Vivo---20200607

Question 1（种花）：

假设你有一个很长的花坛，一部分地块种植了花，另一部分却没有。可是，花卉不能种植在相邻的地块上，它们会争夺水源，两者都会死去。

给定一个花坛（表示为一个数组包含0和1，其中0表示没种植花，1表示种植了花），和一个数 n 。能否在不打破种植规则的情况下种入 n 朵花？能则返回True，不能则返回False。

示例 1:

```
输入：flowerbed = [1,0,0,0,1], n = 1
输出：True
```

示例 2:

```
输入：flowerbed = [1,0,0,0,1], n = 2
输出：False
```

注意:

1. 数组内已种好的花不会违反种植规则。
2. 输入的数组长度范围为 [1, 20000]。
3. n 是非负整数，且不会超过输入数组的大小。

Young版

```
class Solution {
    public boolean canPlaceFlowers(int[] flowerbed, int n) {
        int count = 0;
        int[] ans;
        int len = flowerbed.length;
        if(len==1){
            if(flowerbed[0]==0){
                count ++;
                return count>=n;
            }else{
                return count>=n;
            }
        }
        for(int i=0;i<flowerbed.length;i++){
            if(i==0){
                if(flowerbed[i]==0 && (i+1)<len && flowerbed[i+1]==0 ){
                    flowerbed[i] = 1;
                    count ++;
                }
            }else if(i==flowerbed.length-1){
                if(flowerbed[i]==0 && (i-1)>0 && flowerbed[i-1]==0){
                    flowerbed[i] = 1;
                    count ++;
                }
```

```
            }else if(flowerbed[i]==0 && flowerbed[i-1]==0 && flowerbed[i+1]==0){
                flowerbed[i] = 1;
                count ++;
            }
        }
        return count>=n;
    }
}
```

参考代码：

```
public class Solution {
    public boolean canPlaceFlowers(int[] flowerbed, int n) {
        int i = 0, count = 0;
        while (i < flowerbed.length) {
            if (flowerbed[i] == 0 && (i == 0 || flowerbed[i - 1] == 0) && (i ==
flowerbed.length - 1 || flowerbed[i + 1] == 0)) {    //巧妙
                flowerbed[i] = 1;
                count++;
            }
            i++;
        }
        return count >= n;
    }
}
```

Question 2（鸡蛋掉落）

887. 鸡蛋掉落

难度 困难 384

你将获得 K 个鸡蛋，并可以使用一栋从 1 到 N 共有 N 层楼的建筑。

每个蛋的功能都是一样的，如果一个蛋碎了，你就不能再把它掉下去。

你知道存在楼层 F ，满足 0 <= F <= N 任何从高于 F 的楼层落下的鸡蛋都会碎，从 F 楼层或比它低的楼层落下的鸡蛋都不会破。

每次*移动*，你可以取一个鸡蛋（如果你有完整的鸡蛋）并把它从任一楼层 X 扔下（满足 1 <= X <= N）。

你的目标是**确切地**知道 F 的值是多少。

无论 F 的初始值如何，你确定 F 的值的最小移动次数是多少？

示例 1：

输入： K = 1, N = 2
输出： 2
解释：
鸡蛋从 1 楼掉落。如果它碎了，我们肯定知道 F = 0 。
否则，鸡蛋从 2 楼掉落。如果它碎了，我们肯定知道 F = 1 。
如果它没碎，那么我们肯定知道 F = 2 。
因此，在最坏的情况下我们需要移动 2 次以确定 F 是多少。

Question 3 （合并k个有序链表）

23. 合并K个排序链表

难度 困难 702

合并 *k* 个排序链表，返回合并后的排序链表。请分析和描述算法的复杂度。

示例:

```
输入：
[
  1->4->5,
  1->3->4,
  2->6
]
输出：1->1->2->3->4->4->5->6
```

```
package Vivo;

import java.util.*;

public class Ques2_0608 {
    public static void main(String[] args) {
    List<Integer> list1 = Arrays.asList(new Integer[] {1,4,5});
    List<Integer> list2 = Arrays.asList(new Integer[] {1,3,4});
    List<Integer>[] lists = new List[]{list1,list2};
        System.out.println(mergeKList(lists).toString());
    }

    public static List<Integer> mergeKList(List<Integer>[] lists){
        if(lists.length==0 || lists==null){
            return null;
        }
        int k = lists.length;
        //注意优先队列的使用，需要自定义比较器
        PriorityQueue<Integer> queue = new PriorityQueue<>(new
Comparator<Integer>() {
            @Override
            public int compare(Integer o1, Integer o2) {
                return o1-o2;
            }
        });
        for(int i =0;i<k;i++){
            List<Integer> temp = lists[i];
            Iterator<Integer> iterator = temp.iterator();
            while (iterator.hasNext()){
                queue.add(iterator.next());
            }
        }

        List<Integer> ans = new LinkedList<>();
        while (!queue.isEmpty()){
            ans.add(queue.poll());
        }
```

```
            return ans;
        }
    }
```

Question 4 （生产线）

在vivo产线上，每位职工随着对手机加工流程认识的熟悉和经验的增加，日产量也会不断攀升。
假设第一天量产1台，接下来2天(即第二、三天)每天量产2件，接下来3天(即第四、五、六天)每天量产3件 ... ...
以此类推，请编程计算出第n天总共可以量产的手机数量。

输入例子1:

```
11
```

输出例子1:

```
35
```



```
//思路
//[1,2,2,3,3,3,4,4,4,4,5,5,5,5,5]-->[1,2,3,4,5]对应每段的个数
//GroupCount用来计算小于n的前几段和 count用来计算属于12345哪一段中

package Vivo;

import java.util.Scanner;

public class Ques3 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        while (scanner.hasNext()){
            int numOfDay = scanner.nextInt();
            System.out.println(NumPhones(numOfDay));
        }
    }

    /**
     *
     * @param n 第n天
     * @return 生产的手机总数
     */
    public static int NumPhones(int n){
        int GroupCount = 0;
        int count = 0;
        int sum = 0;
        for(int i=0;i<n+1;i++){
            if(GroupCount<n){
                GroupCount += i+1;
                count ++;
            }else{
                GroupCount -= i;
                break;
```

```
            }
        }
        for(int i=1;i<count;i++){
            sum += Math.pow(i,2);
        }
        sum = sum+(n-GroupCount)*(count);
        return sum;
    }
}
```

Question 5 （数位之积）

现给定任意正整数n,请寻找并输出最小的正整数m(m>9),使得m的各位（个位、十位...）之乘积等于n,若不存在则输出-1.
输入样例： 36
输出样例： 49

```
package Vivo;

import java.util.Scanner;

public class Ques4 {
    public static void main(String[] args) {
        Scanner scanner =  new Scanner(System.in);
        while (scanner.hasNext()){
            int test = scanner.nextInt();
            System.out.println(solution(test));
        }
    }

    public static int solution(int n){
        int ans = resolve(n);
        if(ans>0){
            return ans;
        }
        return -1;
    }

    //递归进行整除运算
    public static int resolve(int n){
        if(n<10){
            return n;
        }
        for(int i=9;i>1;i--){
            if(n%i==0){
                return resolve(n/i)*10+i;
            }
        }
        return -1;
    }
}
```

ICBC

Question 1（考勤记录）

24 给定一个字符串来代表一个员工的考勤纪录，这个纪录仅包含以下两个字符：
'A' : Absent，缺勤
'P' : Present，到场
如果一个员工的考勤纪录中不超过两个'A'(缺勤),那么这个员工会被奖赏。
如果你作为一个员工，想在连续N天的考勤周期中获得奖赏，请问有多少种考勤的组合能够满足要求

```java
package ICBC;

import java.util.Scanner;

public class Test01 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        while (scanner.hasNext()){
            int N = scanner.nextInt();
            if(N==1){
                System.out.println(1);
            }else if(N==2){
                System.out.println(4);
            }else{
                System.out.println(N*(N-1)/2+1+N);
            }
        }
    }
}
```

Question 2（解码方式--动态规划）

25 一条包含字母 A-Z 的消息通过以下方式进行了编码：

```
'A' -> 1
'B' -> 2
...
'Z' -> 26
```

给定一个只包含数字的非空字符串，请计算解码方法的总数。

```java
package ICBC;

import java.util.Scanner;

public class Test02 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        while (scanner.hasNext()){
            String s = scanner.nextLine();
            System.out.println(numDecoding(s));
        }
    }

    //动态规划的思想：
    //当前位置的编码方式需要根据前面的步骤来计算；
```

```
    //举例：
    //1226
    //当前位置：1   -->编码方式：1
    //当前位置：2   -->编码方式：1  2 ；  12
    //当前位置：2   -->编码方式：1 2 2 ；12 2 ；1 22
    //当前位置：6   -->编码方式：1 2 2 6 ；12 2 6 ；1 22 6 ；1 2 26 ；12 26
    //规律总结：
    //如果当前数字不是0，则判断 它和前一位数字组合是否大于26，如果不大于26，则可以通过组合形式形成新的编码方法，如果和前一位组合后大于26，则只能单独出道；可以看出，如果可以组合，则是在dp[i-2] 的基础上进行组合，如果不能组合即单独出道，即在dp[i-1] 的基础上单独出道，如果又能单独出道又能组合，则dp[i] = dp[i-1] + dp[i-2]

    public static int numDecoding(String s){
        if(s.charAt(0)==0){
            return 0;  //不合法
        }
        int[] dp = new int[s.length()+1];
        dp[0] = dp[1] = 1;
        for (int i=2;i<=s.length();i++){   //dp长度为s.length()+1 所以i可以取到s.length()
            if(s.charAt(i-1)!='0'){ //合法字符
                dp[i] += dp[i-1];
            }
            String twoChar = s.substring(i-2,i);
            if(Integer.valueOf(twoChar)<=26){
                dp[i] += dp[i-2];
            }
        }
        return dp[s.length()];
    }
}
```

Question 3（最小船只数--双指针）

26 公司组织团建活动，到某漂流圣地漂流，现有如下情况：
员工各自体重不一，第 i 个人的体重为 people[i]，每艘漂流船可以承载的最大重量为 limit。
每艘船最多可同时载两人，但条件是这些人的重量之和最多为 limit。
为节省开支，麻烦帮忙计算出载到每一个人所需的最小船只数(保证每个人都能被船载)。

```
package ICBC;

import java.util.Arrays;

public class Test03 {
    public static void main(String[] args) {
        int[] test = new int[]{1,2,3,4,5,6,7,8};
        System.out.println(leastShapeNum(test,5));
    }

    public static int leastShapeNum(int[] people,int limit){
        if(people.length==0 || people==null){
            return 0;
        }
```

```
        Arrays.sort(people);
        //双指针 将最大体重和最小体重的放在一起 就可以用最小的船的数量
        int i = 0;
        int j = people.length-1;
        int result = 0;
        while (i<j){
            if(people[j]+people[i]<=limit){
                result ++;
                i++;
                j--;
            }else{
                j--;  //体重最大的独自做一条船
                result ++;
            }
        }

        if(i==j){
            result ++;  //最后剩余的一个人需要一条船
        }
        return result;
    }
}
```

Question 4 （字符串是否由子串拼成）

给出一个非空的字符串，判断这个字符串是否是由它的一个子串进行多次首尾拼接构成的。
例如，"abcabcabc"满足条件，因为它是由"abc"首尾拼接而成的，而"abcab"则不满足条件。

输入描述:

> 非空字符串

输出描述:

> 如果字符串满足上述条件，则输出最长的满足条件的的子串；如果不满足条件，则输出false。

输入例子1:

> abcabc

输出例子1:

> abc

```
public class Main{
    public static void main(String args[]){
        Scanner sc = new Scanner(System.in);
        String s = sc.nextLine();
```

```
        String result = "";
        String[] allResult = new String[s.length()];
        int count = 0;
        for(int i=0;i<s.length()/2;i++){
            if(isSubString(s,s.substring(0,i+1))){
                 allResult[count] = s.substring(0,i+1);
                 count ++;
            }
        }
        if(allResult[0]!=null){
             System.out.print(allResult[0]);
        }else{
            System.out.print(false);
        }

    }

    public static boolean isSubString(String s,String subStr){
        if((subStr+s).equals(s+subStr)){
            return true;
        }
        return false;
    }
}
```

Question 5 (洗牌算法--左右手交替)

洗牌在生活中十分常见，现在需要写一个程序模拟洗牌的过程。 现在需要洗2n张牌，从上到下依次是第1张，第2张，第3张一直到第2n张。首先，我们把这2n张牌分成两堆，左手拿着第1张到第n张（上半堆），右手拿着第n+1张到第2n张（下半堆）。接着就开始洗牌的过程，先放下右手的最后一张牌，再放下左手的最后一张牌，接着放下右手的倒数第二张牌，再放下左手的倒数第二张牌，直到最后放下左手的第一张牌。接着把牌合并起来就可以了。 例如有6张牌，最开始牌的序列是1,2,3,4,5,6。首先分成两组，左手拿着1,2,3；右手拿着4,5,6。在洗牌过程中按顺序放下了6,3,5,2,4,1。把这六张牌再次合成一组牌之后，我们按照从上往下的顺序看这组牌，就变成了序列1,4,2,5,3,6。 现在给出一个原始牌组，请输出这副牌洗牌k次之后从上往下的序列。

本题目内容摘自该链接，**招银的笔试题和这个类似，原题忘记了**，大致多了以下几个条件：

1. 牌的数量可能是奇数，可能是偶数。如果是奇数，则左手多一张牌；
2. 第奇数次洗牌，右手先放；偶数次洗牌，左手先放；

输入：

```
// 第一次数字代表有N张牌，第二个数字代表洗牌K次，后面跟N个数字表示N张牌。
7 2 1 2 3 4 5 6 7
```

复制

输出

```
// 第K次洗牌后从上往下的序列
 1 6 2 4 5 7 3
```

```
package ICBC;

import java.util.Scanner;
```

```java
public class Test09 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int num = scanner.nextInt();
        int k = scanner.nextInt();
        int[] cards = new int[num];
        for(int i=0;i<num;i++){
            cards[i] = scanner.nextInt();
        }

        for (int j=1;j<=k;j++){
            cards = shuffle(cards,j);
        }
        for(int m=0;m<cards.length-1;m++){
            System.out.print(cards[m]+" ");
        }
        System.out.print(cards[cards.length-1]);
    }

    public static int[] shuffle(int[] cards,int curTimes){
        int len = cards.length;
        int left = 0;
        int right = len-1;
        int mid = 0;
        if(len%2==0){
            mid = len/2-1;
        }else{
            mid = len/2;
        }
        left = mid;
        int[] result = new int[len];
        int index = len-1;
        while (index>=0){
            if((curTimes & 1) == 0){ //偶数次洗牌
                if(left>=0) result[index--] = cards[left--];
                if(right>mid) result[index--] = cards[right--];
            }else { //奇数次洗牌
                if(right>mid) result[index--] = cards[right--];
                if(left>=0) result[index--] = cards[left--];
            }
        }
        return result;
    }

}
```