**1. Two Sum**

遍历数组，创建一个HashMap每次都判断当前数组索引位置的值在不在hashtable里，不在的话，加入进去，key为数值，value为它的索引值；在的话，取得他的key，记为n（此时n一定小于循环变量i），接下来再在hashtable中查找（target-当前数值）这个数，利用了hashtable中查找元素时间为常数的优势。

**7. Reverse Integer**

注意判断是否溢出：if((newRes-tail)/10!=res) return 0;

**8. String to Integer (atoi)**

题目不难，但是细节很多，容易想不到

1.数字前面有空格 如s=“ 123456”

2.数字前出现了不必要或多于的字符导致数字认证错误，输出0 如s=“ b1234” ，s=“ ++1233” , s=“ +-1121”

3.数字中出现了不必要的字符，返回字符前的数字 如s=“ 12a12” ， s=“ 123 123”

4.数字越界 超过了范围（-2147483648--2147483647) 若超过了负数的 输出-2147483648 超过了正数的输出2147483647

在科普一个知识点，倘若某个数超过了2147483647则会变为负数，反过来一样

**9. Palindrome Number**

compare half of the digits in x, so don't need to deal with overflow.

**13. Roman to Integer**

罗马数字共有七个，即I(1)，V(5)，X(10)，L(50)，C(100)，D(500)，M(1000)。按照下面的规则可以表示任意正整数。

重复数次：一个罗马数字重复几次，就表示这个数的几倍。

右加左减：在一个较大的罗马数字的右边记上一个较小的罗马数字，表示大数字加小数字。在一个较大的数字的左边记上一个较小的罗马数字，表示大数字减小数字。

**14. Longest Common Prefix**

既然是公共子串，那每个字符串肯定都包含有，并且在头部，首先把第一个字符串作为默认最大，然后依次与后边每一个字符串对比，计算所有的最大匹配长度，长度最小的就是。

**19. Remove Nth Node From End of List**

遍历链表得到链表的长度，再将要删除的节点的前一个节点的next指向next.next。注意链表长度为1以及删除最后一个节点的特殊情况。

**20. Valid Parentheses**

要检测输入字符是否满足这个条件，一个非常合适的数据结构是stack，后进先出的特征正好满足检测的需求。在检测的时候，每次检查一个字符，如果是左括号，就入栈，如果是右括号，并且右括号和当前栈顶符号是左右配对的，那么就弹出栈顶并且进行下一次检测，如果不满足上面两种情况，就说明检查到了一个非法字符，返回false。

**21. Merge Two Sorted Lists**

递归。

**24. Swap Nodes in Pairs**

隔一个去交换两个相邻结点，比如1->2->3->4->NULL，我们先通过指针交换1和2，再交换3和4。

或者递归。

**26. Remove Duplicates from Sorted Array**

重新赋值一遍列表前面不重复的元素，剩下的就不用操作。

**27. Remove Element**

把等于指定值的元素替换为末尾的几个元素的值，每替换一次长度减1。

**28. Implement strStr()**

str.indexOf(substr);

**36. Valid Sudoku**

用三个HashSet依次判断每行、每列、每个3\*3方块是否有重复元素。注意每个3\*3方块的索引号。

**38. Count and Say**

题意：n=1时输出字符串1；n=2时，数上次字符串中的数值个数，因为上次字符串有1个1，所以输出11；n=3时，由于上次字符是11，有2个1，所以输出21；n=4时，由于上次字符串是21，有1个2和1个1，所以输出1211。依次类推，写个countAndSay(n)函数返回字符串。编写String say(String s)函数实现一个字符串的读数，并迭代调用。

**53. Maximum Subarray**

参考《数据结构与算法分析Java语言描述》30页图2-8算法。

**58. Length of Last Word**

调用s.trim()去除首尾的空格，字符串长度减去最后一个空格的索引即为最后一个单词的长度。

**66. Plus One**

维护一个进位，对每一位进行加一，然后判断进位，如果有继续到下一位，否则就可以返回了，因为前面不需要计算了。如果到了最高位进位仍然存在，那么我们必须重新new一个数组，然后把第一个为赋成1（因为只是加一操作，其余位一定是0，否则不会进最高位）。

**67. Add Binary**

从最末位逐位相加，注意进位。

**69. Sqrt(x)**

对半查找。注意处理平方结果的取值范围超出int范围的情况。

**70. Climbing Stairs**

一个台阶的方法次数为1次，两个台阶的方法次数为2个。n个台阶的方法可以理解成上n-2个台阶，然后2步直接上最后一步；或者上n-1个台阶，再单独上一步。

公式是S[n] = S[n-1] + S[n-2] S[1] = 1 S[2] = 2

**88. Merge Sorted Array**

方法一：把nums2的元素加到nums1后面，再sort。

方法二：从nums1的m+n-1位开始，依次添加nums1和nums2中最大的元素。

**100. Same Tree**

递归比较。

**101. Symmetric Tree**

通过一个判断两树是否对称的函数isMirror(TreeNode p,TreeNode q)判断左子树和右子树是否对称，isMirror函数中递归判断左右子树的值是否相等，左子树的左子树和右子树的右子树、左子树的右子树和右子树的左子树是否对称。

**102. Binary Tree Level Order Traversal**

对于二叉树的遍历，很自然的想法是用递归。此题的递归函数关键在于除了要传一个表示节点的参数以外，还要传一个表示当前节点层次（level）的参数，这样才能将该节点的值加入该层次对应的list当中去。先序遍历。用到的方法：ArrayList.add(int index,Object ojb)；ArrayList.get(int).add(Integer);

**104. Maximum Depth of Binary Tree**

递归求树的深度。

**107. Binary Tree Level Order Traversal II**

参考102。层序遍历。

**110. Balanced Binary Tree**

Based on height, check left and right node in every recursion to avoid further useless search.

**111. Minimum Depth of Binary Tree**

递归求左右子树的最小深度。

**112. Path Sum**

递归。

**118. Pascal's Triangle**

按定义，每一层的第i个位置，等于上一层第i-1与第i个位置之和。

**119. Pascal's Triangle II**

a(0)=1;a(1) = n; a(k+1) = a(k) \* (n-k)/(k+1)

**121. Best Time to Buy and Sell Stock**

参考《数据结构与算法分析Java语言描述》30页图2-8算法。

**125. Valid Palindrome**

s.replaceAll("[^A-Za-z0-9]", "").toLowerCase()后首尾依次比较。

**141. Linked List Cycle**

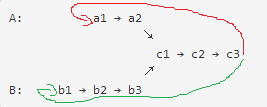
使用快慢引用的思路。两个引用都指向链表头，从链表头开始遍历，慢引用每次前进一步，而快引用每次前进两步，如果链表是有环路的，那么快引用终将追上慢引用；如果没有环路，那么遍历就会有结束的时候。

**155. Min Stack**

当push的数x小于当前最小值时，先将当前最小值压入栈；x在栈顶被弹出后，栈顶的就是x被弹出后的最小值，更新最小值后将该元素也弹出。

**160. Intersection of Two Linked Lists**

We can use two iterations to do that. In the first iteration, we will reset the pointer of one linkedlist to the head of another linkedlist after it reaches the tail node. In the second iteration, we will move two pointers until they points to the same node. Our operations in first iteration will help us counteract the difference. So if two linkedlist intersects, the meeting point in second iteration must be the intersection point. If the two linked lists have no intersection at all, then the meeting pointer in second iteration must be the tail node of both lists, which is null



**165. Compare Version Numbers**

以点号作为分割符，比较相应部分的大小。v1=version1.split("\\.");点号在正则表达式中是通配符，注意转义符。

**168. Excel Sheet Column Title**

本质上就是将一个10进制数转换为一个26进制的数

注意：由于下标从1开始而不是从0开始，因此要减一操作。

**172. Factorial Trailing Zeroes**

求n的阶乘的末尾0的个数：

分解因子, 当且仅当 因子中出现 一对(2,5)时, 最后结果会增加一个trailing zero.

1. 2的个数永远多于5个个数.

2. 计算5的个数时, 最简单的方法是 SUM(N/5^1, N/5^2, N/5^3...)

**190. Reverse Bits**

反转一个32位无符号的整数,但java中没有unsigned int类型。设这个数为k，用一个初值为0的数r保存反转后的结果，用1对k进行求与，其结果与r进行相加，再对k向右进行一位移位，对r向左进行一位移位。直到k的最后一位处理完。>>>表示无符号右移一位。

**191. Number of 1 Bits**

直接调用Integer.bitCount()方法。

**198. House Robber**

维持两个数preYes、preNo, 一个是包含第一个字符的最大值, 一个是不包含第一个字符的最大值， 更新两个值, 最后求 max(preys,preNo)即可。

**202. Happy Number**

递归运算，直至结果为一位数，若该一位数是1或7，则是happy number（7是提交代码时发现的）。

**203. Remove Linked List Elements**

两个指针, 遇到val则删除，否则继续前进，直到链表结尾。

**204. Count Primes**

埃拉托色尼筛法。从1到n范围内的素数表，基本算法思想就是从第一个质数2开始，把2的所有倍数标记为非素数，然后进入到3，把3的所有倍数标记为非素数，然后跳过4（因为4是2的倍数而且已经被标记为非素数了），然后进入到5，把5的所有倍数标记为非素数，循环直至结束。

以上算法还可以继续改进，那就是最外层循环可以只遍历到根号n就行了。因为大于根号n的数要么是素数，要么就有一个小于根号n的因数（而这是不可能发生的，因为如果它有小于根号n的因数的话，那他就已经被筛掉了）。

**205. Isomorphic Strings**

方法一: 建一个map保存映射关系, 同时用一个set保持 被映射的char, 保证同一个char 不会被映射两次。

方法二：The main idea is to store the last seen positions of current (i-th) characters in both strings. If previously stored positions are different then we know that the fact they're occuring in the current i-th position simultaneously is a mistake. We could use a map for storing but as we deal with chars which are basically ints and can be used as indices we can do the whole thing with an array.

**206 Reverse Linked List**

遍历所有节点，把所有节点的next指向前一个节点，返回最后一个节点。

**219. Contains Duplicate II**

利用HashSet或HashMap。（HashSet.add、HashSet.remove）

**225. Implement Stack using Queues**

Stack.push：调用queue.add()将元素加到队尾后，用queue.add(queue.poll())将前面的每个元素依次加到队尾，直至新加进去的元素在队首。

Stack. pop：queue.poll()

Stack.top：queue.peek()

Stack.empty：queue.isEmpty()

**226. Invert Binary Tree**

递归。

**223. Rectangle Area**

两个矩形的面积相加减去重合部分的面积（如果有重合的话）。注意判断是否有重合。

**231. Power of Two**

如果一个数是2的次方数的话，它的二进数必然是最高位为1，其它都为0，那么如果此时我们减1的话，则最高位会降一位，其余为0的位现在都为变为1，那么我们把两数相与，就会得到0。

或者直接调用Integer.bitCount()方法判断其二进制是否只有一位为1。

**232. Implement Queue using Stacks**

用两个stacks实现一个queue。

I have one input stack, onto which I push the incoming elements, and one output stack, from which I peek/pop. I move elements from input stack to output stack when needed, i.e., when I need to peek/pop but the output stack is empty. When that happens, I move all elements from input to output stack, thereby reversing the order so it's the correct order for peek/pop.

The loop in peek does the moving from input to output stack. Each element only ever gets moved like that once, though, and only after we already spent time pushing it, so the overall amortized cost for each operation is O(1).

**234. Palindrome Linked List**

将链表前半段原地翻转，再将前半段、后半段依次比较。

**235. Lowest Common Ancestor of a Binary Search Tree**

利用二叉搜索树左树比父节点小，右树比父节点大的特性。

**237 Delete Node in a Linked List**

从链表中，删除指定的节点，但又没有给出前指针，所以将下一节点的值和next赋予当前节点即可。

**242. Valid Anagram**

哈希表法。变形词的本质是两个单词中，每个字母出现的次数相同，所以我们可以用一个哈希表，记录第一个单词中每个字母的个数，再遍历第二个单词，减去相应的字母出现次数，如果某个字母的计数器不为0了，则说明某个字母出现的次数不一样。这里只用了一个大小为26的数组，是假设只会出现英文字母。

**257. Binary Tree Paths**

用一个函数void bTreePath(String prePath,TreeNode node)递归求每条路径。

**258. Submission Details**

观察1到20的所有的树根：

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 1 2

根据上面的列举，我们可以得出规律，每9个一循环，所有大于9的数的树根都是对9取余，那么对于等于9的数对9取余就是0了，为了得到其本身，而且同样也要对大于9的数适用，我们就用(n-1)%9+1这个表达式来包括所有的情况。

**263. Ugly Number**

将n除以2直至n不能再被2整除，再除以3直至不能被3整除，再除以5直至不能被5整除，返回result==1。

**278. First Bad Version**

由于版本号是从1开始，一直到n，属于增序排列，因此可以采用二分查找的策略，减少比较次数。需要注意的是，在取二分查找的中间值时，不要使用左右相加后再除以2的方式，这样可能会在计算时产生溢出。

**283. Move Zeros**

实际上就是将所有的非0数向前尽可能的压缩，最后把没压缩的那部分全置0就行了。比如103040，先压缩成134，剩余的3为全置为0。过程中需要一个指针记录压缩到的位置。

**290. Word Pattern**

使用str.split(“ “)分割字符串，然后参考205方法一。

**299. Bulls and Cows**

遍历两个字符串，若相同索引的字符相同，则A++；同时将0~9在两个字符串中出现的次数分别记录在两个的数组numS和numG中，遍历这两个数组，将min(numS[i],numG[i])全部相加，减去A即得到B。

**303. Range Sum Query – Immutable**

在构造函数内就计算了从第一个元素到当前元素所有元素的和，保存到数组sums的对应位置中，在函数sumRange中就可以很方便地算出题目中要求的结果了。Java中数组的声明和初始化可以分离。

**326. Power of Three**

方法一：由于输入是int，正数范围是0-231，在此范围中允许的最大的3的次方数为319=1162261467，那么我们只要看这个数能否被n整除即可

方法二：那么如果n是3的倍数，则log3n一定是整数，我们利用换底公式可以写为log3n = log10n / log103，注意这里一定要用10为底数，不能用自然数或者2为底数，否则当n=243时会出错。

**338. Counting Bits**

方法一：调用Integer.bitCount();

方法二：‘1’的个数等于除了最低位之外的‘1’的个数加上最低位‘1’的个数，即count[i] = count[i >> 1] + (i & 1);

**342. Power of Four**

参考326方法二。

**345. Reverse Vowels of a String**

迭代字符串的元素，找到第n个元音字母后，就尝试去找第len-1-n个元音字母，如果找到就交换位置，再进入下一次循环迭代；如果没找到，则退出循环。

**349. Intersection of Two Arrays**

使用哈希Set存入第一个数组的值

遍历第二个数组，如果第二个的数在Set中出现，那么就是交集（与此同时，因为只能返回一个值，所以出现后还需要从Set中删除）。

**350. Intersection of Two Arrays II**

将两个数组排序，用两个索引递增，遍历两个数组进行比较。

**371. Sum of Two Integer**

1、如果不考虑进位，a^b为a+b中各位数字相加的和

2、如果考虑进位，(a&b)<<1为各位数字相加后应有的进位

3、a+b可以转换为其等价形式(a^b)+(a&b)<<1

4、当(a&b)为0时，(a&b)<<1也为0，此时a^b即为所求

5、可以利用递归。

**374. Guess Number Higher or Lower**

折半搜索。

**383. Ransom Note**

问题转化为:ransomNote中的字母是否都是来自于magazine，且magazine中每个字母只能被使用一次。因为题目中说了都是小写字符，所以借助一维数组Num来实现。

1、首先判断特殊情况，如果ransomNote的长度大于magazine的长度，则返回false

2、统计magazine中各个字符出现的次数；

3、依次读ransomNote中的字母i，同时让Num[i-97]--，如果变为负数，则返回false；

4、如果可以读完ransomNote的所有字母，且Num数组中各个元素都大于等于0，则返回true。

“桶”的思想

**387. First Unique Character in a String**

用array[26]来统计26个字母出现的字数，按在s中出现的顺序查询字母的个数，如果等于1则返回该字母在s中的索引。

**389. Find the Difference**

解法一：由于字符串t只比字符串s多了一个字符，那么直接用t中所有字符值的和减去字符串s中字符值的和即可。

解法二：另外一种思路，既然字符串t只比字符串s多了一个字符，也就是说大部分字符都是相同的。那么，我们可以使用异或的方式，来找出这个不同的值。

**396. Rotate Function**

把前后两组求和的系数写出来，对比下会发现，后者跟前者多了sum(A) – n\*A[i]。所以我们计算出F(0)之后，F(1)可以直接推导出来，以此类推，每一个和的计算变成了常数时间。

**400. Nth Digit**

所以给点n，首先确定在几位数之间，如在1000~9999还是在其他之间，然后确定是该区间的哪个数，最后确定是该数字的哪个digit。

**401. Binary Watch**

1,2,4,8都是2的整数倍，每个数都只有1个bit，搜索00：00到11:59所有的解空间，看哪几个数的bit之和等于num。

用到的方法：Integer.bitCount();

规范格式：String.format("%d:%02d",h,m)

**404. Sum of Left Leaves**

检查当前节点的左子节点是否是左子叶，如果是的话，则返回左子叶的值加上对当前结点的右子节点调用递归的结果；如果不是的话，我们对左右子节点分别调用递归函数，返回二者之和。

**405. Convert a Number to Hexadecimal**

如果是负数就将num加16^8再进行下面的运算，注意把num转换成long格式。

**409. Longest Palindrome**

创建一个大小为52的数组，用于存储字母a-z，A-Z的个数为奇数还是偶数。遍历字符串，对于每一个字符，将数组中存储的对应元素&1，这样若最终元素为1则该字母的次数为奇数，若为0则该字母出现的次数为偶数。若最终数组中所有的值均为0，则回文的长度就是字符串的长度；否则，字符串的长度减去数组中所有元素的和，再加1即为回文的长度（回文中心的字母可以是奇数）。

**413. Arithmetic Slices**

找出数组中的等差数列，所有等差数列的子等差数列数目相加。

**419. Battleships in a Board**

由于board中的战舰之间确保有'.'隔开，因此遍历board，若某单元格为'X'，只需判断其左边和上边的相邻单元格是否也是'X'。如果左邻居或者上邻居单元格是'X'，则说明当前单元格是左边或者上边战舰的一部分；否则，令计数器+1

**414. Third Maximum Number**

将nums排序，找出第3大的数，若没有则返回最大的数。

**415. Add Strings**

注意进位，注意处理长短不一样的数字，都从末尾开始相加，最后调用reverse()将结果倒序即可。

**434. Number of Segments in a String**

开头第一个字符不为空格则count++。遍历字符串，遇空格后不是空格的情况则count++。

**437. Path Sum III**

这个想法类似于Two sum，使用HashMap来存储（key：前缀sum，value：多少种方式获得这个前缀sum），并且每当到达一个节点，我们检查前缀sum-target是否存在于hashmap中 ，如果是的话，我们将前缀sum-target的方式加到res中。

例如：在一条路径中，我们有1,2，-1，-1,2，那么前缀和将是：1,3,2,1,3，假设我们想找到目标sum为2，那么我们 将具有{2}，{1,2，-1}和{2，-1，-1,2}方式。

使用全局变量计数，但显然我们可以通过从底部向上传递计数来避免全局变量。 时间复杂度为O（n）。

**438. Find All Anagrams in a String**

从S中找到一个子串，这个子串变化后可以得到P（其实就是这个子串的26字母的分布和P一样）。直接滑动窗口（长度为P的大小），检查S每一个子串的分布情况和P是否一样。

**441. Arranging Coins**

1+2+3+…+x<n,即(1+x)\*x/2=n,所以x=(int)(Math.sqrt(1+8\*n)-1)/2,为防止数据溢出，修改为x=(int)(4\*Math.sqrt(0.125+0.5\*n)-1)/2。

**442. Find All Duplicates in an Array**

把element当做下一个位置坐标。遍历nums，记当前数字为n（取绝对值），将数字n视为下标（因为a[i]∈[1, n]）。当n首次出现时，nums[n - 1]乘以-1。当n再次出现时，则nums[n - 1]一定＜0，将n加入答案

**447. Number of Boomerangs**

i遍历所有点

每个i同时再次遍历所有点，记录下i-j的距离

这一轮中此时如果之前遍历过的点（k）有距离和i-j一样的，那么数量+2（j k可以互换）

**448. Find All Numbers Disappeared in an Array**

在Top Solution中，有网友分享了一种很奇妙的解法—取负法。含义是：将元素对应的位置取负。简单一句话可能不好理解，我们举个例子。假设在位置k放了元素i，则在取负的过程中i的取值有两种可能：为正，表示当前尚未遇到元素k将该位置取负；为负，表示当前已经有元素k出现，并将元素取负。但是我们不关心k，我们关心元素i。元素i既然出现，我们就看一下位置i：为正，表示这是元素i第一次出现，我们将位置i取负；为负，表示元素i已经出现过一次，我们不做任何操作。不管一个元素出现一次还是两次，只要出现它对应的位置就会被取负。当某个元素不出现的时候，该元素对应的位置始终访问不到，所以还是正值，通过这种方法我们就可以找到哪些元素没有出现。

通过上面的分析我们也很容易知道，在取负的过程中，如果发现要取负的位置已经为负，说明这个元素已经出现过，也即该元素出现了两次，我们可以将该元素保留下来。

**453. Minimum Moves to Equal Array Elements**

每次将数组中的n-1个数字加1，相当于将剩余的一个数字减1。所以只需找到数组中的最小值m，计算m与数组中其他数字差的累计和即可。

**455. Assign Cookies**

首先将两个数组先排序。遍历期望尺寸的数组和饼干尺寸的数组，如果饼干尺寸符合则两个数组都向前进一，表示有一个饼干满足了一个小孩；如果饼干尺寸不符合，则饼干数组向前进一，尝试下一个饼干。

**459. Repeated Substring Pattern**

直接每次选择一个可以被整除的较小的数，截取开头的那几个字符，然后重复到原长度验证就好

**461. Hamming Distance**

返回Integer.bitCount(x^y)即可。

**463. Island Perimeter**

方法1:遍历数组找到所有为1的块，判断其周边的情况增加周长。

方法2:每有一个方块边就增加4条，每有一个相邻的方块，边就减少两条，判断好边界就没问题了。

**475. Heaters**

将两个数组分别排序，用两个指针递增比较。

**513. Find Bottom Left Tree Value**

层序遍历，用一个queue保存每一层的节点，循环将队列第一个节点的左右子树加入队列并弹出第一个节点直至队列为空，保存每层的第一个值为result，最后的即为所求值。

注意int的取值范围。

|  |  |  |
| --- | --- | --- |
| **Date** | **Count** | **No.** |
| **12.16ago** | 32 | / |
| **12.17** | 1 | 447 |
| **12.18** | 0 | / |
| **12.19** | 0 | / |
| **12.20** | 6 | 415、405、459、326、434、202 |
| **12.21** | 4 | 121、437、70、263 |
| **12.22** | 6 | 235、21、342、27、345、66 |
| **12.23** | 7 | 198、24、441、118、119、172、112 |
| **12.24** | 1 | 299 |
| **12.25** | 0 | / |
| **12.26** | 4 | 26、9、19、438 |
| **12.27** | 3 | 205、290、58 |
| **12.28** | 6 | 190、28、414、1、7、8 |
| **12.29** | 3 | 165、374、20 |
| **12.30** | 3 | 14、67、88 |
| **12.31** | 0 | / |
| **1.1** | 0 | / |
| **1.2** | 0 | / |
| **1.3** | 0 | / |
| **1.4** | 3 | 448、461、203 |
| **1.5** | 3 | 102、107、101 |
| **1.6** | 3 | 110、141、111 |
| **1.7** | 0 | / |
| **1.8** | 0 | / |
| **1.9** | 1 | 160 |
| **1.10** | 3 | 232、155、257 |
| **1.11** | 3 | 36、38、223 |
| **1.12** | 3 | 234、219、225 |
| **1.13** | 0 | / |
| **1.14** | 0 | / |
| **1.15** | 0 | / |
| **1.16** | 2 | 400、396 |
| **2.16** | 3 | 53、475、69 |
| **2.17** | 3 | 303、204、125 |
| **2.18** | 0 | / |
| **2.19** | 0 | / |
| **2.20** | 2 | 278、189 |
| **2.21** | 2 | 419、338 |
| **2.22** | 2 | 413、442 |
| **3.8** | 1 | 513 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **number** | **done** | **not done** |
| **Easy** | 105 | 105 | 0 |
| **Medium** | 176 | 3 | 173 |
| **Hard** | 77 | 0 | 77 |
| **Total** | 358 | 108 | 250 |

448->452->455->482