叉树不是二叉搜索树时的情况，将题目条件运用的非常巧妙。



# 数据结构

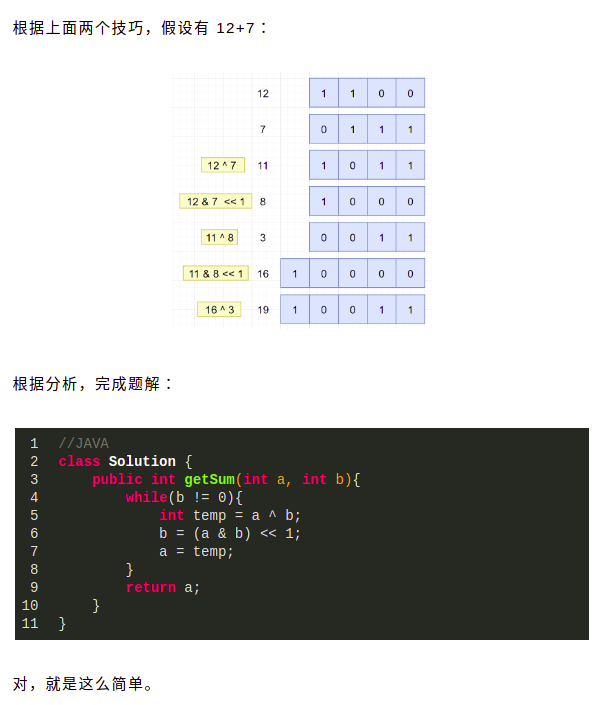
## 位





具体实现看代码： 数据结构--位

### 两数之和

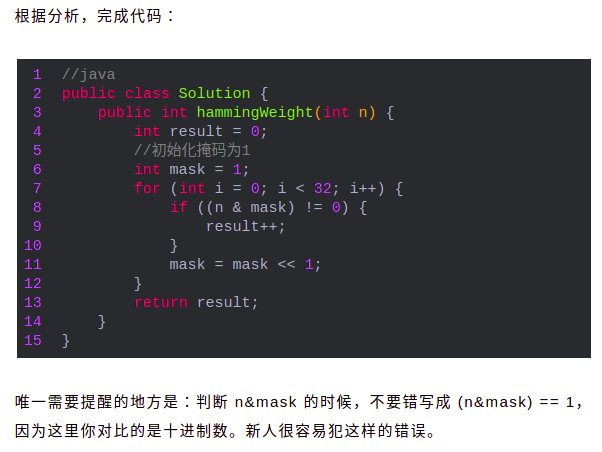


### 是否是2的幂



### 汉明距离





而且 *# 最好将1向左移动，而不是将数向右移动，因为算术右移的话，左侧补的是符号位*

### 只出现一次的数字（其余的数字出现也是奇数次）

## 链表

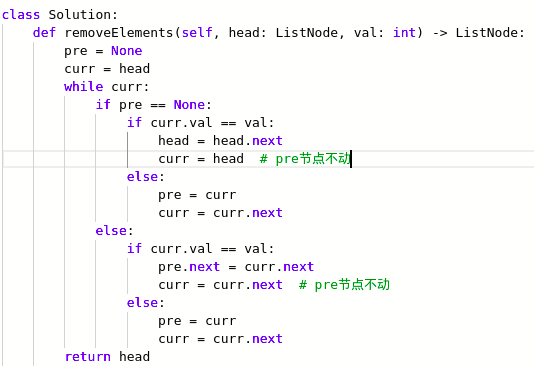
### 链表框架



链表框架最好加上newHead，newHead.next = head， 然后pre = newHead，这样就不用解决pre == Null的问题了。

### 移除链表元素

这个不能照搬链表框架，因为当删除元素时，pre会指向那个删除的元素



### 反转链表的k个元素：

这是一道运用原来的反转链表思路会超时的题目：还是运用我之前的解法解法比较好，因为要求时间复杂度为O（n）。运用改良过后的链表框架解题，值得注意的几点是：

1. 要记录开始反转时的链表的节点
2. 当前节点cur通过pre.next来获得。
3. 当index == n时，直接返回newHead.next即可