1. 正确括号:



2.外星字符串排序，注意continue的新用法，跳过多余判断。

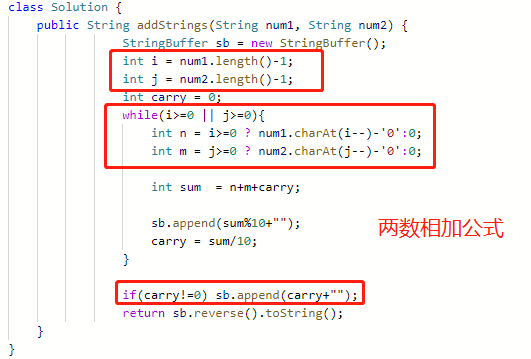
多了一个search控制continue的方法，果然外星。



2.回文判断, 收缩用法:

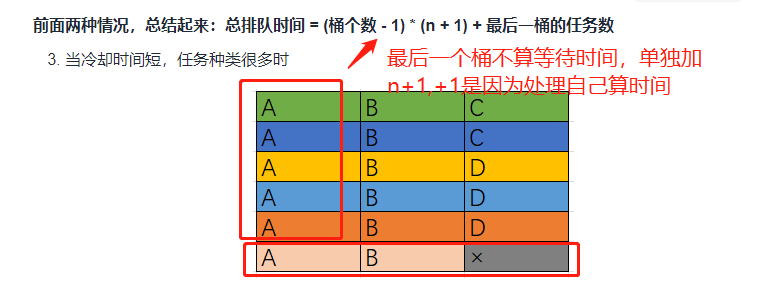


1. stringbuffer的reverse方法:



1. cpu任务:

看图明白那个公式是怎么来的:





1. 纵向遍历原题:



这是层序遍历对每一个元素的操作，并不需要控制是第几层，所以就没有加int size = q.size()然后for循环的束缚。

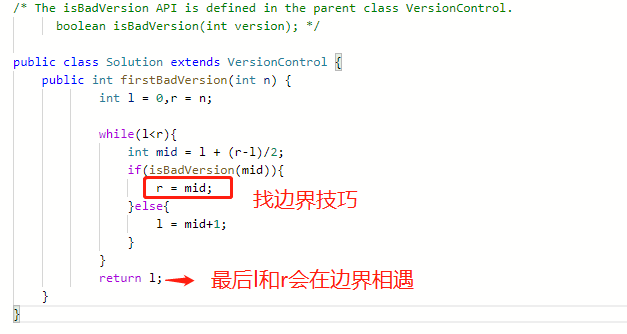
关键点都在红字了。

1. 二分查找找边界技巧:

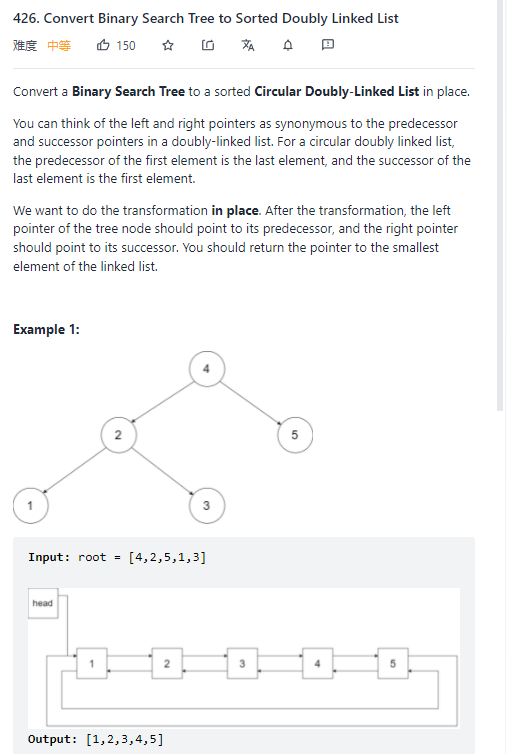
比如 0000011111111

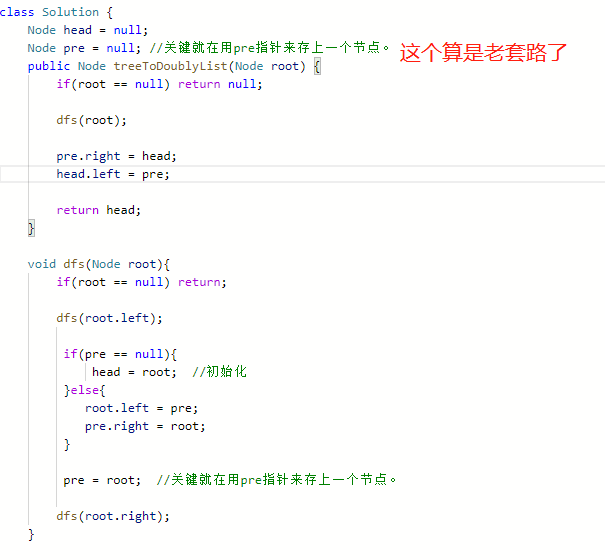
这种找1的边界技巧，就是往右边收缩，遇到1之后，right不-1，而left会一直往右边缩。

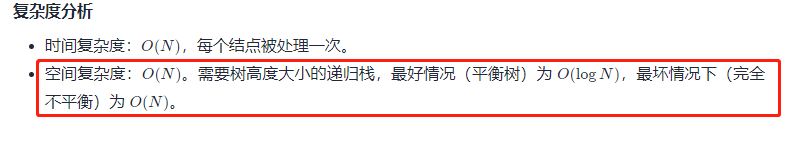
反向同理



1. 二叉搜索树成环，pre指针存上一个节点，老套路了:

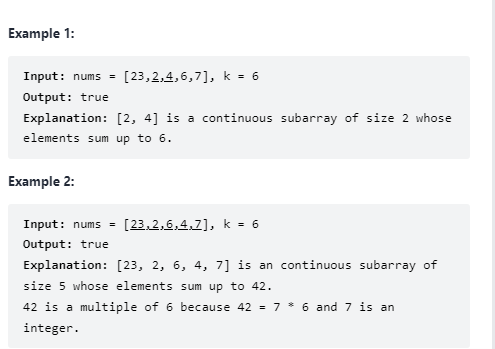






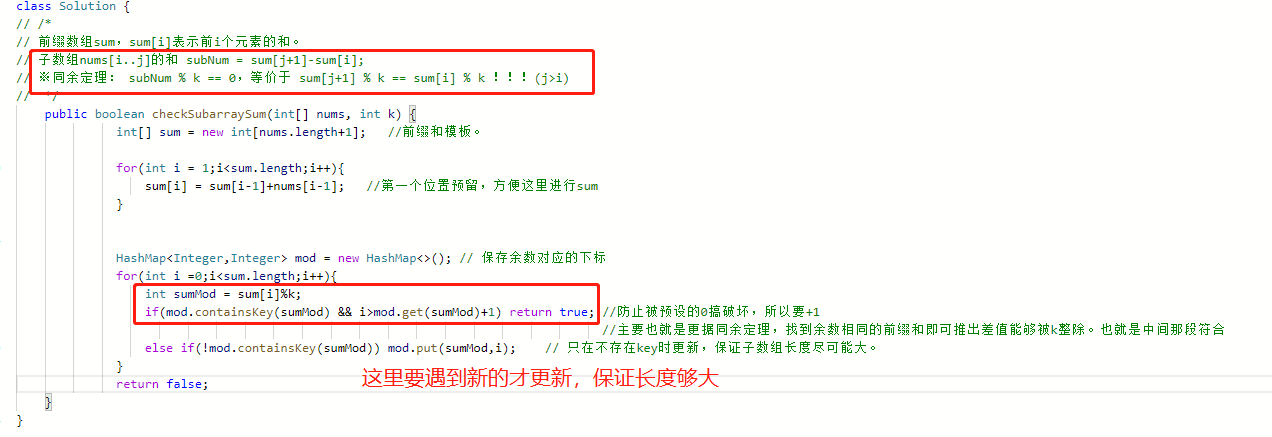
注意复杂度的最好最坏。

1. 连续子数组的和:



这题完全是更据同余定理去反向找答案。

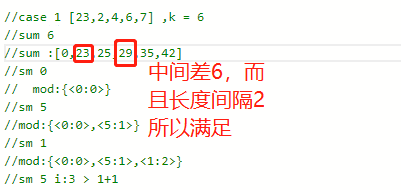
这种前缀和的题型感觉都很典型。一眼前缀和



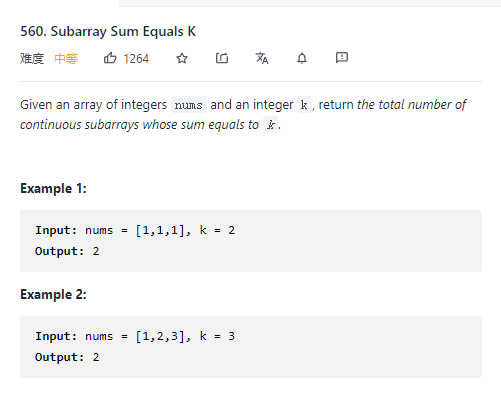
1. 那个地方需要满足+1,因为只差1的话，就表示中间有一个k满足，但不是“至少两个”所以要加1

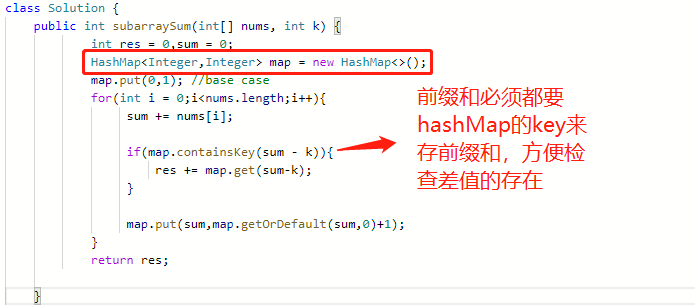
公式：

1. 0的存在位是必须的，补位
2. 前缀和都要有一个hashmap的key来存前缀和，用来方便判断差值的存在。



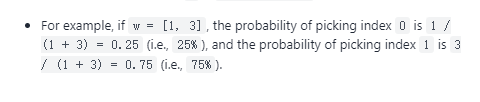
还有一道很经典的前缀和:





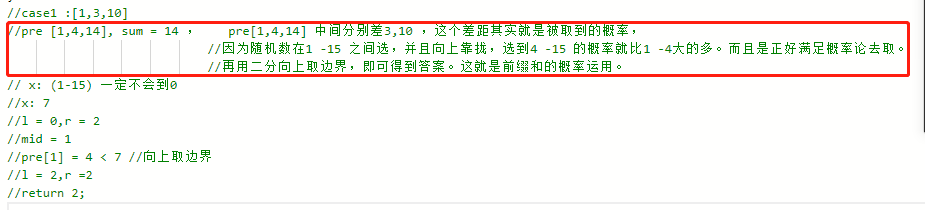
1642042931(1)

再补:前缀和在概率论上的实现:

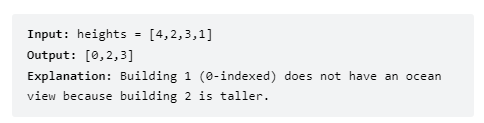




这里因为不用考虑取差，所以不需要前置0计数。而且返回的是对应的index，要保持一致。

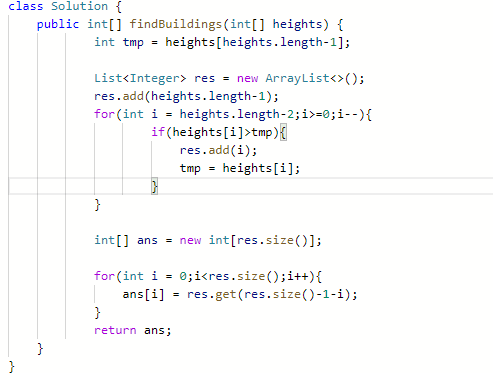


1. 海景房:



右边是海。

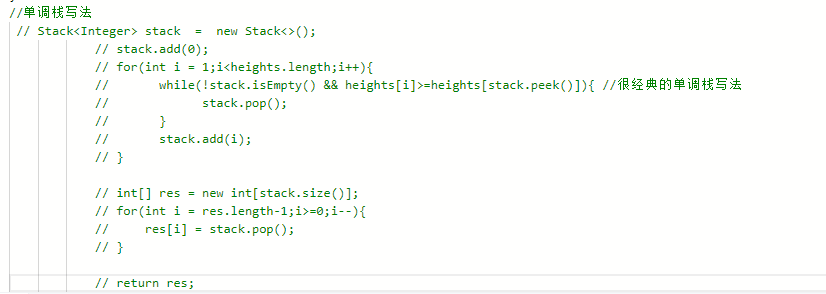
这题从后往前遍历，记录最高楼即可。



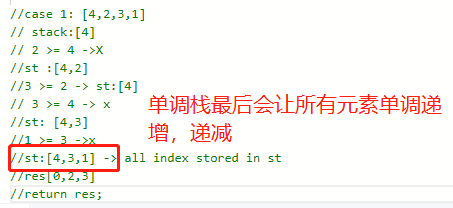
还是挺简单的。这种方式很快，比单调栈快得多。

但单调栈还是应用广一点

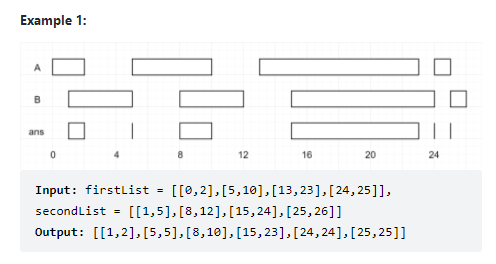
还有一种单调栈写法:



最后栈中排列出来的是单调递减的序号。



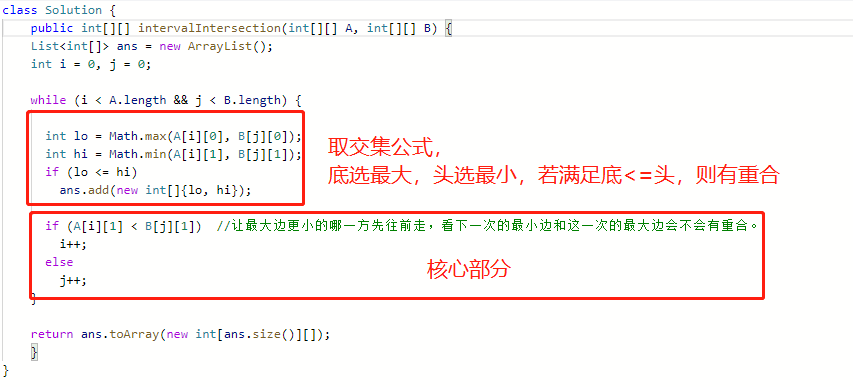
1. 重合数组:

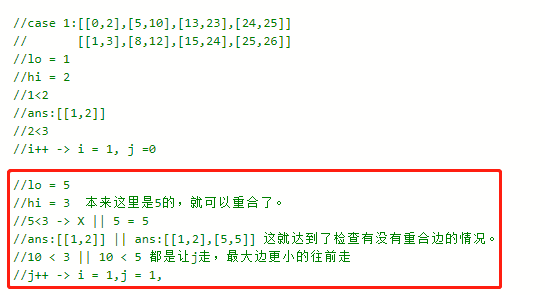


双指针解法

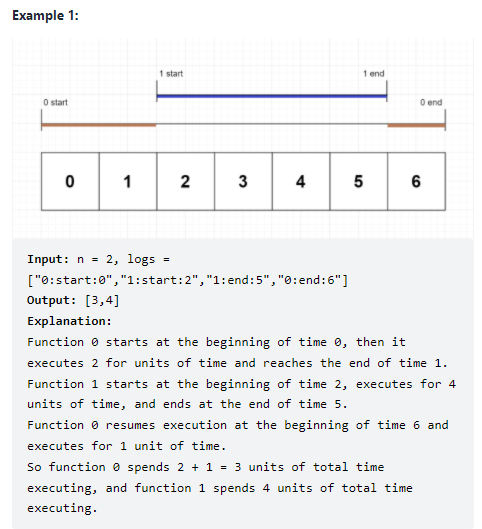
这里有个取交集公式（底大头小）:

还有弄明白为啥头边小的先往前走，因为要看看下一个底边和这次的大边有无重合。



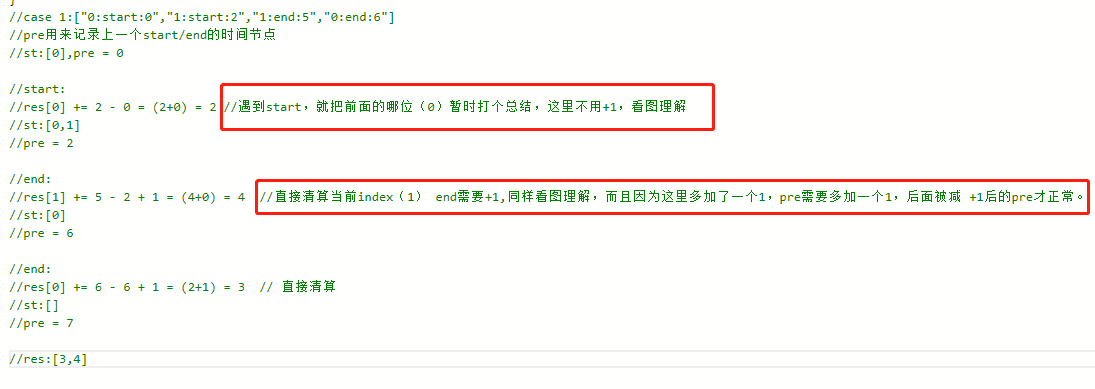


1. cpu时间:

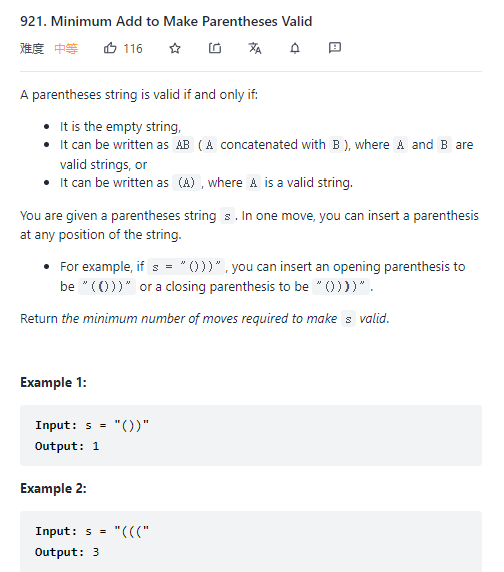


主要就是记住end的时候pre要+1回去，后面的运算才正常。





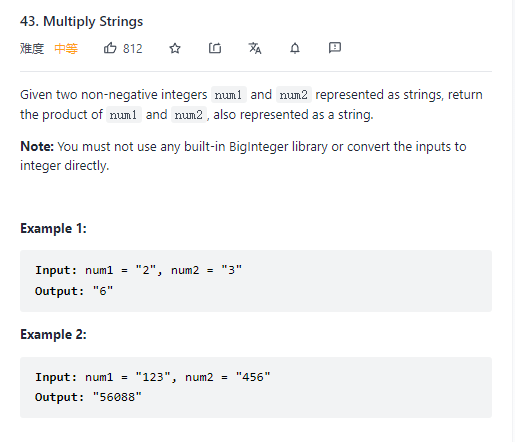
1. 额外的’()’实现valid parentheses:





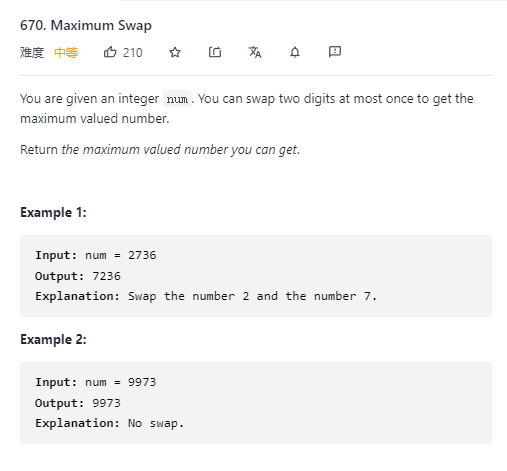
关键就是 ：count用来记录可用的’(’

1. 模拟乘法列式计算:





1. 交换一次使数最大:



核心就是前面的小的换后面的大的。

越前面的位置换，换的数越大。

所以需要检查其位置后面有没有比他大，且最大的数。

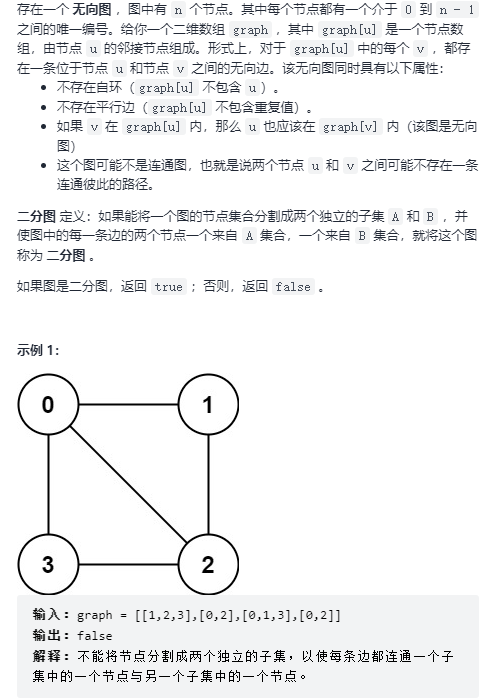
这样就需要记录每个数最后出现的位置了。

还是很好理解的。



补：Integer.toString()比 num+"" 快得多

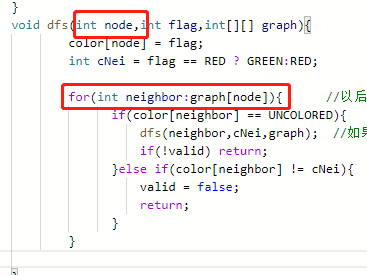
1. 判断二分图:



看红字即可。实现还是很好理解的



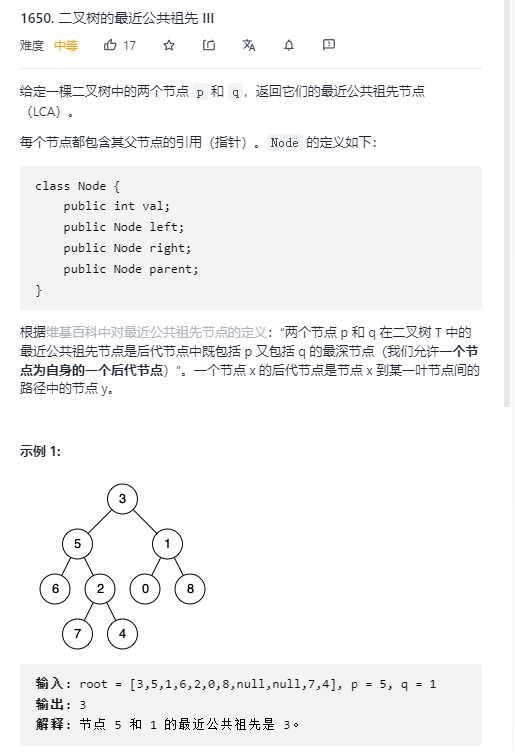
以后图的dfs()，其中统一进入的节点命名为node，其邻居用增强for循环表示为neighbor



1. parent找最近公共祖先:

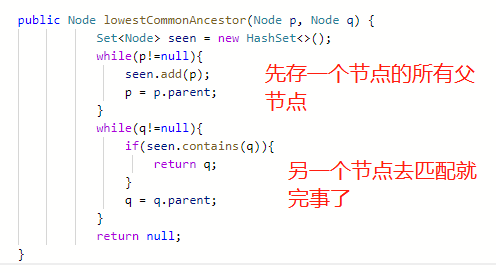
变形题

主要是要记住，不仅仅有dfs这一种方式遍历树，while节点遍历也可以。

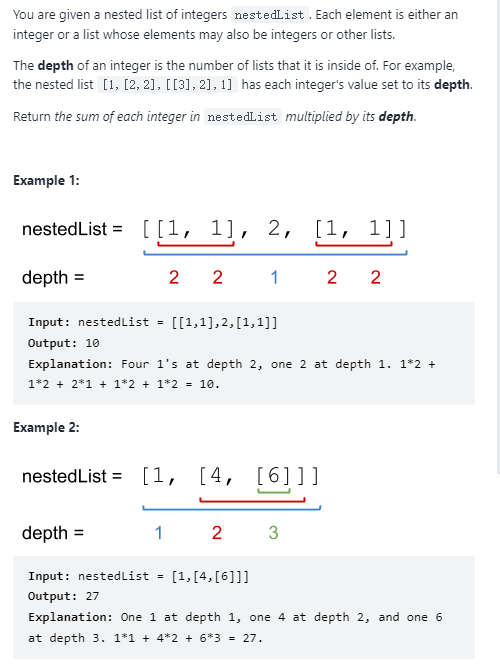


思路还是很暴力清晰的。

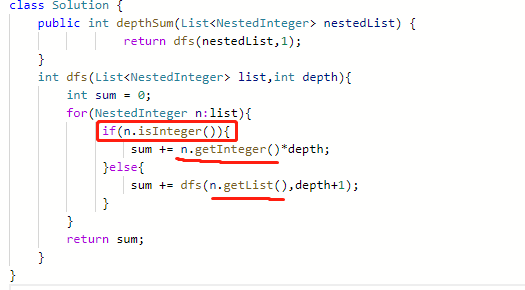
如何分开遍历？就是靠的while



1. 嵌套列表:



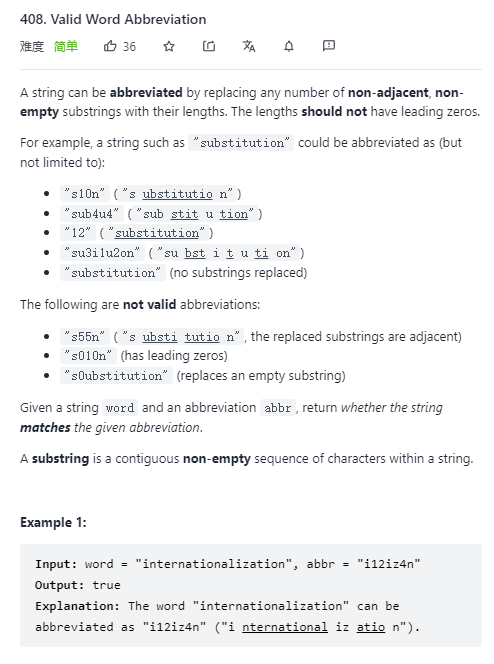
解题是很简单的，就是dfs即可，但就是要看懂api怎么用的



isInteger 这个位置是数字、否的话这个位置是list

然后分别有getInteger和getList的方法去获取该单位。

1. 计算长度:



就是计算长度，中间那个不能在一起的限制其实是无效的，最后结果会反应出来

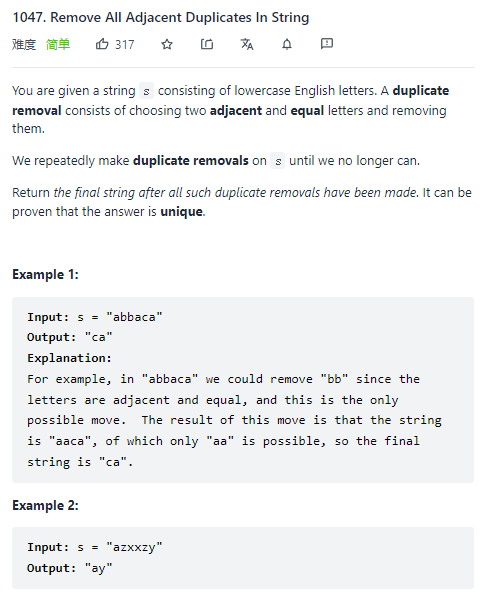
关键是用tmp来记录连续长度。

主要注意前导0的判别，和遇到字母时先加长度，后看特殊情况

解法还是很浅显易懂的。



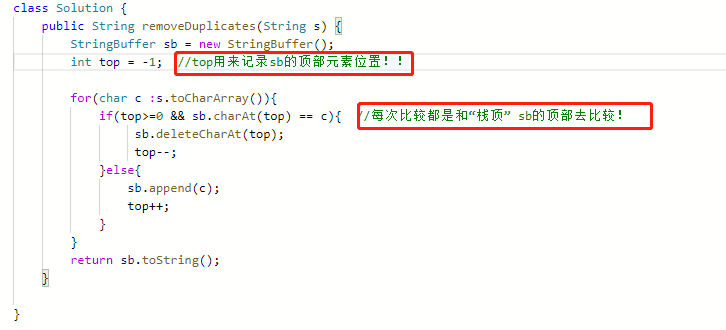
1. 清除相邻重复字母:



不用想多成回文串了，用栈一步步来就好:

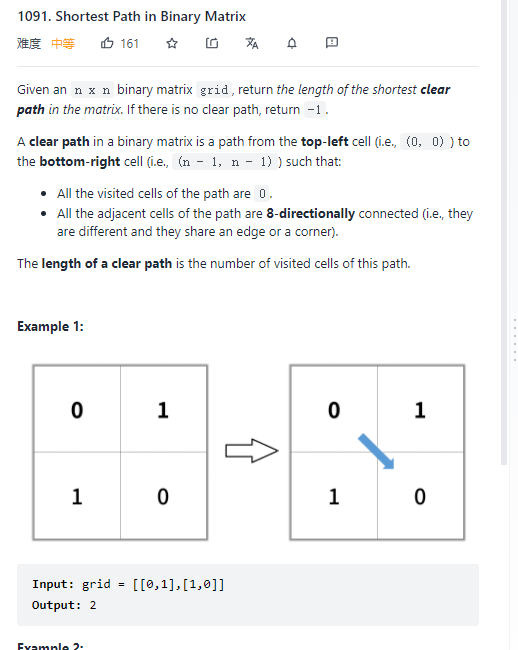
sb就是那个栈，每次比较都是和“栈顶”sb的顶部去比较！

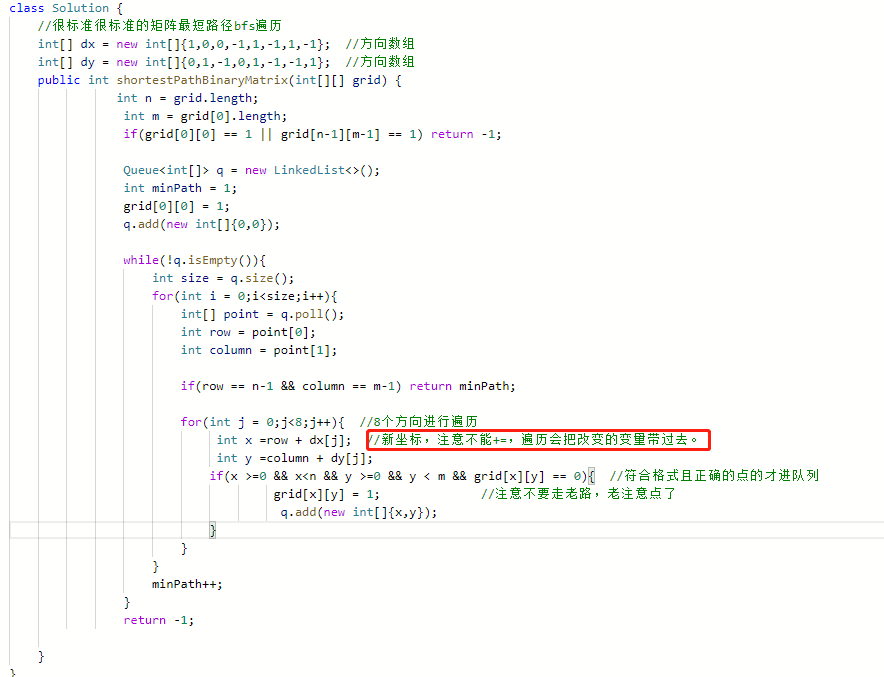
注意top初始化是-1，因为后面操作（0也是需要比较的份子）。



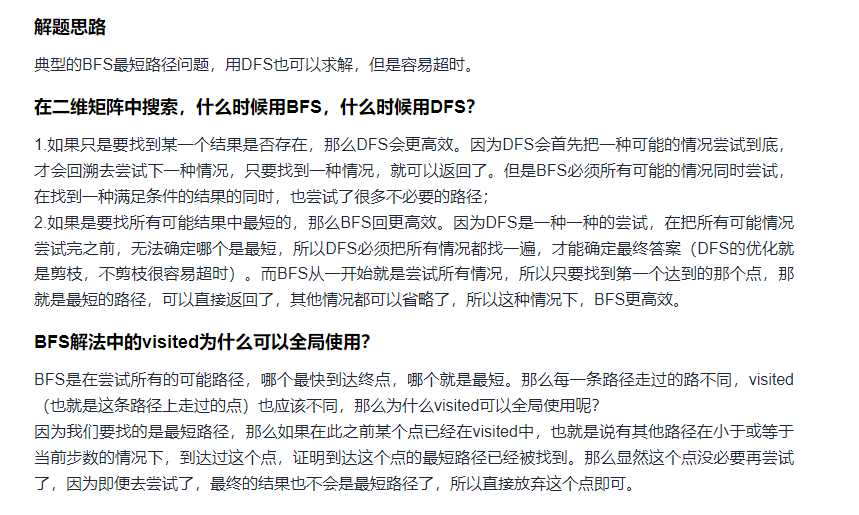
1. 图的bfs最短路径标准例题和标准解答:

只要是图的最短路径，一定用bfs





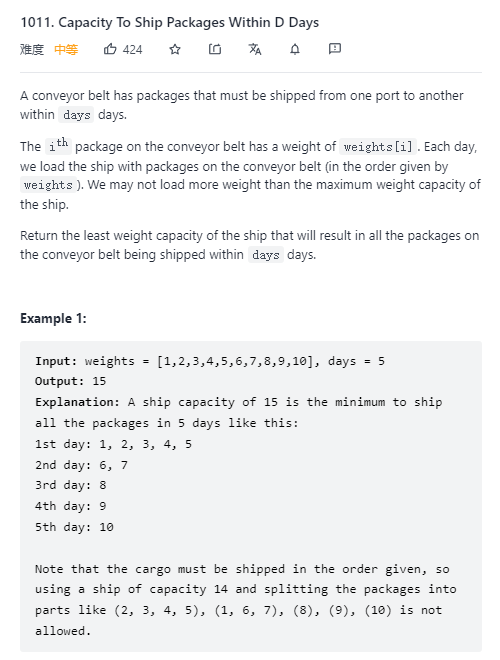
这个得多写几遍，很标准的图的bfs，和树的bfs也特别相似。



目前来看只有图的最短路径需要bfs，一般来说都是dfs

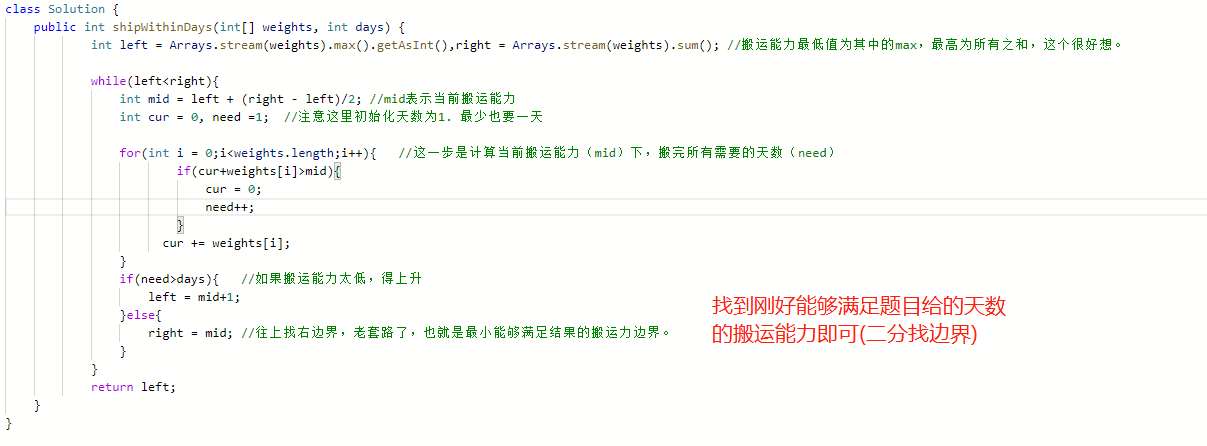
1. 二分查找 搬运能力:

很典型的一道二分查找找边界的问题:

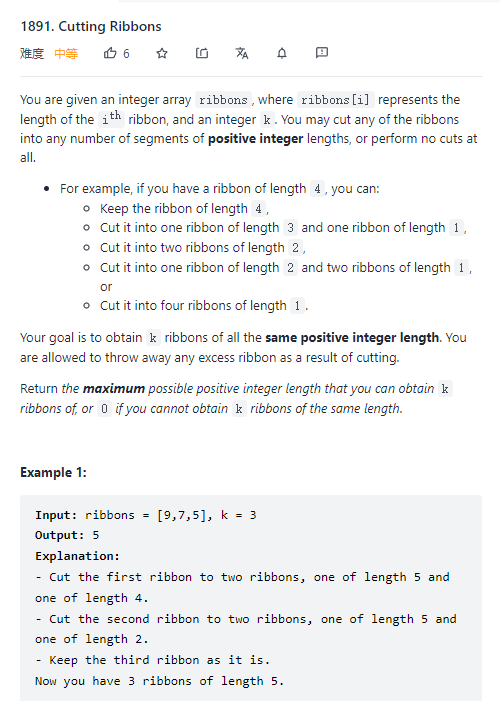


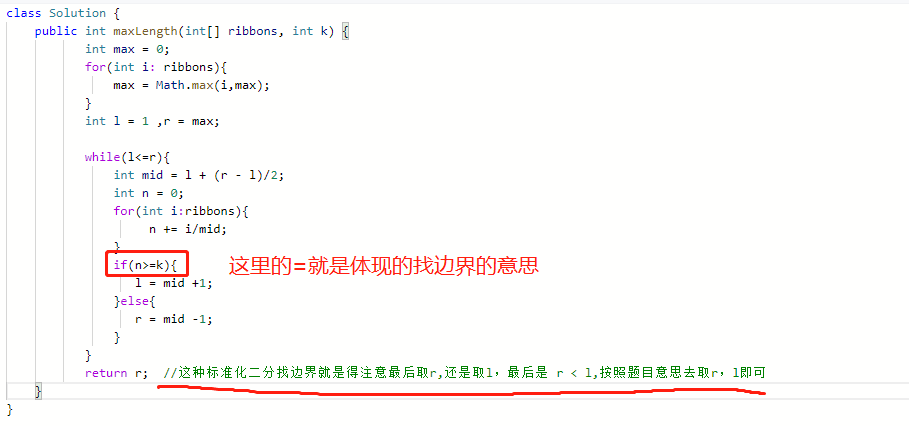
其把可能的搬运能力排出来，然后用二分查找一个个试，找到最小的满足天数的边界值即可。（二分查找找边界）l

理解红绿注释的意思即可。

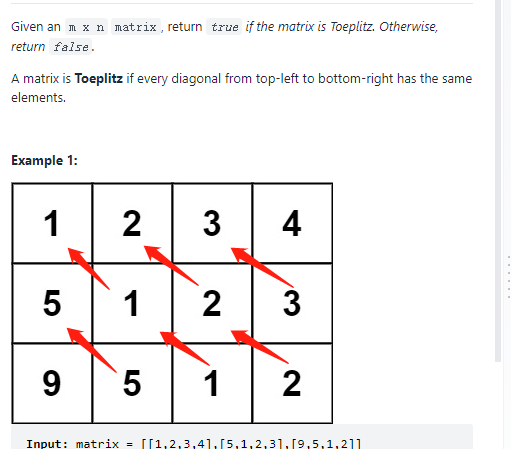


再补一道相似的二分:

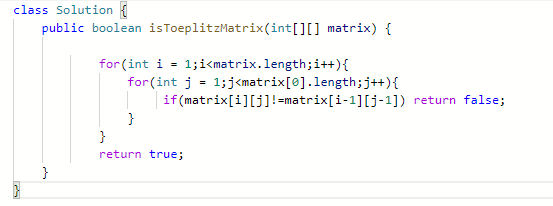
也是以能力值为区间，进行二分找边界，只是公式是标准化的，左右都有加减的情况，



#### [Toeplitz Matrix](https://leetcode-cn.com/problems/toeplitz-matrix/) 检查:



并不需要对角线遍历，而是一一检查即可



1. 强转double:

1642817681(1)

这里是先将sum转double之后，和q.size()相除，结果也是double类型了，等价于:

1642817777(1)

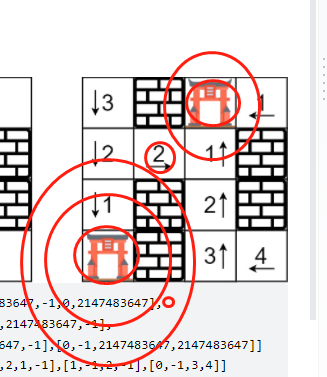
1. bfs的多起点同时开始扩散，确保最短路径:

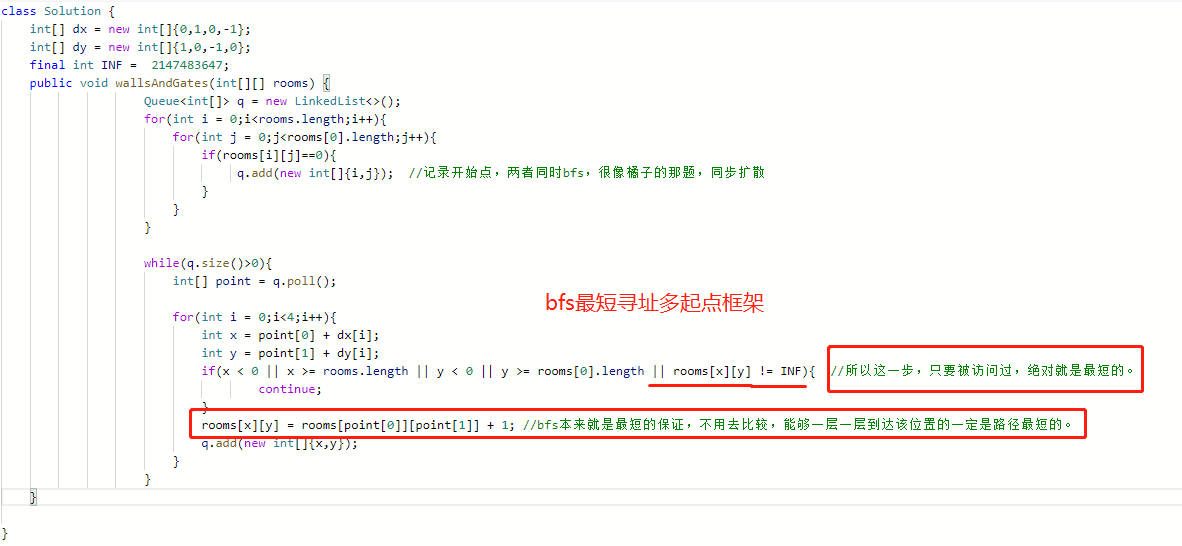
很经典的bfs寻最短路径，因为bfs是多起点同时开始扩散（起点先入队列），如下图，

上面的起点只要走两步就到了中间红圈，而下面的起点要走三步，

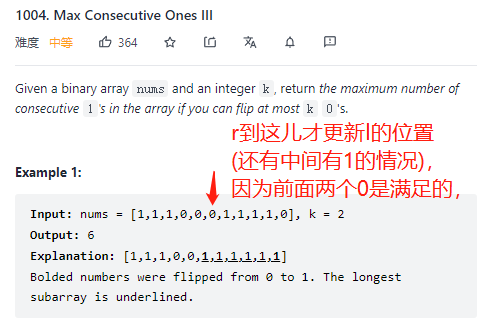
因为同时起步，在下面的起点遍历第三层的时候，该点已经被上面的起点在第二层遍历的时候覆盖掉了，这就保证了bfs的绝对最短路径。

所以bfs的最短路径寻址，多起点一定是同时进行！（先入队起点，再入队各自的第一层，再入队各自的第二层。只要已经遍历过的点，一定是最短的，直接跳过）



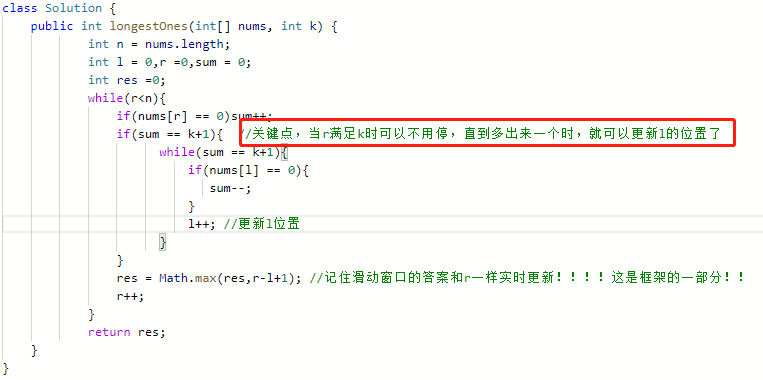


25.翻转后最长连续子数组1(亚麻里有道类似的，这道是最大承载几个0)



就是注意滑动窗口框架，res跟随着r一起更新！！！（别作了，都是这么写的，当成公式吧）

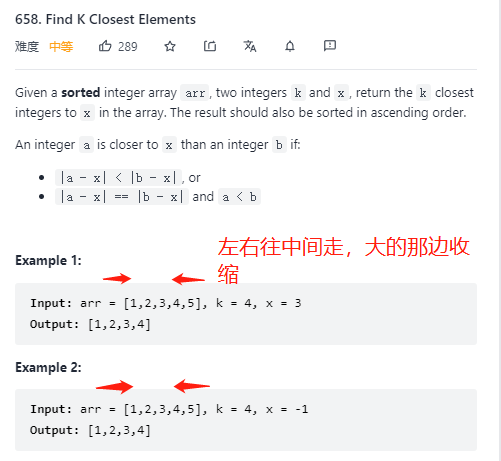
l的位置要不满足了才前进。



26.离x最近的k个值：

这里用收缩法去做，也可以用排序器，但需要先构造List<Integer>.

记住左边是比x小，右边比x大。





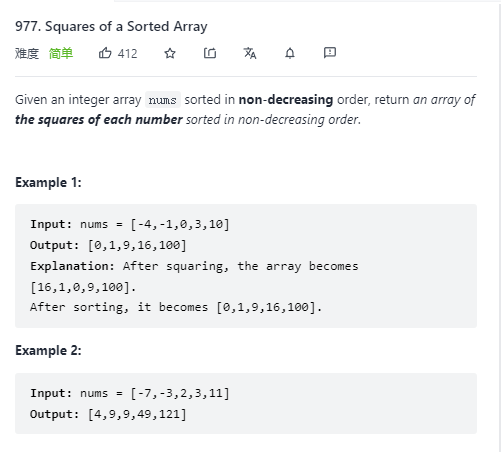
1. 双指针排序:

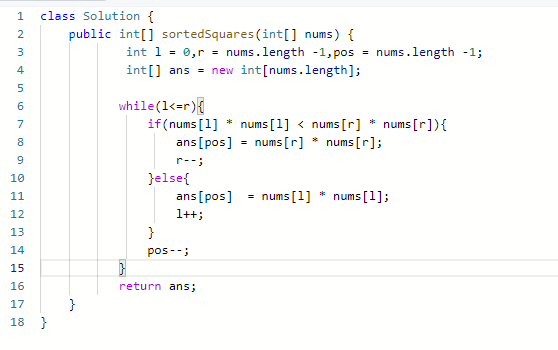
稍微注意一下，有时候用排序器就太简单了。

这种双指针排序只适用于这种两边向中间收缩递减的数据集

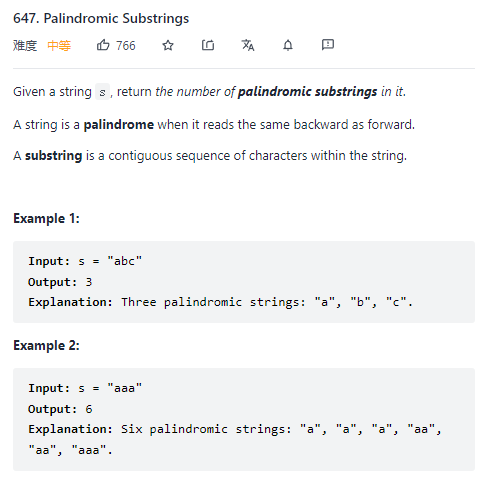
是特殊解法，一般乱序是不适用的。

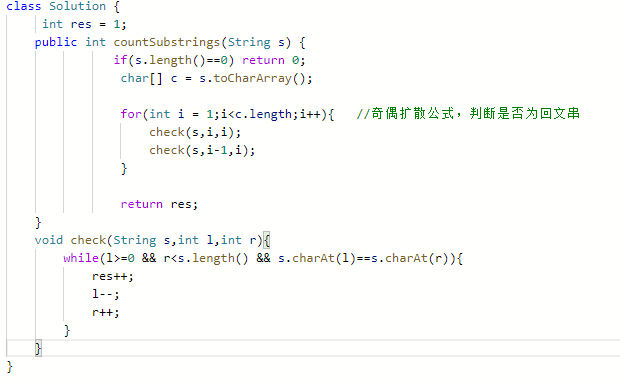
就是左右分边开始比大，谁大谁就占一位，小的先蹲着，等比成大的了在占。





1. 奇偶中心扩散公式，判断回文:

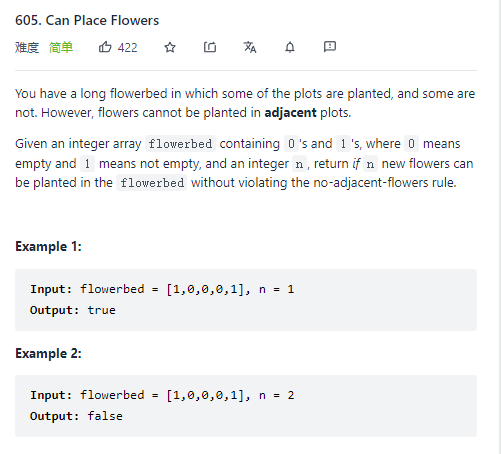


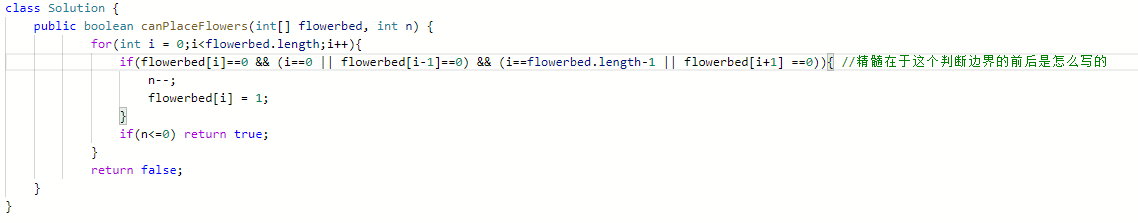


1. 种花：

题解很容易，但有一个可以提取出来的公式:

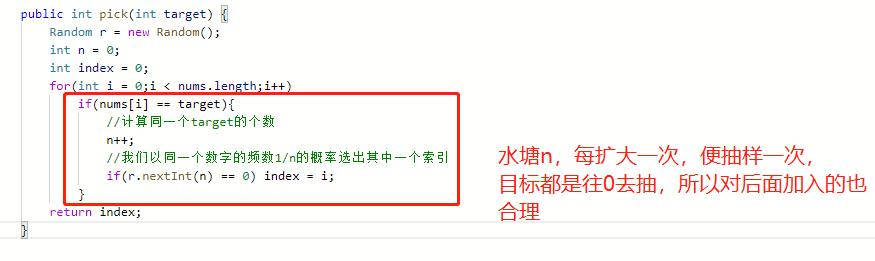
边界前后是怎么处理的，要记住:





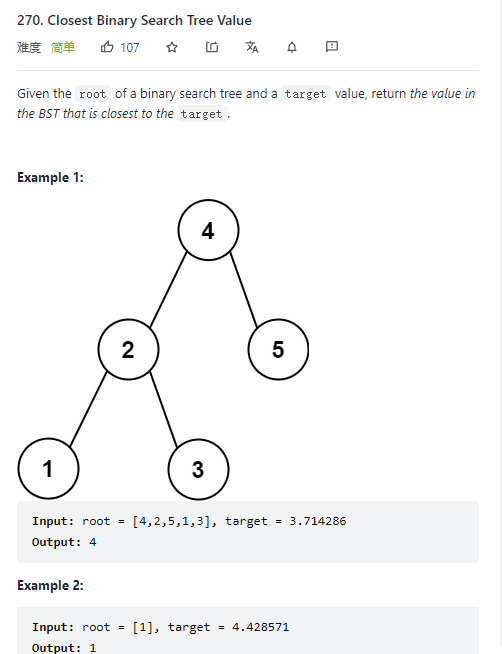
1. 水塘抽样公式:

用来解决同一属性多结果，平均随机返回的概率问题:

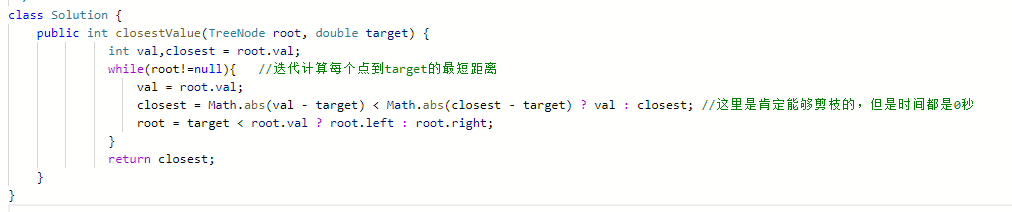


不要全部放在一个list里面再去随机找索引，这样慢很多

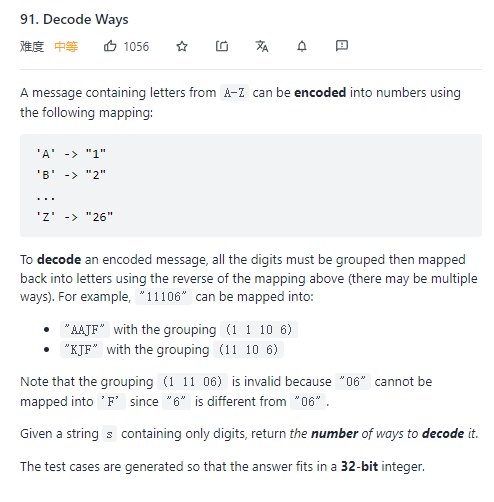
1. 树的最近节点:

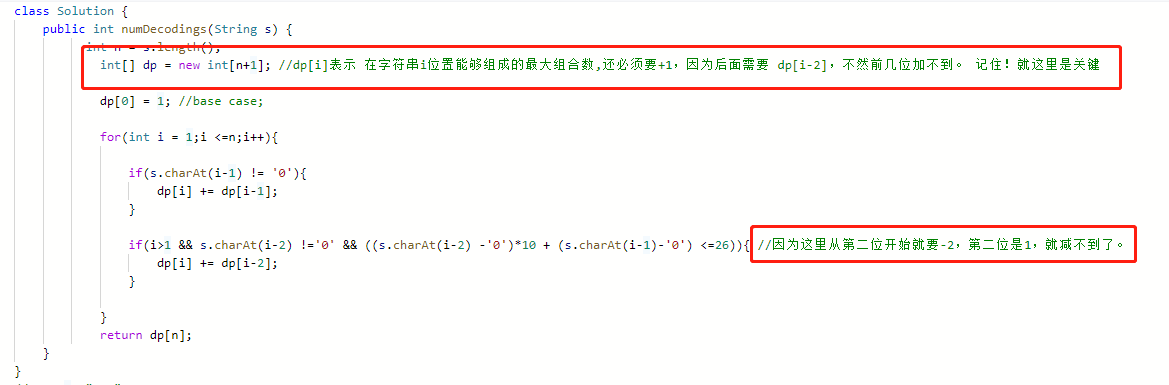


很暴力的解法，计算每个节点到target的距离，找到最小的。

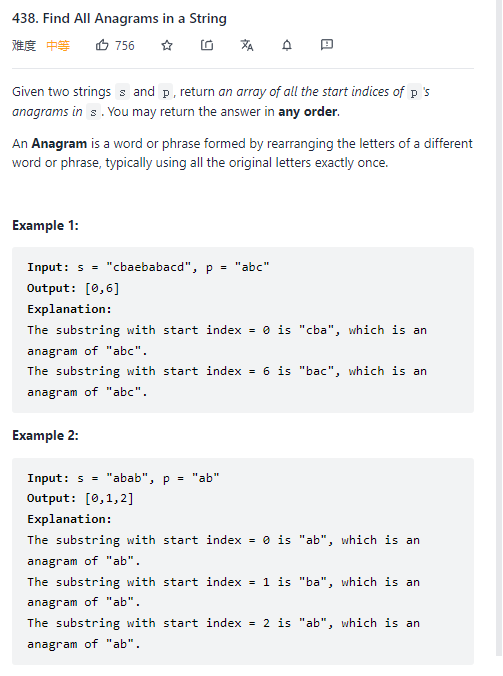


# 需要记忆部分：

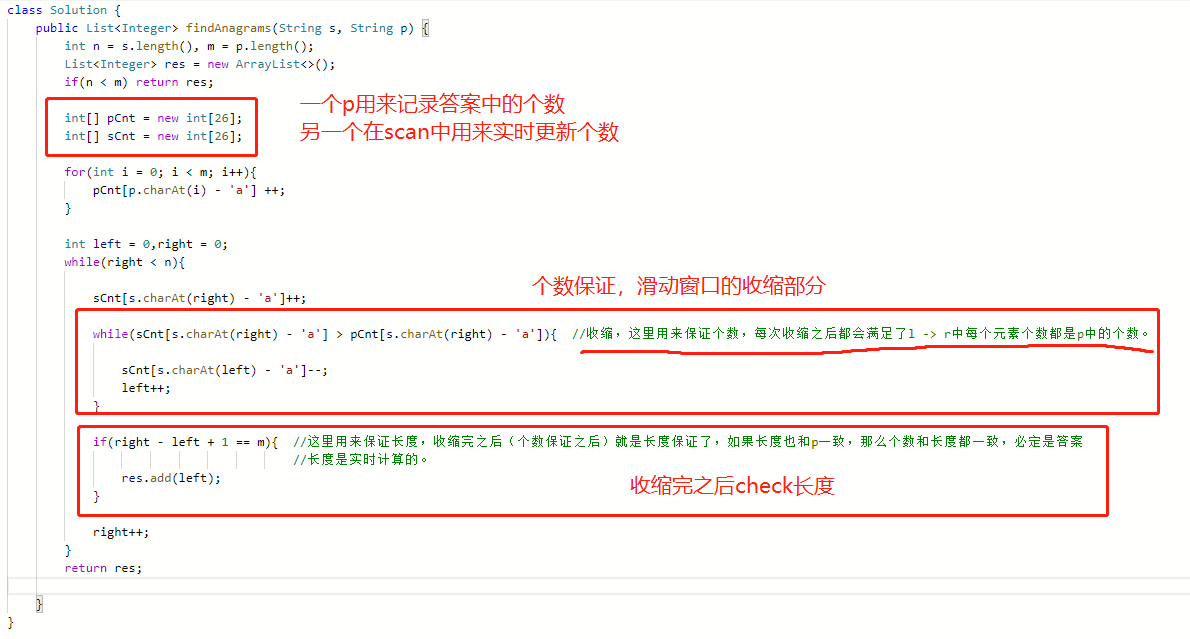


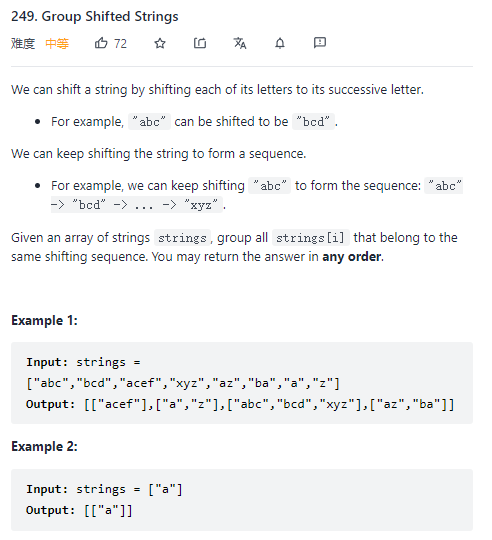


+1.

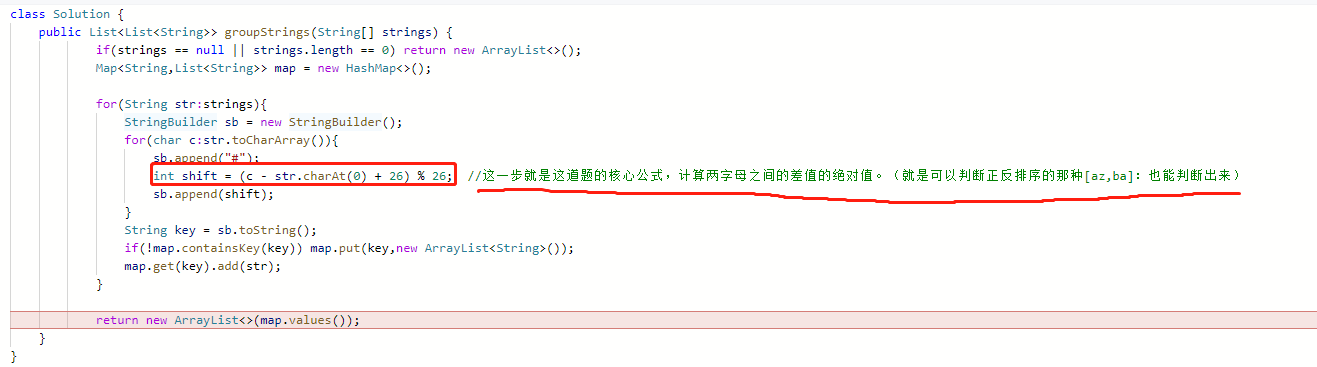


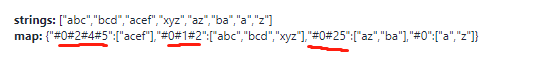
就是 个数保证+长度保证，那么必定是答案。

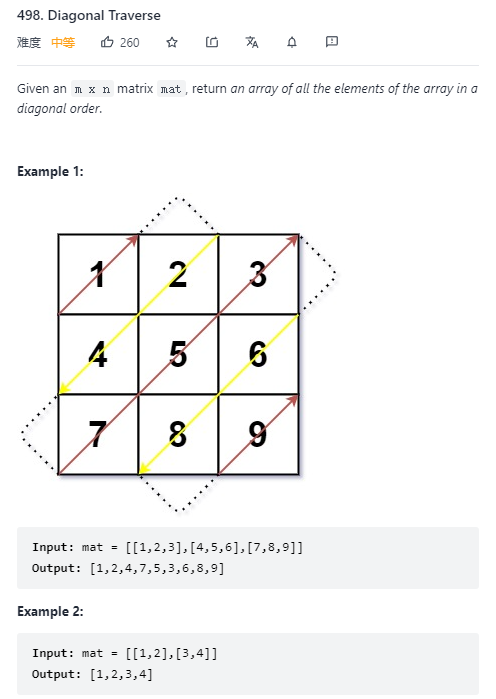




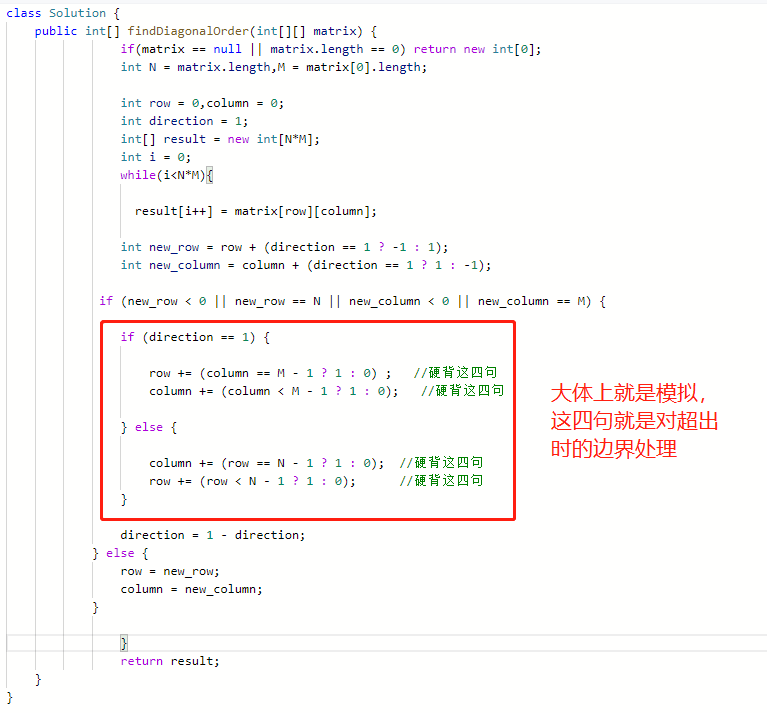
记住那个首字母差值公式（正反顺序都可以判断）



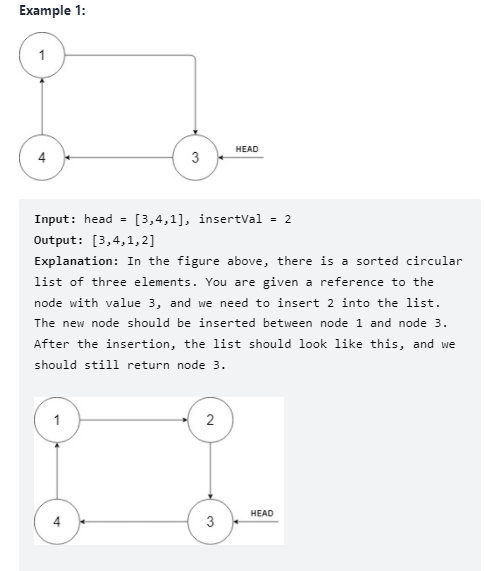




对角线遍历，硬背四句公式:



链表找到元素该插的位置:



这题得理解记忆方法:

双指针两阶段:

1.找起点 2.找插入位置



多写几遍。