leetcode-Easy/Midum(start from easy)

from easy ac 这个有点像流水账，前面还是知识点总结，后面题目不会的多了，就加入了大部分的代码，回头还是做成gitbook这种带个目录的比较方便，不然一长篇不太方便阅读。

####412. fizzbuzz 最简单的数学题，注意if判断顺序或者就都单列出来总不会错

####292. nim game 只要石头堆对4取余有剩余，剩余的就是我的，就能赢

####371. sum Two Integer sum 得到按位异或的结果，在没有进位的情况下就是我们想要的和；如果有进位，进入while循环处理进位 code

####258. add Digits (num-1)%9+1

389-383-387-242

都是很类似题 第一种解法：哈希表（key：character，value：次数或者index）389findDifference：第一次遍历向hashmap中存string1，key-character，value-出现次数，第二次遍历key出现1次value-1，value<0说明是different的char 387UniqueCharacter：第一次遍历向hashmap中存string1，key-character，value-出现次数，第二次通过key-char找最小indexs.indexOf() 383ransomNote：和finddifference方法（hashmap）一样，只不过string的位置换了一下，谁包含谁换了

第二种解法：char转成int，来做index 389findDifference：要求的结果变量re先依次减去第一个string中每个char转成int的值，然后再依次加上第二个string中的每个char，得到正确difference（运行时间比hashmap快了很多）387UniqueCharacter：char转成int当做array的索引，没出现依次，该索引位置对应值++，取所有索引位置对应值=1中最小索引，对应char 242validAnagram：考虑到两个string可能长度不一样，用普通for容易出现outofIndex，for in形式比较好for(char j: t.toCharArray())

##substring & anagrams(上面242，共5) ####49. Group Anagrams 自己的思路是循环拼接，判断是不是anagram以前做过，可以用char值的和的方式，然而不行，不能避免重复匹配的情况。 discuss区不知道咋想到的排单个string的字符序，总之有可行的方法了. 这道题代码比较好读懂，也很简短，思路大体是：

1. 把每个string先按字符序重新排列一下，（anagrams都会得到一样的结果）
2. 把第一次排好的string添加到map中当key，这样后面的anagrams可以通过判断key是否存在的方法找到
3. anagrams 都存在了 key是自己字母升序排列 的 value中组成list。
4. 输出list（每个value本身就是list）

```
public List<List<String>> groupAnagrams(String[] strs) {
    HashMap<String, List<String>> hm = new HashMap<>();
    for(int i = 0; i < strs.length; i++){
        String currString = strs[i];
        char[] charArr = strs[i].toCharArray();
        Arrays.sort(charArr);
        String sortString = new String(charArr);
        if (!hm.containsKey(sortString)){
            hm.put(sortString, new ArrayList<String>());
        }
        hm.get(sortString).add(currString);
    }
    return new ArrayList<>(hm.values());
}
```

####438. Find All Anagrams in a String Here is a 10-line template that can solve most 'substring' problems discuss里找了个高大上的方法，只能看懂，并不能想到，下次但愿能背会。。。还有自己想的一个方法，最直接但是很慢，反正ac了

####14. Longest Common Prefix 每种语言string的各种操作都很风骚，这个题用了两个string的方法，见下面，思路就是取strs[0]当做pre，去比较看是不是strs[1-->len]的prefix，所以需要两层循环，外层遍历数组中每个元素，内层不停

剪短pre直到pre是strs[i]的prefix，内层：while(strs[i].indexOf(pre) != 0){pre = pre.substring(0,pre.length()-1);}

```
/*string.indexOf(String str):
        Returns the index within this string of the first
occurrence of the specified substring.
        If no such value of k exists, then -1 is returned.
      public String substring(int beginIndex,int endIndex)
        Returns a new string that is a substring of this
string.
        The substring begins at the specified beginIndex
and extends to the character at index endIndex - 1.
        Thus the length of the substring is endIndex-
beginIndex.
        --JAVA API
*/
```

####459. Repeated Substring Pattern

1. substring的长度肯定是str的约数(str%substring==0),
2. 遍历所有可能的substring(outer for),可以从str.length()/2开始，到1
3. check(inner for): 从头-依次-取长度为str约数的substring，然后通过叠加(str.length()/substring.length())个substring，看结果是不是str

####3. Longest Substring Without Repeating Characters 思路都差不多，实现方式不同，有用hashset，hashmap和array三种，代码文件中详细都有，这里只写最简单的一种，set实现的 start, end means start of set and end of set. we always keep a substing without repeating characters in the set, the substring's length may change, so the set.size() also changes, we keep track of this size to find the max size.

```
    public int lengthOfLongestSubstring(String s) {
        int start = 0, end = 0, max = 0;
        Set<Character> set = new HashSet<>();
```

```
        while (end < s.length()) {
            if (!set.contains(s.charAt(end))) {
                set.add(s.charAt(end));
                end++;
                max = Math.max(max, set.size());
            } else {
                set.remove(s.charAt(start));
                start++;
            }
        }
        return max;
    }
```

####28. Implement strStr() 看起来是个很简单的题，花了很长时间，心塞. 自己是用了类似双指针的方法，haystack中有needle首字符时，记index of haystack，然后比对needle，haystack不够长就直接返回-1，如果重合元素==needle.length,说明是包含的，返回此时记的index。另外一种大牛的方法看起来很简介，但是有一句j+i不太懂，考试略忙也没有仔细想，mark一下回头看

//postscript 又遇到了一次这个题，还是花了很久，最后还是看了之前这里的答案。。。心塞，再细细记一下思路：

1. 边界值，如果needle==“”那么所有的haystack都可以包含，needle出现在haystack第0位，所以直接返回0；如果target或haystack为null，haystack==“”,那么一定是返回-1的。
2. 实现，
    1. 首先要遍历haystack。可以用for循环；
    2. 之后从每个h开始，看是否对应在needle中，这里为了标记 相同元素的起点，所以把h值存在tmp中，一会儿操作tmp，还有每次新的h都要从头在needle找对应；
    3. 下一步是如果n==tmp了，注意这里用while，只要n==tmp，两者一起自增，直到不相等，
    4. 这个while中有三个操作，第一，如果n==needle.length()-1 说明全都匹配上了，返回之前保存的h即可；第二，如果

tmp>haystack.length()-1，说明needle中出现了更多haystack中没有的元素，肯定返回-1；第三，正在朝着1，2两种情况进行。

5. 还需要注意这三种情况的顺序，先判断 n 中元素遍历完了没，完了就可以返回了。

```
for(int n = 0;n<haystack.length();n++){
    int i = n;
    int j = 0;
    while(haystack.charAt(i)==needle.charAt(j)){
        if(j==(needle.length()-1)) return n;
        j++;
        i++;
        if(i>haystack.length()-1) return -1;
    }
}
```

####lintcode strStr2

####283. move Zeroes 设置计数器，从计数器=0开始放非零数字，剩余（length-计数器）置0

####349.Intersection 嵌套for-loop找相同元素，把未出现过的相同元素存入ArrayListre.contains();re.add(); ArrayList再转回int[]很麻烦for(int i : IntegerList)

####350. Intersection of Two Arrays II 现在用的是类似指针的思路，用while循环（避免for循环会出现的各种index问题）先排序（一般用指针都要排序），然后就一步一步往后挪，如果上下相等了，那就一起后移，不相等就小的后移，找大数来跟另外一个数组中的数匹配 更好的方法应该是hashmap吧

####171. excelColumnNumber re+=(sc[i]-'A'+1)*Math.pow(26,sc.length-i-1);

####168. Excel Sheet Column Title n%26得到0-25之间的26个数，刚好对应A-Z,但题目中given int是从1-A开始，所以只需要把given int n=n-1 后面就可以

按照0-25 分别+65去转换对应的asc码就可以了

```
while(n>0){
    n=n-1;
    re = Character.toString((char)(n%26+65))+re;
    n/=26;
}
```

discuss推送了一个one line code，用了递归，很厉害

```
return n == 0 ? "" : convertToTitle(--n / 26) + (char)('A'
+ (n % 26));
```

####169. Majority Element 有种取巧的写法，可以说得通但是想不到，传统解法还是hashmap再mark一下这个for(Integer k: m.keySet())

####217. Contains Duplicate 用set是比较简单的方法，通过set.contains的方法来判断，for in 来遍历nums

####219. Contains Duplicate II 刷过一边的题果然ac率大大大幅提升，用hashmap，value存index以便判断distance<k

####455 assign cookies 跟下面一题的思路一样，用while循环，把两个数组排序，第二个数组中对应元素不小于第一个数组中对应元素，output++，两个元素都向后移，如果第二个数组中元素比第一个数组中对应元素小，说明饼干不够，要往后找更多饼干，所以第二个数组j++

####350. Intersection of Two Arrays II 现在用的是类似指针的思路，用while循环（避免for循环会出现的各种index问题）先排序（一般用指针都要排序），然后就一步一步往后挪，如果上下相等了，那就一起后移，不相等就小的后移，找大数来跟另外一个数组中的数匹配 更好的方法应该是hashmap吧

####401 binary watch 强行倒着解，把所有可能的小时和分钟列出来，那一时刻刚好对应二进制的1的个数跟num 相等，就输出这个时间，性能不好，也不太好想但好写的一种方法 标准写法应该是backtracking

####13 roman to int

1. 这个题对我来说的难点好像在于for里判断的时候要i+1个元素个i个元素比较，所以for循环很难遍历全整个数组，方法是：会一开始的时候就把遍历不到，而且刚好也需要加上的最后一位直接赋值给result
2. 最近总是想到用true fale这种方式判断，并不简单，而且切记boolean这个值是需要不停改变的，for循环里最后一句重新minus=false忘记加找了半天问题
3. 另外这个题的超时问题也是因为打印语句

####453 minMove

1. 自己的思路是按照题目里说的，数组里除了最大元素都++，然后判断符不符合标准，复杂度太高，不能accepted
2. 大神的思路，把上面那种 除最大值外都+1 做n次这种操作达到数组元素全部相等 转换成 每个非min的元素每次-1需要的操作数

```
for(int n:nums){
    re+=(n-min);
}
```

####462. Minimum Moves to Equal Array Elements II 跟上面453思路类似,都是先排序，因为这个是小的数可以+1，大的数可以-1，所以排序后是两边都可以操作，每次操作次数等于两边差值

```
while(i < j){
    count += nums[j]-nums[i];
    i++;
    j--;
}
```

####447. Number of Boomerangs

1. 暴力解，取每个点当基准点并给他一个hashmap存距离，看有没有两点到他距离相等，

2. 注意两点，首先是map每次外层for都新建一个，然后是count，到基准点距离相等的两点可以互换，所以又一次匹配就有两种形式

####415 Add String

1. string1.length()>string2.length()int x=len1<0?0:num1.charAt(len1)-'0';int y=len2<0?0:num2.charAt(len2)-'0';
2. carry

####463. Island Perimeter

1. 首先想到的思路：每个0格上下左右是否有1，有的话计数器+1，格子是否在周围一圈上的位置，如果是，计数器+1，这个方法一开始超时跑不过，以为写错了，后来发现超时的原因是因为有system.out.println的打印语句。。。
2. 直接用1的格子，每个格子四条边，每和一个左边的1（遍历过的1）相邻，两个格子各少1，所以是-2
3. 程序开始的if判断，没有作用，可以表示一下思路吧，没实质性作用

####384. Shuffle an Arrays

1. 自己想的，生成随机数来做索引，为了避免随机生成了相同的数，把生成过的索引存为map的key，这个方法一开始也是超时跑不过，以为写错了，后来发现超时的原因是因为有system.out.println的打印语句。。。
2. 大神的思路：从0-i 之间选一个随机位置，把original[i]插入，把原位置的值放在后面（也就是shuffled[i]）位置，存着，直到把所有original中每个值都插入一遍 （想不到）

####419. Battleships in a Board 有一点点类似上面那个island perimeter，题目意思其实是以battleships的左上角的X来代表一个battleship，所以算有几个X是top-left

####413. Arithmetic Slices 若序列S为等差数列，其长度为N，则其等差数列切片的个数SUM = 1 + 2 + ... + (N - 2), 例如，等差数列[1, 2, 3, 4, 5, 6]的切片个数为1+2+3+4 = 10if(A[i+1]-A[i]==A[i+2]-A[i+1]) cur += 1;

####434. Number of Segments in a String 这类题都不要变换思路，要求什么

就用什么，求有几个words，就算有几个words，把判定情况写全，这里是两种情况，

1. 第一个words，i==0 && charAt(i)!=' ';
2. 后面的words，charAt(i-1)==' ' && charAt(i)!=' ';

####191. Number of 1 Bits 还是看到就头疼的题，估计再看点这类题就会好点了，，，

1. 方法1：比较直接，判断最右1位是不是1，n&1 如果是，计数器+1，右移，判断下一位，这里有两种代码写法，前面这种比后面的快1倍count = count + (n & 1);,if((n&1)==1) count++;
2. 方法2：用到了类似power of two中n&(n-1)的方法,n & (n-1)!=0说明n不是2的power，肯定还有不止一个1，但为什么n=n & (n-1)不太懂

```
while(n != 0){
    n = n & (n-1);
    count++;
}
```

####405. Convert a Number to Hexadecimal

1. 首先负数这里需要在去掉符号之后-1，-1-->0，-2-->1以此类推，因为-1='ffffffff',然后-2='fffffffe'
2. 之后是num % 16<6 用abcdef剩下用0-9.其他位（前面的位）保留f

####441. Arranging Coins 也是一个很简单的题，给n个硬币，在第k行放k个硬币，能放到第几行，不完整的一行不算 用了while循环，略慢

####66. Plus One 从最末位开始，+1，如果<=9直接返回，>9继续下一位+1，如果整个数（数组）遍历完都没返回，说明发生了这种情况9+1=10，需要在最前面多加1位

####172. Factorial Trailing Zeroes 查了一些题解，刚看到的时候比较懵逼，题解基本都是一个思路：0是2*5得到的，也就是我们需要知道有多少2，5 pair，2个数肯定会比5多很多，所以只需要知道5的个数，又因为25这个特例，意思大概

是，每个25都可以再得到两个5 去跟2凑pair，所以25看似是5-10-15-20这4个，但25自身还有两个五，所以可凑6pairs。如果能明白题意，并且可以分析题目得到这一步，那代码就很简单了

```
 while(n>0){
    re+=n/5;
    n/=5;
}
```

####344.reverse string 这个题有两种方法，好想的是：转成数组s.toCharArray()转回stringString.valueOf(charArrayName)

第二种方法用了类似上面的思想，从前后同时遍历，可以做到in place 空间复杂度更低一点

```
public String reverseString(String s) {
    char[] str = s.toCharArray();
    int start = 0, end = str.length-1;
    for (int i = start, j = end; i < j; i++, j--) {
        char temp = str[i];
        str[i] = str[j];
        str[j] = temp;
    }
    return String.valueOf(str);
}
```

####36. Valid Sudoku 横竖的判断就是ij互换一下，需要哪个坐标变就把内层循环的int放过去，比如列的时候需要x坐标变，就把内层循环的j放过去

横竖都好判断，每个cube不好判断，坐标不好想，discuss里发现了一种很好的方法.类似用一次循环画一个cube或者二维数组的意思，i/3=x,i%3=y,x,y标明一个点

```
if(board[3*(i/3) + j/3][3*(i%3) + j%3]!='.' &&
!cube.add(board[3*(i/3) + j/3][3*(i%3)  + j%3])) return
```

```
false;
```

####37. Sudoku Solver 介个题是hard，是的，意思是不会做，再看看答案，用递归写的，更觉得不会做很正常了。。。什么时候才能会做递归啊。。。

1. 最后那个isValid方法其实跟上题基本是一样的
2. 如果该位置非空，不用填数，continue即可
3. 确定一个非空方格后，从1-9，尝试填入，每次填入都判断填入当前数后是否valid，如果当前填入valid，继续向后填，如果每一格都试到了合适的数字，返回solution，
4. 如果填到最后发现无解，这里最后指的是，填1-9的循环结束，1-9均尝试过都不行，返回false，说明题目无解

```
public class Solution {
        public void solveSudoku(char[][] board) {
        doSolve(board, 0, 0);
    }
    //也可以每次都只传board进去，row col都从零还是判断
    private boolean doSolve(char[][] board, int row, int
col) {
        for (int i = row; i < 9; i++) { // note: must
reset col here!
            col = 0;
            for (int j = col; j < 9; j++) {
                if (board[i][j] != '.') continue;
                for (char num = '1'; num <= '9'; num++) {
                    if (isValid(board, i, j, num)) {
                        board[i][j] = num;
                        if (doSolve(board, i, j + 1))
                            return true;
                            //else 可以不加，因为if 里 return
了，但最好加上
                        else board[i][j] = '.';
                    }
```

```
                }
                return false;
            }
        }
        return true;
    }

    private boolean isValid(char[][] board, int row, int
col, char c){
        for(int i = 0; i < 9; i++) {
            if(board[i][col] != '.' && board[i][col] == c)
return false; //check row
            if(board[row][i] != '.' && board[row][i] == c)
return false; //check column
            if(board[3 * (row / 3) + i / 3][ 3 * (col / 3)
+ i % 3] != '.' && board[3 * (row / 3) + i / 3][3 * (col /
3) + i % 3] == c) return false; //check 3*3 block
        }
        return true;
    }
}
```

##hashmap+string 205

290

####205. Isomorphic Strings 论坛里看到的解题思路，感觉现在不适合刷题，想到hashmap但是不知道isomorphic的两个词到底什么关系

思路：本题的核心点在于，如果不是isomorphic strings，s相同的char在t中会对应不同的char，t中相同的char会对应不同的char。check两者相互对应的关系，不难想到2个hashmap。我这里的做法是1个hashmap和1个set，在遍历s,t的过程中，以s的char做key，t对应的char做value，如果相同key出现了2个value，必然会错。同时set记录t已经用过的char，如果不同的key使用相同的char，必定也不是。（本题易忽略的就是后者） for(int i : hashmap.values()){...}

注意，hm.put()这个方法，官方api写的是：the previous value associated with key, or null if there was no mapping for key. discuss里找到的代码，实现方法很简单，但是含义丰富，所以贴全部代码，又多了一道要背的题

```
 public boolean isIsomorphic(String s, String t) {
    Map<Character, Integer> m1 = new HashMap<>();
    Map<Character, Integer> m2 = new HashMap<>();
    for(Integer i = 0; i < s.length(); i++) {
        //hm.put() returns  it's the previous value
associated to this key

System.out.println("i:"+i+"-1:"+m1.put(s.charAt(i),
i)+"-2:"+m2.put(t.charAt(i), i));
        if(m1.put(s.charAt(i), i) != m2.put(t.charAt(i),
i)) {
            return false;
        }
    }
    return true;
}
/*
注意，hm.put()这个方法，官方api写的是：the previous value
associated with key, or null if there was no mapping for
key.
test case:
    "aba" "baa"
stdout:
    i:0-1:null-2:null
    i:1-1:null-2:null
    i:2-1:0-2:1
根据输出可以看出，如果映射已经存在，会输出对应的value，在这里就是以
char为key对应的value（index）
*/
```

####290. Word Pattern 基本就是复用了上面的代码 注意两点：

1. 如果两者长度不同，pattern肯定不同，所以直接返回
2. for循环是通用的核心代码，就是看映射关系是否一致，比如pattern存入map的时候，如果key已存在，会返回key对应的.value，即pattern中key-Character对应的value-index；同理parts存入map时，如果key-parts[i]已存在，会返回对应value-i

####118. Pascal's Triangle 又是杨辉三角，这种题看要求返回的类型立刻想到用嵌套的list， 实现的时候需要注意第一行自己放好，后面每行的第一个和最后一个肯定都是1，其他没了 注意用list.add，核心就这句inner.add(tri.get(i-1).get(j-1)+tri.get(i-1).get(j));

####119. Pascal's Triangle II 改了一下上面的题（两处：1. 外层for i<rowIndex+1; 2. 返回tri.(rowIndex);），秒ac，不过应该有更简单的方法

```
List<Integer> res = new ArrayList<Integer>();
for(int i = 0;i<rowIndex+1;i++) {
    res.add(1);
    for(int j=i-1;j>0;j--) {
        res.set(j, res.get(j-1)+res.get(j));
    }
}
```

####400. Nth digit 表示不会做，想了一下但是没想到怎么实现，有一个稍微好理解一点的例子，For example given n is 1000, we first -9（9个数，每个数len=1） and then -180（90个数，每个数len=2）. The left is 811. The number is 100+810/3=370.（注意这里是100+，前面都是-99，所以这里811-1） The digit is the (810%3=0)th. Therefore, the digit is 3

####58. Length of Last Word 一句话ac return s.trim().length()-s.trim().lastIndexOf(" ")-1;

####299. Bulls and Cows 算cows的时候用了创建新的长度=10的数组来表示每个string中数字出现情况, secret+1， guess-1 判断secret中ith char是否<0,如

果小了说明guess中猜对了ith char，b++，就是cow++

```
if (numbers[secret.charAt(i)-'0'] <0) b++;
    numbers[secret.charAt(i)-'0']++;
if (numbers[guess.charAt(i)-'0'] >0) b++;
    numbers[guess.charAt(i)-'0']--;
```

####223. Rectangle Area 需要判断一下overlap的各个坐标

```
int left = Math.max(A,E), right = Math.max(Math.min(C,G),
left);
int bottom = Math.max(B,F), top = Math.max(Math.min(D,H),
bottom);
return (C-A)*(D-B)+(G-E)*(H-F)-(right-left)*(top-bottom);
```

####204 count primes Count the number of prime numbers less than a non-negative number, n. 和61B例子不同 根据hint采用这个方法：Sieve of Eratosthenes-ˌer-ə-'täs-thə-ˌnēz
遍历的时候是i*i<n, j+=i; 相当于遍历所有i的倍数

####189.Rotate Array

1. 数组直接赋值是浅拷贝，int[] a = ...; int[] b = a;这种情况下，a改了b也会改动！！所以用array.clone();这个方法
2. k>nums.length的时候需要处理的，明显是%
3. k=0的时候不需要操作，所以操作前判断一下

####396. Rotate Function 笨方法就是按照题目给的强行算，自己能写出来但是跑巨慢，大神的方法在这里, 一句两句的也说不清楚，算法太神奇了

####414. Third Maximum Number 看起来越是简单的题坑越多，按发现问题的顺序

1. 最大值赋给first后，原first值变为second，注意别丢了
2. 重复元素不计数，所以遇到重复元素直接跳过，不然影响结果

3. 根据测试用例来看，需要long,其实就是处理数据极值的问题，大部分题都要考虑这点

####303. Range Sum Query - Immutable 这个也是可以自己写的，就是写出来的比较慢，有个蜜汁答案，想的很新奇，复杂度很低，很好懂，太牛了。思路是，在构造方法中直接创建一个数组，每个元素=该元素和之前所有元素和，之后调用sumRange的时候就返回类属性这个数组的i j 元素差就可以了，棒

####67. Add binary 从末尾起遍历两个string，

1. 如果当前计数>长度，该位为1
2. sum= A当前位+B当前位+carry（可能=1 or 0）+之前求得的sum，拼接在后面 (curIntA+curIntB+c)%2==curIntA ^ curIntB ^ c
3. 计算carry位

####461. Hamming Distance 字符串中不相同的字符有几个，hamming distance就是几，题目看起来好像很难很陌生的样子，其实看懂例子之后还挺简单的。居然自己写的一遍过，真真的一遍过了都没语法错误。 思路：题目的意思就是给两个int，这两个int的二进制表示方式中有几位是不一样的 很简单，%2之后作比较是否一样，然后这两个数/=2往下一位算

####6. ZigZag Conversio discuss中看到的方法，思路是给rownumber， 然后就在row1放一个char，row=row+1放一个，+1和-1分别对应两种情况，其实可以理解为拐弯，比如row=0的时候，说明要往下走，row=row+1，row=rownumber-1的时候说明要往上往回走了，所以row=row-1。高端的不行

####448. Find All Numbers Disappeared in an Array 但愿是easy的最后一题了，总觉得easy要刷完了，结果就会出一道新题。。。

这个也是看起来简单，一开始的思路非常奇葩，看数组中的数+1和-1的数是不是存在，如果不存在就标记一下大概这样，实现了一下发现不行，应该是标记的问题，后来在discuss看到了一个不错的方法：

> The basic idea is that we iterate through the input array and mark elements as negative using nums[nums[i] -1] = -nums[nums[i]-1]. In this way all the numbers that we have seen will be marked as

> negative. In the second iteration, if a value is not marked as negative, it implies we have never seen that index before, so just add it to the return list.

另外这个题有点bug，原题给的example是：

> Input: [4,3,2,7,8,2,3,1] Output: [5,6] 我看到discuss的代码的时候会想，what if nums[i]>nums.length，我就改动了下原有example,删掉了后面的2，3，1，expected answer也是报错的，但是，注意审题”Given an array of integers where 1 ≤ a[i] ≤ n (n = size of array), some elements appear twice and others appear once.“

####190. Reverse Bits result左移，空出一位放n的最右位，n右移，看见位运算还是懵逼，心塞

####atoi

1. 清除空格的方法str = str.trim();
2. 一位一位加int digit = str.charAt(i) - 48;sum = sum*10-digit;

####165 compare version 长度不相等的时候很麻烦:

1. 注意以 . perioud 为分隔符时候的写法version1.split("[.]");version2.split("\\.");
2. 注意每次while中才给temp赋值，即，如果s1比s2多一位，那s2的当前temp=0，然后判断
3. time complexity: O(m+n)，m和n分别是两个字符串的长度。

####7 reverse integer

1. int不行，long才可以,int放不下
2. 不需要取绝对值单独判断符号，%10的时候得到的结果是带-的
3. y是long，但需要的结果是int，所以需要判断y能不能转int，就是在最大值和最小值之间

####53. Maximum Subarray

1. 有种greedy的方法，不懂为什么sum=max(sum,0);

```
public int maxSubArray(int[] nums) {
    if(nums==null || nums.length==0) return 0;
    int max = Integer.MIN_VALUE, sum = 0;
    for(int i : nums){
        sum += i;
        max = Math.max(max, sum);
        sum = Math.max(sum,0);
    }
    return max;
}
```

2. 这个方法比较好懂一点

```
public int maxSubArray(int[] A) {
    int max = Integer.MIN_VALUE, sum = 0;
    for (int i = 0; i < A.length; i++) {
        if (sum < 0) sum = A[i];
        else sum += A[i];
        if (sum > max) max = sum;
    }
    return max;
}
```

3. perfix 表示不懂+1

```
public int maxSubArray(int[] A) {
    if (A == null || A.length == 0){
        return 0;
    }

    int max = Integer.MIN_VALUE, sum = 0, minSum = 0;
    for (int i = 0; i < A.length; i++) {
        sum += A[i];
```

```
            max = Math.max(max, sum - minSum);
            minSum = Math.min(minSum, sum);
        }

        return max;
    }
```

easy除tree和linkedList问题外全部过一遍

##Medium

####338. Counting Bits Q:For num = 5 you should return [0,1,1,2,1,2]. 比如返回2-4之间的数，可以先算上2的结果，再加上后面的部分，所以核心代码result[i] = result[i>>1] + (i&1);

####413. Arithmetic Slices Example: A = [1, 2, 3, 4] return: 3, for 3 arithmetic slices in A: [1, 2, 3], [2, 3, 4] and [1, 2, 3, 4] itself.

一条很有用的discuss： sum += curr really does the trick. Brilliant! I think the easy way to understand this is that adding current number to our existing arithmetic sequence, we will have curr additional combinations of new arithmetic slices. Let's say if we have [1, 2, 3, 4] and currently we have 3 arithmetic slices (curr is 2). We are going to add 5 to our arithmetic sequence. So that we will have curr new slices (curr is 3), which is [3, 4, 5], [2, 3, 4, 5] and [1, 2, 3, 4, 5]. Now, the total valid arithmetic slices is 3 + curr = 6. That's exactly the same as sum += curr.

####406. Queue Reconstruction by Height 题目大意：Suppose you have a random list of people standing in a queue. Each person is described by a pair of integers (h, k), where h is the height of the person and k is the number of people in front of this person who have a height greater than or equal to h. Write an algorithm to reconstruct the queue. tag: Greedy

> ex: Input: [[7,0], [4,4], [7,1], [5,0], [6,1], [5,2]] Output: [[5,0], [7,0], [5,2], [6,1], [4,4], [7,1]]

厉害了的思路（discuss）：(下面的算法就是这个思路的实现)

1. Pick out tallest group of people and sort them in a subarray (S). Since there's no other groups of people taller than them, therefore each guy's index will be just as same as his k value.
2. For 2nd tallest group (and the rest), insert each one of them into (S) by k value. So on and so forth.

一个java语法点：（parameters）->{statements;}

1. 只有一个参数时，可以不加（）
2. 没有参数时 ()->{...;}
3. statements有return时必须{}，其他情况可以不加

```
public class Solution {
    public int[][] reconstructQueue(int[][] people) {
        Arrays.sort(people, new Comparator<int[]>(){
           @Override
           public int compare(int[] a, int[] b){
               //h不相等的时候大的在前面，h相等时，k小的在前面
               return a[0]!=b[0]?-a[0]+b[0]:a[1]-b[1];
           }
        });
        //        Arrays.sort(people, (a, b) -> a[0] ==
b[0] ? a[1] - b[1] : b[0] - a[0]);
        List<int[]> res = new LinkedList<>();
        for(int[] cur : people){
            res.add(cur[1],cur);
        }
        return res.toArray(new int[people.length][]);
    }
}
```

####238. Product of Array Except Self 注意算right的时候不能简写省去right，因为此时re已经有左边的值了，不能像上面处理左边一样直接用re[i-

1]/re[i+1]

####442. Find All Duplicates in an Array Given an array of integers, 1 ≤ a[i] ≤ n (n = size of array),这句话很熟悉，数组中所有元素都是<n所以可以用a[a[i]]元素当下标的方式，跟一个题很类似

1. 遍历数组，把数组中元素值取绝对之后（确保>0）当做索引 idx
2. 如果 nums[idx]>0 (initially all elements >0) num[idx] = -num[idx]
3. 如果 nums[idx]<0，说明我们已经见过一次该元素了，此数即duplicate，注意这里的此数是nums[i]

```
for(int i=0;i<nums.length;i++){
    int idx =Math.abs(nums[i]);
    if(nums[idx-1]<0){
        re.add(idx);
    }else{
        nums[idx-1]=-nums[idx-1];
    }
}
return re;
```

####392. is Subsequence 注意题目：A subsequence of a string is a new string which is formed from the original string by deleting some (can be none) of the characters without disturbing the relative positions of the remaining characters. (ie, "ace" is a subsequence of "abcde" while "aec" is not).

所以有个很简单的方法就是这样，注意转成char[]会快很多，原因还没仔细想，应该是string的操作都慢，因为string都是reference拷贝，所以慢吧，

想了个hashmap的方法,一个存map一次比较，注意这种不行，不能保证顺序

tag说要binary search，先留坑

```
public boolean isSubsequence(String s, String t) {
```

```
        char[] ss = s.toCharArray();
        char[] tt = t.toCharArray();
        int i = 0, j = 0;
        while( i<ss.length&& j<tt.length){
            if(ss[i]==tt[j]){
                i++;
            }
            j++;
        }
        return i == s.length();
    }
```

####128. Longest Consecutive Sequence 这题难点应该在时间复杂度的要求上，不然的话sort很好做，这题时间复杂度要求是O(n). discuss中找到的方法，amorized O(n). 代码有两个重点：

1. 首先选用set，数组中所有数遍历入set中
2. 每次遍历到一个数时，在set中要挨个找比它大1的，也要挨个找比它小1的，所以要在set里找两次
3. 每次找到之后就把当前这个数删掉，降低了复杂度，避免重复比较判断

```
public int longestConsecutive(int[] nums) {
  int max = 0;

  Set<Integer> set = new HashSet<Integer>();
  for (int i = 0; i < nums.length; i++) {
    set.add(nums[i]);
  }

  for (int i = 0; i < nums.length; i++) {
    int count = 1;

    // look left
    int num = nums[i]-1;
```

```
    while (set.contains(num)) {
      count++;
      set.remove(num);
      num--;
    }

    // look right
    num = nums[i]+1;
    while (set.contains(num)) {
      count++;
      set.remove(num);
      num++;
    }
    max = Math.max(max, count);
  }

  return max;
}
```

####454. 4Sum II 这题tag是binary search，但没看到binary search的答案，所以选择hashmap，time complexity: O(n^2); space complexity: O(n^2)

> discuss 的一段解释，写的很清楚：Take the arrays A and B, and compute all the possible sums of two elements. Put the sum in the Hash map, and increase the hash map value if more than 1 pair sums to the same value. Compute all the possible sums of the arrays C and D. If the hash map contains the opposite value of the current sum, increase the count of four elements sum to 0 by the counter in the map.

```
    public int fourSumCount(int[] A, int[] B, int[] C,
int[] D) {
        Map<Integer,Integer> sums = new HashMap<>();
        for(int i = 0; i< A.length; i++){
```

```
            for(int j=0;j<B.length;j++){
                            int sum = A[i]+B[j];
                //这里也可以用 map.getOrDefault 方法，是java7
新加入的
                            if(sums.containsKey(sum)) {
                                    sums.put(sum,
sums.get(sum)+1);
                            } else {
                                    sums.put(sum, 1);
                            }
                    }
        }
        int count = 0;
        for(int i = 0; i< C.length; i++){
            for(int j=0;j<D.length;j++){
                            int sum = 0-(C[i]+D[j]);
                            if(sums.containsKey(sum)) {
                                    count+=sums.get(sum);
                            }
                    }
        }
        return count;
    }
```

算是目标吧，easy-->mediam,ac高到低排，到这里应该200道

##hard

####41. First Missing Positive

1. If the datastructure can be mutated in place and supports random access then you can do it in O(N) time and O(1) additional space.
2. Just go through the array sequentially and for every index write the value at the index to the index specified by value, recursively placing any value at that location to its place and throwing away values > N.

3. Then go again through the array looking for the spot where value doesn't match the index - that's the smallest value not in the array.
4. This results in at most 3N comparisons and only uses a few values worth of temporary space.

###遇到的关于github的问题 问题： 12/23/2016刷的题的commit都没显示到账户。具体如下： 12/23 换电脑，在新电脑上通过terminal，clone repository，开始刷题，通过vscode commit和push，push的时候vscode提示输入github帐密，我就输入了一下，然后github上不显示我原本github的用户名xy7313的commit反而显示了真实用户名提交的commit，所以github说23号这天的contribution是0，然而并不是，是3，只不过归在了真实姓名下面而不是xy7313下面。 解决方法：

1. 更改了vs code 用户setting git sync false改成了true，猜测可能是vscode问题，所以在vscode中随便改了一点用户设置作为尝试，失败
2. 想到更应该用终端解决，果然在终端中尝试commit的时候会提示committer的username，是真实姓名，根据终端提示设置修改配置，之后成功

> 终端提示内容： You can suppress this message by setting them explicitly: git config --global user.name "Your Name" git config --global user.email you@example.com After doing this, you may fix the identity used for this commit with: git commit --amend --reset-author