```

题号 题目 知识点 难度 通过率

QY1 循环数比较 字符串模拟穷举 中等 25.56%

QY2 DNA序列 字符串穷举 中等 22.69%

QY3 判断题 贪心模拟 中等 46.89%

QY4 删除重复字符 字符串模拟 较难 33.46%

QY5 空中旅行 数组模拟贪心 中等 29.16%

QY6 回文素数 模拟穷举 中等 24.36%

QY7 排序 贪心排序 中等 37.11%

QY8 字符串价值 字符串贪心 中等 30.15%

QY9 拼凑正方形 贪心穷举排序 中等 37.67%

QY10 区间表达 数组模拟穷举贪心 中等 45.23%

QY11 数字游戏 数组排序贪心 中等 34.90%

QY12 红和绿 动态规划贪心穷举字符串 中等 24.02%

QY13 拼凑三角形 贪心穷举 中等 40.84%

QY14 循环数比较 字符串模拟穷举 中等 37.09%

QY15 青草游戏 模拟 中等 26.28%

QY16 无聊的牛牛和羊羊 中等 22.86%

QY17 幸运子序列 排序数组分治贪心 中等 22.30%

QY18 缺失的括号 字符串栈贪心模拟 中等 34.59%

QY19 最后一位 贪心模拟 中等 21.43%

QY20 冒泡排序 排序动态规划贪心 中等 26.74%

QY21 括号匹配深度 递归字符串穷举栈 中等 31.78%

QY22 奶牛编号 排序贪心 中等 26.40%

QY23 平方串 字符串动态规划 中等 25.42%

```

### QY1 循环数比较

#### 题目描述

```

对于任意两个正整数x和k,我们定义repeat(x, k)为将x重复写k次形成的数,例如repeat(1234, 3) = 123412341234,repeat(20,2) = 2020.

牛牛现在给出4个整数x1, k1, x2, k2, 其中v1 = (x1, k1), v2 = (x2, k2),请你来比较v1和v2的大小。

```

#### 输入描述:

```

输入包括一行,一行中有4个正整数x1, k1, x2, k2(1 ≤ x1,x2 ≤ 10^9, 1 ≤ k1,k2 ≤ 50),以空格分割

```

#### 输出描述:

```

如果v1小于v2输出"Less",v1等于v2输出"Equal",v1大于v2输出"Greater".

```

#### 示例1

#### 输入

```

1010 3 101010 2

```

#### 输出

```

Equal

```

```java

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

public class Main {

public static void main(String[] args) throws IOException {

BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));

String[] str = bufferedReader.readLine().split(" ");

String s1 = str[0];

String s2 = str[2];

int x1 = Integer.parseInt(str[1]);

int x2 = Integer.parseInt(str[3]);

System.out.println(repeat(s1, x1, s2, x2));

}

public static String repeat(String s1, Integer x1, String s2, Integer x2) {

StringBuilder stringBuffer1 = new StringBuilder(s1);

StringBuilder stringBuffer2 = new StringBuilder(s2);

for (int i = 1; i < x1; i++) {

stringBuffer1.append(s1);

}

for (int i = 1; i < x2; i++) {

stringBuffer2.append(s2);

}

char[] c1 = stringBuffer1.toString().toCharArray();

char[] c2 = stringBuffer2.toString().toCharArray();

if (stringBuffer1.length() > stringBuffer2.length()) {

return "Greater";

} else if (stringBuffer1.length() < stringBuffer2.length()) {

return "Less";

} else {

for (int i = 0; i < c1.length; i++) {

if (c1[i] - '0' > c2[i] - '0') return "Greater";

else if (c1[i] - '0' < c2[i] - '0') return "Less";

}

}

return "Equal";

}

}

```

### QY2 DNA序列

#### 题目描述

```

牛牛又从生物科研工作者那里获得一个任务,这次牛牛需要帮助科研工作者从DNA序列s中找出最短没有出现在DNA序列s中的DNA片段的长度。

例如:s = AGGTCTA

序列中包含了所有长度为1的('A','C','G','T')片段,但是长度为2的没有全部包含,例如序列中不包含"AA",所以输出2。

```

#### 输入描述:

```

输入包括一个字符串s,字符串长度length(1 ≤ length ≤ 2000),其中只包含'A','C','G','T'这四种字符。

```

#### 输出描述:

```

输出一个正整数,即最短没有出现在DNA序列s中的DNA片段的长度。

```

#### 示例1

#### 输入

```

AGGTCTA

```

#### 输出

```

2

```

```java

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.\*;

public class Main {

public static void main(String[] args) throws IOException {

BufferedReader bf = new BufferedReader(new InputStreamReader(System.in));

String str = bf.readLine();

int length = 1;

Set<String> set = new HashSet<>();

int i = 0;

while (i + length <= str.length()) {

set.add(str.substring(i, i + length));

i++;

if (set.size() == (int) Math.pow(4, length)) {

set = new HashSet<>();

length++;

i = 0;

}

}

System.out.println(length);

}

}

```

### QY3 判断题

#### 题目描述

```

牛牛参加了一场考试,考试包括n道判断题,每做对一道题获得1分,牛牛考试前完全没有准备,所以考试只能看缘分了,牛牛在考试中一共猜测了t道题目的答案是"正确",其他的牛牛猜为"错误"。考试结束后牛牛知道实际上n道题中有a个题目的答案应该是"正确",但是牛牛不知道具体是哪些题目,牛牛希望你能帮助他计算可能获得的最高的考试分数是多少。

```

#### 输入描述:

```

输入包括一行,一行中有三个正整数n, t, a(1 ≤ n, t, a ≤ 50), 以空格分割

```

#### 输出描述:

```

输出一个整数,表示牛牛可能获得的最高分是多少。

```

#### 示例1

#### 输入

```

3 1 2

```

#### 输出

```

2

```

```java

import java.io.\*;

public class Main {

public static void main(String[] args) throws IOException {

BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

String[] arg = reader.readLine().split(" ");

int n = Integer.parseInt(arg[0]);

int t = Integer.parseInt(arg[1]);

int a = Integer.parseInt(arg[2]);

if (t >= a) System.out.println("" + (a + n - t));

else System.out.println("" + (t + n - a));

}

}

```

### QY4 删除重复字符

#### 题目描述

```

牛牛有一个由小写字母组成的字符串s,在s中可能有一些字母重复出现。比如在"banana"中,字母'a'和字母'n'分别出现了三次和两次。

但是牛牛不喜欢重复。对于同一个字母,他只想保留第一次出现并删除掉后面出现的字母。请帮助牛牛完成对s的操作。

```

#### 输入描述:

```

输入包括一个字符串s,s的长度length(1 ≤ length ≤ 1000),s中的每个字符都是小写的英文字母('a' - 'z')

```

#### 输出描述:

```

输出一个字符串,表示满足牛牛要求的字符串

```

#### 示例1

#### 输入

```

banana

```

#### 输出

```

ban

```

```java

import java.io.\*;

import java.util.Arrays;

public class Main {

public static String delStr(String str) {

int[] temp = new int[256];

Arrays.fill(temp, 0);

char[] chars = str.toCharArray();

StringBuilder sb = new StringBuilder();

for (char c : chars) {

temp[c] += 1;

if (temp[c] == 1) sb.append(c);

}

return sb.toString();

}

public static void main(String[] args) throws Exception {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

String line;

while ((line = br.readLine()) != null) {

String s = delStr(line);

System.out.println(s);

}

}

}

```

### QY5 空中旅行

#### 题目描述

```

牛牛有羊羊有了属于他们自己的飞机。于是他们进行几次连续的飞行。f[i]表示第i次飞行所需的燃油的升数。飞行只能按照f数组所描述的顺序进行。

起初飞机里有s升燃油,为了正常飞行,每次飞行前飞机内燃油量应大于等于此处飞行所需要的燃油量。请帮助他们计算在不进行加油的情况下他们能进行的飞行次数。

```

#### 输入描述:

```

输入包括两行,第一行包括两个整数n和s(1 ≤ n ≤ 50, 1 ≤ s ≤ 1000),分别表示计划飞行的次数和飞起初始状态下有的燃油量。

第二行包括n个整数f[i], (1 ≤ f[i] ≤ 1000), 表示每次计划飞行所需要的燃油量。

```

#### 输出描述:

```

输出一个整数,表示他们能进行的飞行次数。

```

#### 示例1

#### 输入

```

7 10

1 2 3 4 5 6 7

```

#### 输出

```

4

```

```java

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.Arrays;

import java.util.HashSet;

import java.util.LinkedHashSet;

public class Main {

public static void main(String[] args) throws IOException {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

String[] firstLine = br.readLine().split(" ");

int time = Integer.parseInt(firstLine[0]);

int restOil = Integer.parseInt(firstLine[1]);

int count = 0;

String[] secondLine = br.readLine().split(" ");

for (int i = 0; i < time; i++) {

if (restOil - Integer.parseInt(secondLine[i]) >= 0) {

count++;

restOil -= Integer.parseInt(secondLine[i]);

} else break;

}

System.out.println(count);

}

}

```

### QY6 回文素数

#### 题目描述

```

如果一个整数只能被1和自己整除,就称这个数是素数。

如果一个数正着反着都是一样,就称为这个数是回文数。例如:6, 66, 606, 6666

如果一个数字既是素数也是回文数,就称这个数是回文素数

牛牛现在给定一个区间[L, R],希望你能求出在这个区间内有多少个回文素数。

```

#### 输入描述:

```

输入包括一行,一行中有两个整数(1 ≤ L ≤ R ≤ 1000)

```

#### 输出描述:

```

输出一个整数,表示区间内回文素数个数。

```

#### 示例1

#### 输入

```

100 150

```

#### 输出

```

2

```

```java

import java.io.\*;

public class Main {

public static void main(String[] args) throws IOException {

BufferedReader bf = new BufferedReader(new InputStreamReader(System.in));

String[] s = bf.readLine().split(" ");

int l = Integer.parseInt(s[0]);

int r = Integer.parseInt(s[1]);

int count = 0;

//循环判断[L, R]区间内的每一个数字是否为即为回文又为素数

for (int i = l; i <= r; i++) {

if (isPalindrome(i) && isprime(i)) count++;

}

System.out.println(count);

}

//判断一个数是否为素数

public static boolean isprime(int n) {

if (n == 1) return false;

for (int i = 2; i <= Math.sqrt(n); i++) {

if (n % i == 0) return false;

}

return true;

}

//判断一个数是否为回文

public static boolean isPalindrome(int n) {

int k = n;

int num = 0;

while (k != 0) {

num = num \* 10 + k % 10;

k = k / 10;

}

return num == n;

}

}

```

### QY7 排序

#### 题目描述

```

牛牛有一个长度为n的整数序列,牛牛想对这个序列进行重排为一个非严格升序序列。牛牛比较懒惰,他想移动尽量少的数就完成重排,请你帮他计算一下他最少需要移动多少个序列中的元素。(当一个元素不在它原来所在的位置,这个元素就是被移动了的)

```

#### 输入描述:

```

输入包括两行,第一行一个整数n(1 ≤ n ≤ 50),即序列的长度

第二行n个整数x[i](1 ≤ x[i] ≤ 100),即序列中的每个数

```

#### 输出描述:

```

输出一个整数,即最少需要移动的元素个数

```

#### 示例1

#### 输入

```

3

3 2 1

```

#### 输出

```

2

```

```java

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.Arrays;

public class Main {

public static void main(String[] args) throws IOException {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

int n = Integer.parseInt(br.readLine());

String[] s = br.readLine().split(" ");

int[] x = new int[n];

for (int i = 0; i < n; i++) {

x[i] = Integer.parseInt(s[i]);

}

System.out.println(getMinMoveNum(n, x));

}

private static int getMinMoveNum(int n, int[] x) {

int[] y;

int sum = 0;

y = x.clone();

Arrays.sort(y);

for (int i = 0; i < n; i++) {

if (x[i] != y[i]) sum++;

}

return sum;

}

}

```

### QY8 字符串价值

#### 题目描述

```

有一种有趣的字符串价值计算方式:统计字符串中每种字符出现的次数,然后求所有字符次数的平方和作为字符串的价值

例如: 字符串"abacaba",里面包括4个'a',2个'b',1个'c',于是这个字符串的价值为4 \* 4 + 2 \* 2 + 1 \* 1 = 21

牛牛有一个字符串s,并且允许你从s中移除最多k个字符,你的目标是让得到的字符串的价值最小。

```

#### 输入描述:

```

输入包括两行,第一行一个字符串s,字符串s的长度length(1 ≤ length ≤ 50),其中只包含小写字母('a'-'z')。

第二行包含一个整数k(0 ≤ k ≤ length),即允许移除的字符个数。

```

#### 输出描述:

```

输出一个整数,表示得到的最小价值

```

#### 示例1

#### 输入

```

aba

1

```

#### 输出

```

2

```

```java

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

public class Main {

public static void main(String[] args) throws IOException {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

String str = br.readLine();

int k = Integer.parseInt(br.readLine());

int[] z = new int[26];

for (int i = 0; i < str.length(); i++) {

z[str.charAt(i) - 'a']++;

}

for (int i = 0; i < k; i++) {

int maxId = 0;

for (int j = 0; j < 26; j++) {

if (z[j] > z[maxId]) maxId = j;

}

z[maxId]--;

}

int res = 0;

for (int i = 0; i < 26; i++) {

res += Math.pow(z[i], 2);

}

System.out.println(res);

}

}

```

### QY9 拼凑正方形

#### 题目描述

```

牛牛有4根木棍,长度分别为a,b,c,d。羊羊家提供改变木棍长度的服务,如果牛牛支付一个硬币就可以让一根木棍的长度加一或者减一。牛牛需要用这四根木棍拼凑一个正方形出来,牛牛最少需要支付多少硬币才能让这四根木棍拼凑出正方形。

```

#### 输入描述:

```

输入包括一行,四个整数a,b,c,d(1 ≤ a,b,c,d ≤ 10^6), 以空格分割

```

#### 输出描述:

```

输出一个整数,表示牛牛最少需要支付的硬币

```

#### 示例1

#### 输入

```

4 1 5 4

```

#### 输出

```

4

```

```java

import java.io.\*;

import java.util.\*;

public class Main {

public static void main(String[] args) throws IOException {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

String[] line = br.readLine().split(" ");

int[] a = new int[line.length];

for (int i = 0; i < a.length; i++) {

a[i] = Integer.parseInt(line[i]);

}

Arrays.sort(a);

System.out.println(a[3] - a[0] + a[2] - a[1]);

}

}

```

### QY10 区间表达

#### 题目描述

```

牛牛的老师给出了一个区间的定义:对于x ≤ y,[x, y]表示x到y之间(包括x和y)的所有连续整数集合。例如[3,3] = {3}, [4,7] = {4,5,6,7}.牛牛现在有一个长度为n的递增序列,牛牛想知道需要多少个区间并起来等于这个序列。

例如:

{1,2,3,4,5,6,7,8,9,10}最少只需要[1,10]这一个区间

{1,3,5,6,7}最少只需要[1,1],[3,3],[5,7]这三个区间

```

#### 输入描述:

```

输入包括两行,第一行一个整数n(1 ≤ n ≤ 50),

第二行n个整数a[i](1 ≤ a[i] ≤ 50),表示牛牛的序列,保证序列是递增的。

```

#### 输出描述:

```

输出一个整数,表示最少区间个数。

```

#### 示例1

#### 输入

```

5

1 3 5 6 7

```

#### 输出

```

3

```

```java

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

public class Main {

public static void main(String[] args) throws IOException {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

int n = Integer.parseInt(br.readLine().trim());

String[] str = br.readLine().split(" ");

int[] arr = new int[n];

for (int i = 0; i < n; i++) {

arr[i] = Integer.parseInt(str[i]);

}

int count = 0;

for (int i = 1; i < n; i++) {

if ((arr[i] - arr[i - 1]) != 1) count++;

}

System.out.println(count + 1);

}

}

```

### QY11 数字游戏

#### 题目描述

```

牛牛举办了一场数字游戏,有n个玩家参加这个游戏,游戏开始每个玩家选定一个数,然后将这个数写在纸上(十进制数,无前缀零),然后接下来对于每一个数字将其数位按照非递减顺序排列,得到新的数,新数的前缀零将被忽略。得到最大数字的玩家赢得这个游戏。

```

#### 输入描述:

```

输入包括两行,第一行包括一个整数n(1 ≤ n ≤ 50),即玩家的人数

第二行n个整数x[i](0 ≤ x[i] ≤ 100000),即每个玩家写下的整数。

```

#### 输出描述:

```

输出一个整数,表示赢得游戏的那个玩家获得的最大数字是多少。

```

#### 示例1

#### 输入

```

3

9638 8210 331

```

#### 输出

```

3689

```

```java

import java.util.\*;

import java.io.\*;

public class Main {

public static void main(String[] args) throws Exception {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

int n = Integer.parseInt(br.readLine());

String[] s = br.readLine().split(" ");

int[] a = new int[n];

for (int i = 0; i < n; i++) {

char[] t = s[i].toCharArray();

Arrays.sort(t);

String c = new String(t);

a[i] = Integer.parseInt(c);

}

Arrays.sort(a);

System.out.println(a[n - 1]);

}

}

```

### QY12 红和绿

#### 题目描述

```

牛牛有一些排成一行的正方形。每个正方形已经被染成红色或者绿色。牛牛现在可以选择任意一个正方形然后用这两种颜色的任意一种进行染色,这个正方形的颜色将会被覆盖。牛牛的目标是在完成染色之后,每个红色R都比每个绿色G距离最左侧近。牛牛想知道他最少需要涂染几个正方形。

如样例所示: s = RGRGR

我们涂染之后变成RRRGG满足要求了,涂染的个数为2,没有比这个更好的涂染方案。

```

#### 输入描述:

```

输入包括一个字符串s,字符串s长度length(1 ≤ length ≤ 50),其中只包括'R'或者'G',分别表示红色和绿色。

```

#### 输出描述:

```

输出一个整数,表示牛牛最少需要涂染的正方形数量

```

#### 示例1

#### 输入

```

RGRGR

```

#### 输出

```

2

```

```java

import java.lang.String;

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.io.IOException;

public class Main {

public static void main(String[] args) throws IOException {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

String str = br.readLine().trim();

int n = str.length();

int greenNum = 0;

int minCount = 0;

for (int i = 0; i < n; i++) {

if (str.charAt(i) == 'G') greenNum++;

// 计算是将当前的红变成绿少，还是把前面的绿都变成红少

else minCount = Math.min(greenNum, minCount + 1);

}

System.out.println(minCount);

}

}

```

### QY13 拼凑三角形

#### 题目描述

```

牛牛手中有三根木棍,长度分别是a,b,c。牛牛可以把任一一根木棍长度削短,牛牛的目标是让这三根木棍构成一个三角形,并且牛牛还希望这个三角形的周长越大越好。

```

#### 输入描述:

```

输入包括一行,一行中有正整数a, b, c(1 ≤ a, b, c ≤ 100), 以空格分割

```

#### 输出描述:

```

输出一个整数,表示能拼凑出的周长最大的三角形。

```

#### 示例1

#### 输入

```

1 2 3

```

#### 输出

```

5

```

```java

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.Arrays;

import java.util.HashSet;

import java.util.LinkedHashSet;

public class Main {

public static void main(String[] args) throws IOException {

// 牛牛手中有三根木棍,长度分别是a,b,c。牛牛可以把任一一根木棍长度削短,

// 牛牛的目标是让这三根木棍构成一个三角形,并且牛牛还希望这个三角形的周长越大越好。

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

String[] strs = br.readLine().split(" ");

int[] lines = new int[3];

for (int i = 0; i < 3; i++) lines[i] = Integer.parseInt(strs[i]);

Arrays.sort(lines);

while (lines[0] + lines[1] <= lines[2]) lines[2]--;

System.out.println(lines[0] + lines[1] + lines[2]);

}

}

```

### QY14 循环数比较

#### 题目描述

```

对于任意两个正整数x和k,我们定义repeat(x, k)为将x重复写k次形成的数,例如repeat(1234, 3) = 123412341234,repeat(20,2) = 2020.

牛牛现在给出4个整数x1, k1, x2, k2, 其中v1 = (x1, k1), v2 = (x2, k2),请你来比较v1和v2的大小。

```

#### 输入描述:

```

输入包括一行,一行中有4个正整数x1, k1, x2, k2(1 ≤ x1,x2 ≤ 10^9, 1 ≤ k1,k2 ≤ 50),以空格分割

```

#### 输出描述:

```

如果v1小于v2输出"Less",v1等于v2输出"Equal",v1大于v2输出"Greater".

```

#### 示例1

#### 输入

```

1010 3 101010 2

```

#### 输出

```

Equal

```

```java

import java.io.BufferedReader;

import java.io.InputStreamReader;

public class Main {

public static void main(String[] args) throws Exception {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

String[] strings = br.readLine().split(" ");

StringBuilder s1 = new StringBuilder(strings[0]);

String temp1 = s1.toString();

int count1 = Integer.parseInt(strings[1]);

StringBuilder s2 = new StringBuilder(strings[2]);

String temp2 = s2.toString();

int count2 = Integer.parseInt(strings[3]);

//生成

while (count1 > 1) {

s1.append(temp1);

count1--;

}

while (count2 > 1) {

s2.append(temp2);

count2--;

}

//比较

//如果长度不同，长度较大的更大

if (s1.length() < s2.length()) System.out.println("Less");

else if (s1.length() > s2.length()) System.out.println("Greater");

else {

if (s1.toString().compareTo(s2.toString()) < 0) System.out.println("Less");

else if (s1.toString().compareTo(s2.toString()) == 0) System.out.println("Equal");

else System.out.println("Greater");

}

}

}

```

### QY15 青草游戏

#### 题目描述

```

牛牛和羊羊都很喜欢青草。今天他们决定玩青草游戏。

最初有一个装有n份青草的箱子,牛牛和羊羊依次进行,牛牛先开始。在每个回合中,每个玩家必须吃一些箱子中的青草,所吃的青草份数必须是4的x次幂,比如1,4,16,64等等。不能在箱子中吃到有效份数青草的玩家落败。假定牛牛和羊羊都是按照最佳方法进行游戏,请输出胜利者的名字。

```

#### 输入描述:

```

输入包括t+1行。

第一行包括一个整数t(1 ≤ t ≤ 100),表示情况数.

接下来t行每行一个n(1 ≤ n ≤ 10^9),表示青草份数

```

#### 输出描述:

```

对于每一个n,如果牛牛胜利输出"niu",如果羊羊胜利输出"yang"。

```

#### 示例1

#### 输入

```

3

1

2

3

```

#### 输出

```

niu

yang

niu

```

```java

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.Scanner;

public class Main {

public static void main(String[] args) throws IOException {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

int num = Integer.parseInt(br.readLine());

int[] green = new int[num];

for (int i = 0; i < num; i++) green[i] = Integer.parseInt(br.readLine());

for (int i = 0; i < num; i++) {

if (green[i] % 5 == 1 || green[i] % 5 == 3 || green[i] % 5 == 4) System.out.println("niu");

else System.out.println("yang");

}

}

}

```

### QY16 无聊的牛牛和羊羊

#### 题目描述

```

牛牛和羊羊非常无聊.他们有n + m个共同朋友,他们中有n个是无聊的,m个是不无聊的。每个小时牛牛和羊羊随机选择两个不同的朋友A和B.(如果存在多种可能的pair(A, B),任意一个被选到的概率相同。),然后牛牛会和朋友A进行交谈,羊羊会和朋友B进行交谈。在交谈之后,如果被选择的朋友之前不是无聊会变得无聊。现在你需要计算让所有朋友变得无聊所需要的时间的期望值。

```

#### 输入描述:

```

输入包括两个整数n 和 m(1 ≤ n, m ≤ 50)

```

#### 输出描述:

```

输出一个实数,表示需要时间的期望值,四舍五入保留一位小数。

```

#### 示例1

#### 输入

```

2 1

```

#### 输出

```

1.5

```

```java

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

public class Main {

public static void main(String[] args) throws IOException {

BufferedReader bufr = new BufferedReader(new InputStreamReader(System.in));

String[] str = bufr.readLine().split(" ");

int n = Integer.parseInt(str[0]);

int m = Integer.parseInt(str[1]);

int s = m + n;

double f0 = 0, f1 = s / 2.0;

for (int k = 2; k <= m; k++) {

double cur = s \* (s - 1) / (1.0 \* k \* (2 \* s - k - 1)) + 2 \* (s - k) \* 1.0 / (2 \* s - k - 1) \* f1 + (k - 1) \* 1.0 / (2 \* s - k - 1) \* f0;

f0 = f1;

f1 = cur;

}

System.out.println(Math.round(f1 \* 10) / 10.0);

}

}

```

### QY17 幸运子序列

#### 题目描述

```

牛牛得到一个长度为n的整数序列V,牛牛定义一段连续子序列的幸运值为这段子序列中最大值和次大值的异或值(次大值是严格的次大)。牛牛现在需要求出序列V的所有连续子序列中幸运值最大是多少。请你帮帮牛牛吧。

```

#### 输入描述:

```

第一行一个整数n,即序列的长度。(2<= n <= 100000)

第二行n个数，依次表示这个序列每个数值V[i], (1 ≤ V[i] ≤ 10^8)且保证V[1]到V[n]中至少存在不同的两个值.

```

#### 输出描述:

```

输出一个整数,即最大的幸运值

```

#### 示例1

#### 输入

```

5

5 2 1 4 3

```

#### 输出

```

7

```

```java

import java.io.\*;

import java.util.\*;

public class Main {

public static void main(String[] args) throws IOException {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

int n = Integer.parseInt(br.readLine());

String[] str = br.readLine().split(" ");

int[] arr = new int[n];

int res = 0;

for (int i = 0; i < n; i++) {

arr[i] = Integer.parseInt(str[i]);

}

for (int i = 0; i < n; i++) {

for (int j = i - 1; j >= 0; j--) {

if (arr[j] > arr[i]) {

res = Math.max(res, arr[j] ^ arr[i]);

break;

}

}

for (int j = i + 1; j < n; j++) {

if (arr[j] > arr[i]) {

res = Math.max(res, arr[j] ^ arr[i]);

break;

}

}

}

System.out.println(res);

}

}

```

### QY18 缺失的括号

#### 题目描述

```

一个完整的括号字符串定义规则如下:

1、空字符串是完整的。

2、如果s是完整的字符串，那么(s)也是完整的。

3、如果s和t是完整的字符串，将它们连接起来形成的st也是完整的。

例如，"(()())", ""和"(())()"是完整的括号字符串，"())(", "()(" 和 ")"是不完整的括号字符串。

牛牛有一个括号字符串s,现在需要在其中任意位置尽量少地添加括号,将其转化为一个完整的括号字符串。请问牛牛至少需要添加多少个括号。

```

#### 输入描述:

```

输入包括一行,一个括号序列s,序列长度length(1 ≤ length ≤ 50).

s中每个字符都是左括号或者右括号,即'('或者')'.

```

#### 输出描述:

```

输出一个整数,表示最少需要添加的括号数

```

#### 示例1

#### 输入

```

(()(()

```

#### 输出

```

2

```

```java

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.nio.Buffer;

public class Main {

public static void main(String[] args) throws IOException {

BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

String s = reader.readLine().trim();

System.out.println(makeBracketStrComplete(s));

}

public static int makeBracketStrComplete(String s) {

int count = 0;

int result = 0;

for (int i = 0; i < s.length(); i++) {

if (s.charAt(i) == '(') {

count++;

} else if (s.charAt(i) == ')') {

if (--count < 0) {

result++;

count++;

}

}

}

return result + count;

}

}

```

### QY19 最后一位

#### 题目描述

```

牛牛选择了一个正整数X,然后把它写在黑板上。然后每一天他会擦掉当前数字的最后一位,直到他擦掉所有数位。 在整个过程中,牛牛会把所有在黑板上出现过的数字记录下来,然后求出他们的总和sum.

例如X = 509, 在黑板上出现过的数字依次是509, 50, 5, 他们的和就是564.

牛牛现在给出一个sum,牛牛想让你求出一个正整数X经过上述过程的结果是sum.

```

#### 输入描述:

```

输入包括正整数sum(1 ≤ sum ≤ 10^18)

```

#### 输出描述:

```

输出一个正整数,即满足条件的X,如果没有这样的X,输出-1。

```

#### 示例1

#### 输入

```

564

```

#### 输出

```

509

```

```java

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.io.IOException;

public class Main {

public static void main(String[] args) throws IOException {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

String strSum;

while ((strSum = br.readLine()) != null) System.out.println(solve(Long.parseLong(strSum)));

}

private static long solve(long sum) {

// 使用二分法搜索法，检验数mid的sum是否符合题意

long l = 0, r = sum;

while (l <= r) {

long mid = l + ((r - l) >>> 1);

long midRes = getSum(mid);

if (midRes == sum) return mid;

else if (midRes < sum) l = mid + 1;

else r = mid - 1;

}

return -1;

}

private static long getSum(long num) {

long sum = 0;

while (num != 0) {

sum += num;

num /= 10;

}

return sum;

}

}

```

### QY20 冒泡排序

#### 题目描述

```

牛牛学习了冒泡排序,并写下以下冒泡排序的伪代码,注意牛牛排序的数组a是从下标0开始的。

BubbleSort(a):

Repeat length(a)-1 times:

For every i from 0 to length(a) - 2:

If a[i] > a[i+1] then:

Swap a[i] and a[i+1]

牛牛现在要使用上述算法对一个数组A排序。

在排序前牛牛允许执行最多k次特定操作(可以不使用完),每次特定操作选择一个连续子数组,然后对其进行翻转,并且k次特定操作选择的子数组不相交。

例如A = {1, 2, 3, 4, 5, 6, 7}, k = 1,如果牛牛选择的子数组是[2,4](注意下标从0开始),那么翻转之后的数组变为A = {1, 2, 5, 4, 3, 6, 7}。

牛牛知道冒泡排序的效率一定程度上取决于Swap操作次数,牛牛想知道对于一个数组A在进行k次特定操作之后,再进行上述冒泡排序最少的Swap操作次数是多少?

```

#### 输入描述:

```

输入包括两行,第一行包括两个正整数n和k(2 ≤ n ≤ 50, 1 ≤ k ≤ 50),表示数组的长度和允许最多的特定操作次数。

第二行n个正整数A[i](1 ≤ A[i] ≤ 1000),表示数组内的元素,以空格分割。

```

#### 输出描述:

```

输出一个整数,表示在执行最多k次特定操作之后,对数组进行上述冒泡排序需要的Swap操作次数。

```

#### 示例1

#### 输入

```

3 2

2 3 1

```

#### 输出

```

1

```

```java

import java.io.\*;

public class Main {

public static void main(String[] args) throws IOException {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

String[] s = br.readLine().split(" ");

int n = Integer.parseInt(s[0]);

int k = new Integer(s[1]);

s = br.readLine().split(" ");

int[] nums = new int[n];

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

nums[i] = new Integer(s[i]);

}

int[][] dp = new int[n + 1][k + 1];

//dp[i][j]表示前i个数进行j次反转之后最多能消除多少个逆序对

for (int i = 2; i <= n; i++) {

for (int j = 1; j <= k; j++) {

int tmp = Integer.MIN\_VALUE;

for (int t = i - 1; t >= 1; t--) {

tmp = Math.max(beforeReverse(nums, t - 1, i - 1) - afterReverse(nums, t - 1, i - 1) + dp[t - 1][j - 1], tmp);

}

dp[i][j] = Math.max(dp[i - 1][j], tmp);

}

}

System.out.println(beforeReverse(nums, 0, n - 1) - dp[n][k]);

}

//求出反转之后还有多少个逆序对，

public static int afterReverse(int[] nums, int left, int right) {

int num = 0;

for (int i = right; i > left; i--) {

for (int j = i - 1; j >= left; j--) {

if (nums[j] < nums[i]) num++;

}

}

return num;

}

//求出在某个范围内有多少个逆序对

public static int beforeReverse(int[] nums, int left, int right) {

int num = 0;

for (int i = left; i < right; i++) {

for (int j = i + 1; j <= right; j++) {

if (nums[i] > nums[j]) num++;

}

}

return num;

}

}

```

### QY21 括号匹配深度

#### 题目描述

```

一个合法的括号匹配序列有以下定义:

1、空串""是一个合法的括号匹配序列

2、如果"X"和"Y"都是合法的括号匹配序列,"XY"也是一个合法的括号匹配序列

3、如果"X"是一个合法的括号匹配序列,那么"(X)"也是一个合法的括号匹配序列

4、每个合法的括号序列都可以由以上规则生成。

例如: "","()","()()","((()))"都是合法的括号序列

对于一个合法的括号序列我们又有以下定义它的深度:

1、空串""的深度是0

2、如果字符串"X"的深度是x,字符串"Y"的深度是y,那么字符串"XY"的深度为max(x,y) 3、如果"X"的深度是x,那么字符串"(X)"的深度是x+1

例如: "()()()"的深度是1,"((()))"的深度是3。牛牛现在给你一个合法的括号序列,需要你计算出其深度。

```

#### 输入描述:

```

输入包括一个合法的括号序列s,s长度length(2 ≤ length ≤ 50),序列中只包含'('和')'。

```

#### 输出描述:

```

输出一个正整数,即这个序列的深度。

```

#### 示例1

#### 输入

```

(())

```

#### 输出

```

2

```

```java

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Scanner in = new Scanner(System.in);

String str = in.nextLine();

int stack = 0, max = 0;

for (int i = 0; i < str.length(); i++) {

if (str.charAt(i) == '(') stack++;

if (str.charAt(i) == ')') stack--;

max = Math.max(max, stack);

}

System.out.println(max);

in.close();

}

}

```

### QY22 奶牛编号

#### 题目描述

```

牛牛养了n只奶牛,牛牛想给每只奶牛编号,这样就可以轻而易举地分辨它们了。 每个奶牛对于数字都有自己的喜好,第i只奶牛想要一个1和x[i]之间的整数(其中包含1和x[i])。

牛牛需要满足所有奶牛的喜好,请帮助牛牛计算牛牛有多少种给奶牛编号的方法,输出符合要求的编号方法总数。

```

#### 输入描述:

```

输入包括两行,第一行一个整数n(1 ≤ n ≤ 50),表示奶牛的数量 第二行为n个整数x[i](1 ≤ x[i] ≤ 1000)

```

#### 输出描述:

```

输出一个整数,表示牛牛在满足所有奶牛的喜好上编号的方法数。因为答案可能很大,输出方法数对1,000,000,007的模。

```

#### 示例1

#### 输入

```

4

4 4 4 4

```

#### 输出

```

24

```

```java

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.Arrays;

/\*\*

\* 解决方法：

\* 给奶牛所希望的数字排个序，然后第一个奶牛有X[0]种方法，第二个奶牛有X[1]-1种方法。。。

\*/

public class Main {

private static final long MOD = 1000000007;

public static void main(String[] args) throws IOException {

BufferedReader bf = new BufferedReader(new InputStreamReader(System.in));

long n = Integer.parseInt(bf.readLine());

String[] str = bf.readLine().split(" ");

int[] num = new int[str.length];

for (int i = 0; i < n; i++) {

num[i] = Integer.parseInt(str[i]);

}

Arrays.sort(num);

long res = 1;

for (int j = 0; j < str.length; j++) {

res = res \* ((num[j] - j) % MOD);

res = res % MOD;

}

System.out.println(res);

}

}

```

### QY23 平方串

#### 题目描述

```

如果一个字符串S是由两个字符串T连接而成,即S = T + T, 我们就称S叫做平方串,例如"","aabaab","xxxx"都是平方串.

牛牛现在有一个字符串s,请你帮助牛牛从s中移除尽量少的字符,让剩下的字符串是一个平方串。换句话说,就是找出s的最长子序列并且这个子序列构成一个平方串。

```

#### 输入描述:

```

输入一个字符串s,字符串长度length(1 ≤ length ≤ 50),字符串只包括小写字符。

```

#### 输出描述:

```

输出一个正整数,即满足要求的平方串的长度。

```

#### 示例1

#### 输入

```

frankfurt

```

#### 输出

```

4

```

```java

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

public class Main {

public static void main(String[] args) throws IOException {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

String s = br.readLine();

int res = 0;

for (int i = 1; i < s.length(); i++) {

res = Math.max(res, lcs(s.substring(0, i), s.substring(i)));

}

System.out.println(res \* 2);

}

//求s1和s2的最长公共子序列的长度

private static int lcs(String s1, String s2) {

//dp[i + 1][j + 1]: 以s1[i]、s2[j]为结尾的最长公共子序列的长度

int[][] dp = new int[s1.length() + 1][s2.length() + 1];

for (int i = 0; i < s1.length(); i++) {

for (int j = 0; j < s2.length(); j++) {

if (s1.charAt(i) == s2.charAt(j)) dp[i + 1][j + 1] = dp[i][j] + 1;

else dp[i + 1][j + 1] = Math.max(dp[i + 1][j], dp[i][j + 1]);

}

}

return dp[s1.length()][s2.length()];

}

}

```