BITTIGER

CS102 Top100高频算法设计课

第七、八节 字符串、数组、矩阵和其他常见题目

左程云

BITTIGER

版权声明

BITTIGER

Copyright Policy

All content included on the Site or third-party platforms as part of the class, such as text, graphics, logos, button icons, images, audio clips, video clips, live streams, digital downloads, data compilations, and software, is the property of BitTiger or its content suppliers and protected by copyright laws.

Any attempt to redistribute or resell BitTiger content will result in the appropriate legal action being taken.

We thank you in advance for respecting our copyrighted content.For more info see https://www.bittiger.io/termsofuse and https://www.bittiger.io/termsofservice

判断两个字符串是否互为旋转词

【题目一】

如果一个字符串str，把字符串str前面任意的部分挪到后面形成的字符串叫作str的旋转词。比如str="12345"，str的旋转词有"12345"、"23451"、"34512"、"45123"和"51234"。给定两个字符串a和b，请判断a和b是否互为旋转词。

【举例】

a="cdab"，b="abcd"，返回true。

a="1ab2"，b="ab12"，返回false。

a="2ab1"，b="ab12"，返回true。

【要求】

如果a和b长度不一样，那么a和b必然不互为旋转词，可以直接返回false。当a和b长度一样，都为*N*时，要求解法的时间复杂度为*O*(*N*)。

判断两个字符串是否互为旋转词

【解题点】

如果a和b的长度不一样，字符串a和b不可能互为旋转词。如果a和b长度一样，先生成一个大字符串b2，b2是两个字符串b拼在一起的结果，即String b2 = b + b。然后看b2中是否包含字符串a，如果包含，说明字符串a和b互为旋转词，否则说明两个字符串不互为旋转词。

翻转字符串

【题目二】
给定一个字符类型的数组chas，请在单词间做逆序调整。只要做到单词顺序逆序即可，对空格的位置没有特别要求。
【举例】
如果把chas看作字符串为"dog loves pig"，调整成"pig Loves dog"。
如果把chas看作字符串为"I’m a student."，调整成"student. a I'm"。
【补充题目】
给定一个字符类型的数组chas和一个整数size，请把大小为size的左半区整体移到右半区，右半区整体移到左边。
【举例】
如果把chas看作字符串为"ABCDE"，size=3，调整成"DEABC"。
【要求】
如果chas长度为*N*，两道题都要求时间复杂度为*O*(*N*)，额外空间复杂度为*O*(1)。

翻转字符串

【解题点】
原问题。首先把chas整体逆序。在逆序之后，遍历chas找到每一个单词，然后把每个单词里的字符逆序即可。比如“dog loves pig”，先整体逆序变为“gip sevol god”，然后每个单词进行逆序处理就变成了“pig loves dog”。

补充问题，方法一。先把chas[0..size-1]部分逆序，再把chas[size..N-1]部分逆序，最后把chas整体逆序即可。比如，chas="ABCDE"，size=3。先把chas[0..2]部分逆序，chas变为"CBADE"，再把chas[3..4]部分逆序，chas变为"CBAED"，最后把chas整体逆序，chas变为"DEABC"。

方法二，辗转反侧法，算是在俗不可耐的套路中给你点变化

公式字符串求值

【题目三】
给定一个字符串str，str表示一个公式，公式里可能有整数、加减乘除符号和左右括号，返回公式的计算结果。
【举例】
str="48*((70-65)-43)+8*1"，返回-1816。
str="3+1*4"，返回7。
str="3+(1*4)"，返回7。
【说明】
1. 可以认为给定的字符串一定是正确的公式，即不需要对str做公式有效性检查。
2. 如果是负数，就需要用括号括起来，比如"4*(-3)"。但如果负数作为公式的开头或括号部分的开头，则可以没有括号，比如"-3*4"和"(-3*4)"都是合法的。
3. 不用考虑计算过程中会发生溢出的情况。

公式字符串求值

【解题点】
从左到右遍历str，开始遍历或者遇到字符'('时，就进行递归过程。当发现str遍历完，或者遇到字符')'时，递归过程就结束。

int[] value(char[] str, int i)

value方法的第二个参数代表递归过程是从什么位置开始的，返回的结果是一个长度为2的数组，记为res。res[0]表示这个递归过程计算的结果，res[1]表示这个递归过程遍历到str的什么位置。

公式字符串求值

【解题点】
既然在递归过程中遇到'('就交给下一层的递归过程处理，自己只用接收'('和')'之间的公式字符子串的结果，所以对所有的递归过程来说，可以看作计算的公式都是不含有'('和')'字符的。比如，对递归过程value(str,0)来说，实际上计算的公式是"3*9+7"，"(4+5)"的部分交给递归过程value(str,3)处理，拿到结果9之后，再从字符'+'继续。所以，只要想清楚如何计算一个不含有'('和')'的公式字符串，整个实现就完成了。

一个不含有‘(’和‘)’的公式字符串，你可以把乘除的部分先求出来，剩下的部分就是加减连接了。顺序求即可。

找到字符串的最长无重复字符子串

【题目四】
给定一个字符串str，返回str的最长无重复字符子串的长度。
【举例】
str="abcd"，返回4
str="aabcb"，最长无重复字符子串为"abc"，返回3。
【要求】
如果str的长度为*N*，请实现时间复杂度为*O*(*N*)的方法。

找到字符串的最长无重复字符子串

【解题点】

如果str长度为*N*，字符编码范围是*M*，本题可做到的时间复杂度为*O*(*N*)，额外空间复杂度为*O*(*M*)。下面介绍这种方法的具体实现。

1．在遍历str之前，先申请几个变量。哈希表map，key表示某个字符，value为这个字符最近一次出现的位置。整型变量pre，如果当前遍历到字符str[i]，pre表示在必须以str[i-1]字符结尾的情况下，最长无重复字符子串开始位置的前一个位置，初始时pre=-1。整型变量len，记录以每一个字符结尾的情况下，最长无重复字符子串长度的最大值，初始时，len=0。从左到右依次遍历str，假设现在遍历到str[i]，接下来求在必须以str[i]结尾的情况下，最长无重复字符子串的长度。

2．map(str[i])的值表示之前的遍历中最近一次出现str[i]字符的位置，假设在a位置。想要求以str[i]结尾的最长无重复子串，a位置必然不能包含进来，因为str[a]等于str[i]。

找到字符串的最长无重复字符子串

【解题点】

3. 根据pre的定义，pre+1表示在必须以str[i-1]字符结尾的情况下，最长无重复字符子串的开始位置，也就是说，以str[i-1]结尾的最长无重复子串是向左扩到pre位置停止的。

4. 如果pre位置在a位置的左边，因为str[a]不能包含进来，而str[a+1..i-1]上都是不重复的，所以以str[i]结尾的最长无重复字符子串就是str[a+1..i]。如果pre位置在a位置的右边，以str[i-1]结尾的最长无重复子串是向左扩到pre位置停止的。所以以str[i]结尾的最长无重复子串向左扩到pre位置也必然会停止，而且str[pre+1..i-1]这一段上肯定不含有str[i]，所以以str[i]结尾的最长无重复字符子串就是str[pre+1..i]。

5. 计算完长度之后，pre位置和a位置哪一个在右边，就作为新的pre值。然后去计算下一个位置的字符，整个过程中求得所有长度的最大值用len记录下来返回即可。

最小包含子串的长度

【题目五】
给定字符串str1和str2，求str1的子串中含有str2所有字符的最小子串长度。
【举例】
str1="abcde"，str2="ac"。因为"abc"包含str2的所有字符，并且在满足这一条件的str1的所有子串中，"abc"是最短的，返回3。
str1="12345"，str2="344"。最小包含子串不存在，返回0。

最小包含子串的长度

【解题点】

1，建立一张str2的词频表map，统计str2中字符的出现情况

2，用left和right两个指针表示在str1中的窗口，开始时left和right在str1的最左边

3，用全局变量all表示目前left和right组成的窗口一共欠str2几个字符，开始时all的值是str2的长度

4，right向右扩的过程中遇到一个字符，在map中相应的词频就减1。如果减完之后词频不为负，all的数量也减1。如果减完之后词频为负，all的数量不变

5，当all变为0，left开始往右移动，过程中遇到一个字符，在map中相应的词频就加1，如果加完之后词频不为正，all的数量不变。如果加完之后词频为正，all的数量加1，left移动停止。此时统计left和right窗口的大小

6，重复步骤4直到窗口滑动完毕

转圈打印矩阵

【题目六】
给定一个整型矩阵matrix，请按照转圈的方式打印它。
例如：

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

打印结果为：1，2，3，4，8，12，16，15，14，13，9，5，6，7，11，10
【要求】
额外空间复杂度为$O(1)$。

转圈打印矩阵

【解题点】

本题在算法上没有难度，关键在于设计一种逻辑容易理解、代码易于实现的转圈遍历方式。这里介绍这样一种矩阵处理方式，该方式不仅可用于这道题，还适合很多其他的面试题，就是矩阵分圈处理。在矩阵中用左上角的坐标(tR,tC)和右下角的坐标(dR,dC)就可以表示一个子矩阵，比如，题目中的矩阵，当(tR,tC)=(0,0)、(dR,dC)=(3,3)时，表示的子矩阵就是整个矩阵，那么这个子矩阵最外层的部分如下：

```
1   2   3   4
5           8
9           12
13  14  15  16
```

转圈打印矩阵

【解题点】

如果能把这个子矩阵的外层转圈打印出来，那么在(tR,tC)=(0,0)、(dR,dC)=(3,3)时，打印的结果为：1，2，3，4，8，12，16，15，14，13，9，5。接下来令tR和tC加1，即(tR,tC)=(1,1)，令dR和dC减1，即(dR,dC)=(2,2)，此时表示的子矩阵如下：

6　　7

10　　11

再把这个子矩阵转圈打印出来，结果为：6，7，11，10。把tR和tC加1，即(tR,tC)=(2,2)，令dR和dC减1，即(dR,dC)=(1,1)。如果发现左上角坐标跑到了右下角坐标的右方或下方，整个过程就停止。已经打印的所有结果连起来就是我们要求的打印结果。

将正方形矩阵顺时针转动90°

【题目七】

给定一个*N*×*N*的矩阵matrix，把这个矩阵调整成顺时针转动90°后的形式。

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

顺时针转动90°后为：

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

【要求】

额外空间复杂度为*O*(1)。

将正方形矩阵顺时针转动90°

【解题点】

使用分圈处理的方式，在矩阵中用左上角的坐标(tR,tC)和右下角的坐标(dR,dC)就可以表示一个子矩阵。比如，题目中的矩阵，当(tR,tC)=(0,0)、(dR,dC)=(3,3)时，表示的子矩阵就是整个矩阵，那么这个子矩阵最外层的部分如下。

```
1  2  3  4
5           8
9          12
13 14 15 16
```

将正方形矩阵顺时针转动90°

【解题点】

在这个外圈中，1，4，16，13为一组，然后让1占据4的位置，4占据16的位置，16占据13的位置，13占据1的位置，一组就调整完了。然后2，8，15，9为一组，继续占据调整的过程，最后3，12，14，5为一组，继续占据调整的过程。然后(tR,tC)=(0,0)、(dR,dC)=(3,3)的子矩阵外层就调整完毕。接下来令tR和tC加1，即(tR,tC)=(1,1)，令dR和dC减1，即(dR,dC)=(2,2)，此时表示的子矩阵如下。

6　　7

10　　11

这个外层只有一组，就是6，7，11，10，占据调整之后即可。所以，如果子矩阵的大小是$M \times M$，一共就有M-1组，分别进行占据调整即可。

“之”字形打印矩阵

【题目八】
给定一个矩阵matrix，按照“之”字形的方式打印这个矩阵，例如：
1　　2　　3　　4
5　　6　　7　　8
9　　10　　11　　12
“之”字形打印的结果为：1，2，5，9，6，3，4，7，10，11，8，12
【要求】
额外空间复杂度为O(1)。

“之”字形打印矩阵

【解题点】

1. 上坐标(tR,tC)初始为(0,0)，先沿着矩阵第一行移动(tC++)，当到达第一行最右边的元素后，再沿着矩阵最后一列移动(tR++)。
2. 下坐标(dR,dC)初始为(0,0)，先沿着矩阵第一列移动(dR++)，当到达第一列最下边的元素时，再沿着矩阵最后一行移动(dC++)。
3. 上坐标与下坐标同步移动，每次移动后的上坐标与下坐标的连线就是矩阵中的一条斜线，打印斜线上的元素即可。
4. 如果上次斜线是从左下向右上打印的，这次一定是从右上向左下打印，反之亦然。总之，可以把打印的方向用boolean值表示，每次取反即可。

需要排序的最短子数组长度

【题目九】
给定一个无序数组arr，求出需要排序的最短子数组长度。
例如：arr = [1，5，3，4，2，6，7]返回4，因为只有[5，3，4，2]需要排序。

需要排序的最短子数组长度

【解题点】

初始化变量noMinIndex=-1，从右向左遍历，遍历的过程中记录右侧出现过的数的最小值，记为min。假设当前数为arr[i]，如果arr[i]>min，说明如果要整体有序，min值必然会挪到arr[i]的左边。用noMinIndex记录最左边出现这种情况的位置。

如果遍历完成后，noMinIndex依然等于-1，说明从右到左始终不升序，原数组本来就有序，直接返回0，即完全不需要排序。

接下来从左向右遍历，遍历的过程中记录左侧出现过的数的最大值，记为max。假设当前数为arr[i]，如果arr[i]<max，说明如果排序，max值必然会挪到arr[i]的右边。用变量noMaxIndex记录最右边出现这种情况的位置。

遍历完成后，arr[noMinIndex..noMaxIndex]是真正需要排序的部分，返回它的长度即可。

在数组中找到出现次数大于*N*/*K*的数

【题目十】
给定一个整型数组arr，打印其中出现次数大于一半的数，如果没有这样的数，打印提示信息。
【进阶】
给定一个整型数组arr，再给定一个整数*K*，打印所有出现次数大于*N*/*K*的数，如果没有这样的数，打印提示信息。
【要求】
原问题要求时间复杂度为*O*(*N*)，额外空间复杂度为*O*(1)。进阶问题要求时间复杂度为*O*(*N*×*K*)，额外空间复杂度为*O*(*K*)。

在数组中找到出现次数大于*N/K*的数

【解题点】

1，一次删掉K个不同的数，如果最后没法删了，剩下的数是唯一的候选数

2，代码实现可以做到很简洁

在行列都排好序的矩阵中找数

【题目十一】
给定一个有*N*×*M*的整型矩阵matrix和一个整数*K*，matrix的每一行和每一列都是排好序的。实现一个函数，判断*K*是否在matrix中。
例如：

0	1	2	5
2	3	4	7
4	4	4	8
5	7	7	9

如果*K*为7，返回true；如果*K*为6，返回false。
【要求】
时间复杂度为*O*(*N*+*M*)，额外空间复杂度为*O*(1)。

在行列都排好序的矩阵中找数

【解题点】

左上角开始，当前数大于k，往下；当前数小于k，往左；

最长的可整合子数组的长度

【题目十二】
先给出可整合数组的定义。如果一个数组在排序之后，每相邻两个数差的绝对值都为1，则该数组为可整合数组。例如，[5,3,4,6,2]排序之后为[2,3,4,5,6]，符合每相邻两个数差的绝对值都为1，所以这个数组为可整合数组。
给定一个整型数组arr，请返回其中最大可整合子数组的长度。例如，[5,5,3,2,6,4,3]的最大可整合子数组为[5,3,2,6,4]，所以返回5。

最长的可整合子数组的长度

【解题点】

判断一个数组是否是可整合数组还可以用以下方法来判断，一个数组中如果没有重复元素，并且如果最大值减去最小值，再加1的结果等于元素个数（max-min+1==元素个数），那么这个数组就是可整合数组。比如[3,2,5,6,4]，max-min+1=6-2+1=5==元素个数，所以这个数组是可整合数组。

这样，验证每一个子数组是否是可整合数组的时间复杂度可以从第一种方法的$O(N\log N)$加速至$O(1)$，整个过程的时间复杂度就可加速到$O(N^2)$。

未排序正数数组中累加和为给定值的最长子数组长度

【题目十三】
给定一个数组arr，该数组无序，但每个值均为正数，再给定一个正数*k*。求arr的所有子数组中所有元素相加和为*k*的最长子数组长度。
例如，arr=[1,2,1,1,1]，*k*=3。
累加和为3的最长子数组为[1,1,1]，所以结果返回3。

未排序正数数组中累加和为给定值的最长子数组长度

【解题点】

首先用两个位置来标记子数组的左右两头，记为left和right，开始时都在数组的最左边(left=0,right=0)。整体过程如下：

1. 开始时变量left=0，right=0，代表子数组arr[left..right]。
2. 变量sum始终表示子数组arr[left..right]的和。开始时sum=arr[0]，即arr[0..0]的和。
3. 变量len一直记录累加和为*k*的所有子数组中最大子数组的长度。开始时，len=0。

未排序正数数组中累加和为给定值的最长子数组长度

【解题点】

4．根据sum与*k*的比较结果决定是left移动还是right移动，具体如下：

如果sum==k，说明arr[left..right]累加和为*k*，如果arr[left..right]长度大于len，则更新len，此时因为数组中所有的值都为正数，那么所有从left位置开始，在right之后的位置结束的子数组，即arr[left..i(i>right)]，累加和一定大于*k*。所以，令left加1，这表示我们开始考查以left之后的位置开始的子数组，同时令sum-=arr[left]，sum此时开始表示arr[left+1..right]的累加和。如果sum小于*k*，说明arr[left..right]还需要加上right后面的值，其和才可能达到*k*，所以，令right加1，sum+=arr[right]。需要注意的是，right加1后是否越界。如果sum大于*k*，说明所有从left位置开始，在right之后的位置结束的子数组，即arr[left..i(i>right)]，累加和一定大于*k*。所以，令left加1，这表示我们开始考查以left之后的位置开始的子数组，同时令sum-=arr[left]，sum此时表示arr[left+1..right]的累加和。

5．如果right<arr.length，重复步骤4。否则直接返回len，全部过程结束。

计算数组的小和

【题目十四】
数组小和的定义如下：
例如，数组s=[1,3,5,2,4,6]，在s[0]的左边小于或等于s[0]的数的和为0，在s[1]的左边小于或等于s[1]的数的和为1，在s[2]的左边小于或等于s[2]的数的和为1+3=4，在s[3]的左边小于或等于s[3]的数的和为1，在s[4]的左边小于或等于s[4]的数的和为1+3+2=6，在s[5]的左边小于或等于s[5]的数的和为1+3+5+2+4=15，所以s的小和为0+1+4+1+6+15=27。
给定一个数组s，实现函数返回s的小和。

计算数组的小和

【解题点】

1，归并排序的改写而已

2，逆序对问题也一样

子数组、子矩阵的最大累加和问题

【题目十五】
给定一个数组arr，返回子数组的最大累加和。
例如，arr=[1,-2,3,5,-2,6,-1]，所有的子数组中，[3,5,-2,6]可以累加出最大的和12，所以返回12。
【要求】
如果arr长度为*N*，要求时间复杂度为*O*(*N*)，额外空间复杂度为*O*(1)。

子数组、子矩阵的最大累加和问题

【题目十五补充】
给定一个矩阵matrix，其中的值有正、有负、有0，返回子矩阵的最大累加和。
例如，矩阵matrix为：
-90　　48　　78
64　　-40　　64
-81　　-7　　66
其中，最大累加和的子矩阵为：
48　　78
-40　　64
-7　　66
所以返回累加和209

子数组、子矩阵的最大累加和问题

【子数组题目解题点】
如果arr中没有正数，产生的最大累加和一定是数组中的最大值。
如果arr中有正数，从左到右遍历arr，用变量cur记录每一步的累加和，遍历到正数cur增加，遍历到负数cur减少。当cur<0时，说明累加到当前数出现了小于0的结果，那么累加的这一部分肯定不能作为产生最大累加和的子数组的左边部分，此时令cur=0，表示重新从下一个数开始累加。当cur>=0时，每一次累加都可能是最大的累加和，所以，用另外一个变量max全程跟踪记录cur出现的最大值即可。

子数组、子矩阵的最大累加和问题

【子矩阵题目解题点】

请先理解“子数组的最大累加和问题”，因为本题的最优解深度利用了上一题的解法。首先来看这样一个例子，假设一个2行4列的矩阵如下：

-2　　3　　-5　　7

1　　4　　-1　　-3

如何求必须含有2行元素的子矩阵中的最大累加和？可以把两列的元素累加，然后得到累加数组[-1,7,-6,4]，接下来求这个累加数组的最大累加和，结果是7。也就是说，必须含有2行元素的子矩阵中的最大和为7，且这个子矩阵是：

3

4

子数组、子矩阵的最大累加和问题

【子矩阵题目解题点】

也就是说，如果一个矩阵一共有*k*行且限定必须含有*k*行元素的情况下，我们只要把矩阵中每一列的*k*个元素累加生成一个累加数组，然后求出这个数组的最大累加和，这个最大累加和就是必须含有*k*行元素的子矩阵中的最大累加和。

请读者务必理解以上解释，下面看原问题如何求解。为了方便讲述，我们用题目的第一个例子来展示求解过程，首先考虑只有一行的矩阵[-90,48,78]，因为只有一行，所以累加数组arr就是[-90,48,78]，这个数组的最大累加和为126。

接下来考虑含有两行的矩阵：

-90	48	78
64	-40	64

这个矩阵的累加数组就是在上一步的累加数组[-90,48,78]的基础上，依次在每个位置上加上矩阵最新一行[64, -40, 64]的结果，即[-26,8,142]，这个数组的最大累加和为150。

子数组、子矩阵的最大累加和问题

【子矩阵题目解题点】

接下来考虑含有三行的矩阵：

-90 48 78

64 -40 64

-81 -7 66

这个矩阵的累加数组就是在上一步累加数组[-26,8,142]的基础上，依次在每个位置上加上矩阵最新一行[-81,-7,66]的结果，即[-107,1,208]，这个数组的最大累加和为209。此时，必须从矩阵的第一行元素开始，并往下的所有子矩阵已经查找完毕，接下来从矩阵的第二行开始，继续这样的过程，含有一行矩阵：

64 -40 64

因为只有一行，所以累加数组就是[64,-40,64]，这个数组的最大累加和为88。

接下来考虑含有两行的矩阵：

子数组、子矩阵的最大累加和问题

【子矩阵题目解题点】

接下来考虑含有两行的矩阵：

64　　-40　　64

-81　　-7　　66

这个矩阵的累加数组就是在上一步累加数组[64,-40,64]的基础上，依次在每个位置上加上矩阵最新一行[-81,-7,66]的结果，即[-17,-47,130]，这个数组的最大累加和为130。

此时，必须从矩阵的第二行元素开始，并往下的所有子矩阵已经查找完毕，接下来从矩阵的第三行开始，继续这样的过程，含有一行矩阵：

-81　　-7　　66

因为只有一行，所以累加数组就是[-81,-7,66]，这个数组的最大累加和为66

子数组、子矩阵的最大累加和问题

【子矩阵题目解题点】
全部过程结束，所有的子矩阵都已经考虑到了，结果为以上所有最大累加和中最大的209
整个过程最关键的地方有两处：
用求累加数组的最大累加和的方式得到每一步的最大子矩阵的累加和。
每一步的累加数组可以利用前一步求出的累加数组很方便地更新得到。

在数组中找到一个局部最小的位置

【题目十六】
定义局部最小的概念。arr长度为1时，arr[0]是局部最小。arr的长度为*N*(*N*>1)时，如果arr[0]<arr[1]，那么arr[0]是局部最小；如果arr[N-1]<arr[N-2]，那么arr[N-1]是局部最小；如果0<*i*<*N*-1，既有arr[i]<arr[i-1]，又有arr[i]<arr[i+1]，那么arr[i]是局部最小。
给定无序数组arr，已知arr中任意两个相邻的数都不相等。写一个函数，只需返回arr中任意一个局部最小出现的位置即可。

在数组中找到一个局部最小的位置

【解题点】

本题可以利用二分查找做到时间复杂度为$O(\log N)$、额外空间复杂度为$O(1)$，步骤如下：

1. 如果arr为空或者长度为0，返回-1表示不存在局部最小。
2. 如果arr长度为1或者arr[0]<arr[1]，说明arr[0]是局部最小，返回0。
3. 如果arr[N-1]<arr[N-2]，说明arr[N-1]是局部最小，返回N-1。
4. 如果arr长度大于2且arr的左右两头都不是局部最小，则令left=1，right=N-2，然后进入步骤5做二分查找。

在数组中找到一个局部最小的位置

【解题点】

5. 令mid=(left+right)/2，然后进行如下判断：

1）如果arr[mid]>arr[mid-1]，可知在arr[left..mid-1]上肯定存在局部最小，令right=mid-1，重复步骤5。

2）如果不满足1)，但arr[mid]>arr[mid+1]，可知在arr[mid+1..right]上肯定存在局部最小，令left=mid+1，重复步骤5。

3）如果既不满足1)，也不满足2)，那么arr[mid]就是局部最小，直接返回mid。

6. 步骤5一直进行二分查找，直到left==right时停止，返回left即可。

数组中子数组的最大累乘积

【题目十七】
给定一个double类型的数组arr，其中的元素可正、可负、可0，返回子数组累乘的最大乘积。例如，arr=[-2.5，4，0，3，0.5，8，-1]，子数组[3，0.5，8]累乘可以获得最大的乘积12，所以返回12。

数组中子数组的最大累乘积

【解题点】

本题可以做到时间复杂度为*O*(*N*)、额外空间复杂度为*O*(1)。所有的子数组都会以某一个位置结束，所以，如果求出以每一个位置结尾的子数组最大的累乘积，在这么多最大累乘积中最大的那个就是最终的结果。也就是说，结果=Max{以arr[0]结尾的所有子数组的最大累乘积，以arr[1]结尾的所有子数组的最大累乘积……以arr[arr.length-1]结尾的所有子数组的最大累乘积}。

如何快速求出所有以*i*位置结尾(arr[i])的子数组的最大乘积呢？假设以arr[i-1]结尾的最小累乘积为min，以arr[i-1]结尾的最大累乘积为max。那么，以arr[i]结尾的最大累乘积只有以下三种可能：

数组中子数组的最大累乘积

【解题点】

可能是max*arr[i]。max既然表示以arr[i-1]结尾的最大累乘积，那么当然有可能以arr[i]结尾的最大累乘积是max*arr[i]。例如，[3,4,5]在算到5的时候。

可能是min*arr[i]。min既然表示以arr[i-1]结尾的最小累乘积，当然有可能min是负数，而如果arr[i]也是负数，两个负数相乘的结果也可能很大。例如，[-2,3,-4]在算到-4的时候。

可能仅是arr[i]的值。以arr[i]结尾的最大累乘积并不一定非要包含arr[i]之前的数。例如，[0.1,0.1,100]在算到100的时候。

这三种可能的值中最大的那个就作为以*i*位置结尾的最大累乘积，最小的作为最小累乘积，然后继续计算以*i*+1位置结尾的时候，如此重复，直到计算结束。

边界都是1的最大正方形大小

【题目十八】
给定一个*N*×*N*的矩阵matrix，在这个矩阵中，只有0和1两种值，返回边框全是1的最大正方形的边长长度。
例如：

0	1	1	1	1
0	1	0	0	1
0	1	0	0	1
0	1	1	1	1
0	1	0	1	1

其中，边框全是1的最大正方形的大小为4×4，所以返回4。

边界都是1的最大正方形大小

【解题点】

1，生成预处理矩阵right和down。right[i][j]的值表示从位置(*i*,*j*)出发向右，有多少个连续的1。down[i][j]的值表示从位置(*i*,*j*)出发向下有多少个连续的1

2，生成完这两个矩阵，可以把检查一个位置是否可以成为边长为*N*的正方形的左上角的过程加速至O(1)

数组中未出现的最小正整数

【题目十九】
给定一个无序整型数组arr，找到数组中未出现的最小正整数。
【举例】
arr=[-1,2,3,4]。返回1。
arr=[1,2,3,4]。返回5。
【要求】
做到时间复杂度为$O(N)$，额外空间复杂度为$O(1)$

数组中未出现的最小正整数

【解题点】

1. 在遍历arr之前先生成两个变量。变量*l*表示遍历到目前为止，数组arr已经包含的正整数范围是[1,*l*]，所以没有开始遍历之前令*l*=0，表示arr目前没有包含任何正整数。变量*r*表示遍历到目前为止，在后续出现最优状况的情况下，arr可能包含的正整数范围是[1,*r*]，所以没有开始遍历之前，令*r*=*N*，因为还没有开始遍历，所以后续出现的最优状况是arr包含1~*N*所有的整数。*r*同时表示arr当前的结束位置。
2. 从左到右遍历arr，遍历到位置*l*，位置*l*的数为arr[l]。
3. 如果arr[l]==l+1。没有遍历arr[l]之前，arr已经包含的正整数范围是[1,*l*]，此时出现了arr[l]==l+1的情况，所以arr包含的正整数范围可以扩到[1,*l*+1]，即令*l*++。然后重复步骤2。

数组中未出现的最小正整数

【解题点】

4. 如果arr[l]<=l。没有遍历arr[l]之前，arr在后续最优的情况下可能包含的正整数范围是[1,*r*]，已经包含的正整数范围是[1,*l*]，所以需要[*l*+1,*r*]上的数。而此时出现了arr[l]<=l，说明[*l*+1,*r*]范围上的数少了一个，所以arr在后续最优的情况下，可能包含的正整数范围缩小了，变为[1,*r*-1]，此时把arr最后位置的数(arr[r-1])放在位置*l*上，下一步检查这个数，然后令r--。重复步骤2。

5. 如果arr[l]>r，与步骤4同理，把arr最后位置的数(arr[r-1])放在位置*l*上，下一步检查这个数，然后令r--。重复步骤2。

数组中未出现的最小正整数

【解题点】

6. 如果arr[arr[l]-1]==arr[l]。如果步骤4和步骤5没中，说明arr[l]是在[*l*+1,*r*]范围上的数，而且这个数应该放在arr[l]-1位置上。可是此时发现arr[l]-1位置上的数已经是arr[l]，说明出现了两个arr[l]，既然在[*l*+1,*r*]上出现了重复值，那么[*l*+1,*r*]范围上的数又少了一个，所以与步骤4和步骤5一样，把arr最后位置的数(arr[r-1])放在位置*l*上，下一步检查这个数，然后令r--。重复步骤2。

7. 如果步骤4、步骤5和步骤6都没中，说明发现了[*l*+1,r]范围上的数，并且此时并未发现重复。那么arr[l]应该放到arr[l]-1位置上，所以把*l*位置上的数和arr[l]-1位置上的数交换，下一步继续遍历*l*位置上的数。重复步骤2。

8. 最终*l*位置和*r*位置会碰在一起（l==r），arr已经包含的正整数范围是[1,*l*]，返回*l*+1即可。

由随机到随机

【题目二十】

给定一个等概率随机产生1~5的随机函数rand1To5如下：

```
public int rand1To5() {
        return (int) (Math.random() * 5) + 1;
}
```

除此之外，不能使用任何额外的随机机制，请用rand1To5实现等概率随机产生1~7的随机函数rand1To7。

由随机到随机

【题目二十补充题目】

给定一个以*p*概率产生0，以1-*p*概率产生1的随机函数rand01p如下：

```
public int rand01p() {
        // 可随意改变p
        double p = 0.83;
        return Math.random() < p ? 0 : 1;
}
```

除此之外，不能使用任何额外的随机机制，请用rand01p实现等概率随机产生1~6的随机函数rand1To6。

由随机到随机

【原问题解题点】
插空儿
【进阶问题解题点】
P*(1-P) = (P-1)*P
【注意】
从a～b的随机，变成c～d的随机，可以一律先把a～b的随机变成二进制的随机，然后再去表示c～d的随机就容易很多了

设计RandomPool结构

【题目二十一】
设计一种结构，在该结构中有如下三个功能：
insert(key)：将某个key加入到该结构，做到不重复加入。
delete(key)：将原本在结构中的某个key移除。
getRandom()：等概率随机返回结构中的任何一个key。
【要求】
Insert、delete和getRandom方法的时间复杂度都是O(1)。

题目title

【解题点】

1，map1表示string到index的对应关系， map2表示index到string的对应关系

2，size表示结构中加入的字符串数量，也用来表示一个string加入时的index

3，加入一个字符串str时，同步更新map1，map2，size

4，删除时采用把最后一个元素放到要删除的位置上然后减小size的方式，这么做可以保证index的连续，同步更新map1，map2，size

5，getRandom时利用连续的index可以实现等概率随机

关于刷题和准备念叨两句

1，刷题时自己一定要先实现，不管多烂的实现，自己写完
2，迅速在网上找好的实现，通过时间复杂度粗筛帖子，比你好的重点看
3，理解时间复杂度为什么比你自己实现的要好，提炼出这道题的算法原型
4，最优解通常是指在先满足时间复杂度的情况下，使用更少的空间，所以当你发现时间复杂度已经无法优化，去优化自己的空间
5，总有无穷无尽的抖机灵的点，让你做不到最优解，别沮丧，因为沮丧也没用，体会到那些机灵点的美，赶紧下一道题
6，在面试一家公司之前，重点看那家公司的面经题（careercup）
7，面试第一原则：让面试官喜欢你！所以，遇到做过的题，好好装；遇到没做过的题，好好聊

关于刷题和准备念叨两句

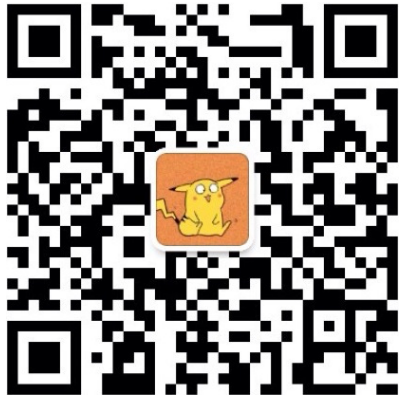
好运！

课程项目负责人：**Xiuting**

邮件：xiuting@bittiger.io

左程云答疑邮箱：**chengyunzuo@gmail.com**

微信二维码：



关注微信，获得太阁最新信息

微信: bit_tiger

官网: BitTiger.io