# ****【算法题】最长子字符串的长度****

## **题目描述**

****给你一个字符串 s，字符串s首尾相连成一个环形 ，请你在环中找出 'o' 字符出现了偶数次最长子字符串的长度。****

## **题目要求**

**输入是一串小写字母组成的字符串  
输出是一个整数**

#### **示例1**

输入 ‘alolobo’  
输出 6  
说明：  
最长子字符串之一是 "alolob"，它包含'o' 2个。

#### **示例2**

输入 ‘looxdolx’  
输出 7  
说明：  
最长子字符串是 "oxdolxl"，由于是首尾连接在一起的，所以最后一个 'x' 和开头的 'l'是连接在一起的，此字符串包含 2 个'o' 。

#### **示例3**

输入 ‘bcbcbc’  
输出 6  
说明：  
这个示例中，字符串 "bcbcbc" 本身就是最长的，因为 'o' 都出现了 0 次。

#### **备注**

1 <= s.length <= 5 x 10^5  
s 只包含小写英文字母

## **JAVA解法**

/\*\*

\* @since 2022-04

\*/

public class LongestSubString {

public static void main(String[] args) {

Scanner in = new Scanner(System.in);

String input = in.nextLine();

char[] chrs = input.toCharArray();

int len = chrs.length;

int num = 0;

for (char chr : chrs) {

if (chr == 'o') {

num += 1;

}

}

if (num % 2 == 0) {

System.out.println(len);

} else {

//出现了奇数次

System.out.println(len - 1);

}

}

}

# ****【算法题】素数之积****

## **题目**

****RSA加密算法在网络安全世界中无处不在，它利用了极大整数因数分解的困难度，数据越大，安全系数越高，给定一个32位正整数， 请对其进行因数分解，找出是哪两个素数的乘积。****

## **描述:**

输入：一个正整数num（0 < num <= 2147483647）  
输出：如果成功找到，以单个空格分割，从小到大输出两个素数，分解失败，请输出-1 -1

### **示例1**

输入 15  
输出 3 5  
说明 因数分解后，找到两个素数3和5，使得3\*5=15，按从小到大排列后，输出3 5

### **示例2**

输入 27  
输出 -1 -1  
说明 通过因数分解，找不到任何素数，使得他们的乘积为27，输出-1 -1

## **知识点补充**

1. ****如果一个数字不是素数 那它除了1和他本身一定还有别的约数****
2. ****如果一个数的约数在其开平方的右边 则必然会存在一个约数在其开平方的左边****
3. ****判断一个数是否为质数 只需要观察在其2 到 开平方数中间是否含有约数即可****  
   比如说 判断16是否为质数 我们去找16的约数时 判断它的范围 只需要找到  
   16的开平方数就可以了 而不必一直找到<16 或者<= 16/2

## **JAVA解法**

import java.util.Scanner;

/\*\*

\* @since 2022-04-11

\*/

public class ProductOfPrime {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int num = scanner.nextInt();

int a = -1;

int b = -1;

// 计算输入数字的平方根，原因如上

double sqrt = Math.sqrt(num);

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

// 判断是否为素数

if (!isPrime(i)) {

continue;

}

// 判断能否整除

if (num % i != 0) {

continue;

}

// 判断另一个乘数是否为素数

if (!isPrime(num / i)) {

continue;

}

// 结果赋值

a = i;

b = num / i;

}

System.out.println(a + " " + b);

}

/\*\*

\* 判断是否是素数

\*

\* @param arg 入参

\* @return bool

\*/

public static boolean isPrime(int arg) {

if (arg <= 1) {

return false;

}

// 计算输入数字的平方根，原因如上

double sqrt = Math.sqrt(arg);

for (int j = 2; j <= sqrt; j++) {

if (arg % j == 0) {

return false;

}

}

return true;

}

}

# ****【算法题】拼接URL****

## **题目描述**

****给定一个URL前缀和URL后缀,通过,分割， 需要将其连接为一个完整的URL，如果前缀结尾和后缀开头都没有/，需要自动补上/连接符，如果前缀结尾和后缀开头都为/，需要自动去重  
约束：不用考虑前后缀URL不合法情况****

## **题目要求**

**输入url前缀(一个长度小于100的字符串) url后缀(一个长度小于100的字符串)  
输出拼接后的url**

#### **示例1**

输入 /acm,/bb  
输出 /acm/bb

#### **示例2**

输入 /abc/,/bcd  
输出 /abc/bcd

#### **示例3**

输入 /acd,bef  
输出 /acd/bef

#### **示例4**

输入 ,  
输出 /

## **JAVA解法**

import java.util.Scanner;

/\*\*

\* @since 2022年4月13日

\*/

public class ConcatURL {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

String line = scanner.nextLine();

String[] strings = line.replace("/", "").split(",");

StringBuilder stringBuilder = new StringBuilder();

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

stringBuilder.append("/").append(strings[i]);

}

System.out.println(stringBuilder.toString());

}

}

# ****【算法题】转骰子问题****

## **题目描述**

****骰子是一个立方体，每个面一个数字，初始为左1，右2，前3（观察者方向），后4，上5，下6，用123456表示这个状态。放置在平面上，可以向左翻转（用L表示向左翻转1次），可以向右翻转（用R表示向右翻转1次），可以向前翻转（用F表示向前翻转1次），可以向后翻转（用B表示向后翻转一次），可以逆时针旋转（用A表示逆时针旋转90度），可以逆时针旋转（用C表示顺时针旋转90度），现从123456这个初始状态开始，根据输入的动作序列，计算得到最终的状态。****

## **题目要求**

**输入一行，为只包含LRFBAC的字母序列，最大长度为50，可重复  
输出最终状态**

#### **示例1**

输入 LB  
输出 5 6 1 2 3 4  
说明 骰子先向左翻转，在想后翻转，那么此时的状态561234

#### **示例2**

输入 LR  
输出 1 2 3 4 5 6  
说明 骰子先向左翻转，在想右翻转回到了原始状态，那么此时的状态123456

#### **示例3**

输入 FCR  
输出 3 4 2 1 5 6  
说明 ：  
骰子先向前翻转，状态变为125643  
再向顺时针旋转回90度，状态变成为562143  
再向右翻转，状态就变成为342156

### **TIPS**

* 先观察一下，变化规则，初始状态{1,2,3,4,5,6}
  + 'L': {5,6,3,4,2,1}；索引赋值：[0]=[4]；[1]=[5]；[2]=[2]；[3]=[3]；[4]=[1]；[5]=[0]
  + 'R': {6,5,3,4,1,2}；索引赋值：[0]=[5]；[1]=[4]；[2]=[2]；[3]=[3]；[4]=[0]；[5]=[1]
  + 'F': {1,2,5,6,4,3}；索引赋值：[0]=[0]；[1]=[1]；[2]=[4]；[3]=[5]；[4]=[3]；[5]=[2]
  + 'B': {1,2,6,5,3,4}；索引赋值：[0]=[0]；[1]=[1]；[2]=[5]；[3]=[4]；[4]=[2]；[5]=[3]
  + 'A': {4,3,1,2,5,6}；索引赋值：[0]=[3]；[1]=[2]；[2]=[0]；[3]=[1]；[4]=[4]；[5]=[5]
  + 'C': {3,4,2,1,5,6}；索引赋值：[0]=[2]；[1]=[3]；[2]=[1]；[3]=[0]；[4]=[4]；[5]=[5]

## **JAVA解法**

import java.util.Arrays;

import java.util.Scanner;

/\*\*

\* @since 2022年4月14日

\*/

public class RevolveDice {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

String line = scanner.nextLine();

int[] init = {1, 2, 3, 4, 5, 6};

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

char charAt = line.charAt(i);

if ('L' == charAt) {

int temp = init[0];

init[0] = init[4];

init[4] = init[1];

init[1] = init[5];

init[5] = temp;

continue;

}

if ('R' == charAt) {

int temp = init[0];

init[0] = init[5];

init[5] = init[1];

init[1] = init[4];

init[4] = temp;

continue;

}

if ('F' == charAt) {

int temp = init[2];

init[2] = init[4];

init[4] = init[3];

init[3] = init[5];

init[5] = temp;

continue;

}

if ('B' == charAt) {

int temp = init[2];

init[2] = init[5];

init[5] = init[3];

init[3] = init[4];

init[4] = temp;

continue;

}

if ('A' == charAt) {

int temp = init[0];

init[0] = init[3];

init[3] = init[1];

init[1] = init[2];

init[2] = temp;

continue;

}

if ('C' == charAt) {

int temp = init[0];

init[0] = init[2];

init[2] = init[1];

init[1] = init[3];

init[3] = temp;

}

}

String s = Arrays.toString(init);

System.out.println(s.substring(1, s.length() - 1).replace(",", ""));

}

}

# ****【算法题】字符串分割****

## **题目**

****给定非空字符串在s，将该字符串分割成一些子串，使每个子串的ASCII码值的和均为水仙花数。  
1、若分割不成功则返回 0  
2、若分割成功且分割结果不唯一则返回-1  
3、若分割成功且分割结果唯一，则返回分割后的子串数目****  
**备注：“水仙花数”是指一个三位数，每位上数字的立方和等于该数字本身，如 371是“水仙花数”，因为:371=3^3+7^3+1^3。**

## **描述:**

输入：输入字符串的最大长度为 200  
输出：返回相应的结果

### **示例1**

输入 abc  
输出 0  
说明 分割不成功

### **示例2**

输入 f3@d5a8  
输出 -1  
说明 分割成功但分割结果不唯一，可以分割为两组，  
一组：""f3"" 和 ""@d5a8""  
另一组: ""f3@d5"" 和 ""a8""

### **示例3**

输入 AXdddF  
输出 2  
说明 分割成功且结果唯一，可以分割为AX""(153) 和""dddF""(370)

## **JAVA解法**

public class SubString {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

String line = scanner.nextLine();

// 超过长度200，返回分割不成功

if (line.length() > 200) {

System.out.println(0);

}

StringBuilder sb = new StringBuilder();

ArrayList<String> strings = new ArrayList<>();

char[] chars = line.toCharArray();

digui(chars, sb, strings, 0);

// 分割结果不唯一打印-1

if (strings.size() > 1) {

System.out.println(strings);

System.out.println(-1);

}

// 分割结果唯一打印分割的子串个数

if (strings.size() == 1) {

System.out.println(strings);

System.out.println(strings.get(0).split(" ").length);

}

// 分割没有结果打印0

if (strings.size() == 0) {

System.out.println(0);

}

}

public static void digui(char[] chars, StringBuilder sb, ArrayList<String> strings, int startIndex) {

int num = 0;

StringBuilder sb\_temp = new StringBuilder();

for (int i = startIndex; i < chars.length; i++) {

char aChar = chars[i];

num += aChar;

sb\_temp.append(aChar);

if (!isNar(num)) {

continue;

}

StringBuilder sb\_copy = new StringBuilder(sb);

sb\_copy.append(sb\_temp).append(" ");

if (i == chars.length - 1) {

strings.add(sb\_copy.toString().trim());

return;

}

digui(chars, sb\_copy, strings, i + 1);

}

}

public static boolean isNar(int num) {

if (num < 100 || num > 999) return false;

int bai = num / 100;

int shi = (num % 100) / 10;

int ge = (num % 100) % 10;

int v = (int) (Math.pow(bai, 3) + Math.pow(shi, 3) + Math.pow(ge, 3));

return v == num;

}

}

# ****【算法题】统计射击比赛成绩****

## **题目描述**

****给定一个射击比赛成绩单，包含多个选手若干次射击的成绩分数，请对每个选手按其最高三个分数之和进行降序排名，输出降序排名后的选手ID序列。****

## **题目条件**

* 一个选手可以有多个射击成绩的分数，且次序不固定
* 如果一个选手成绩少于3个，则认为选手的所有成绩无效，排名忽略该选手

如果选手的成绩之和相等，则相等的选手按照其ID降序排列

## **题目要求**

输入第一行，一个整数N，表示该场比赛总共进行了N次射击，产生N个成绩分数（2<=N<=100）

* 输入第二行，一个长度为N整数序列，表示参与每次射击的选手id（0<=id<=99）

输入第三行，一个长度为N整数序列，表示参与每次射击选手对应的成绩（0<=成绩<=100）

输出 符合题设条件的降序排名后的选手ID序列

#### **示例1**

输入：  
13  
3,3,7,4,4,4,4,7,7,3,5,5,5  
53,80,68,24,39,76,66,16,100,55,53,80,55  
输出：5,3,7,4  
说明:  
该场射击比赛进行了13次，参赛的选手为{3,4,5,7}  
3号选手成绩53,80,55 最高三个成绩的和为188  
4号选手成绩24,39,76,66 最高三个成绩的和为205  
5号选手成绩53,80,55 最高三个成绩的和为188  
7号选手成绩68,16,100 最高三个成绩的和为184  
比较各个选手最高3个成绩的和，有4号>3号=5号>7号，由于3号和5号成绩相等，且id 5>3，所以输出 7,5,3,4

## **JAVA解法**

import java.util.\*;

import java.util.stream.Collectors;

/\*\*

\* @since 2022年4月18日

\*/

public class ShootCompetition {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// 总射击次数

int count = scanner.nextInt();

// 射击选手的ID

List<Integer> ids = Arrays.stream(scanner.nextLine().split(",")).map(Integer::parseInt).collect(Collectors.toList());

// 射击选手对应的成绩

List<Integer> scores = Arrays.stream(scanner.nextLine().split(",")).map(Integer::parseInt).collect(Collectors.toList());

HashMap<Integer, List<Integer>> id\_scores = new HashMap<>();

// 分配选手的各自的分数

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

Integer id = ids.get(i);

Integer score = scores.get(i);

List<Integer> list = id\_scores.getOrDefault(id, new LinkedList<>());

list.add(score);

id\_scores.put(id, list);

}

StringBuilder builder = new StringBuilder();

id\_scores.entrySet()

.stream()

.filter(x -> x.getValue().size() >= 3)

.sorted((o1, o2) -> {

Integer sum1 = sum(o1.getValue());

Integer sum2 = sum(o2.getValue());

if (Objects.equals(sum1, sum2)) {

return o2.getKey() - o1.getKey();

} else {

return sum2 - sum1;

}

})

.map(Map.Entry::getKey)

.forEach(x -> builder.append(x).append(","));

System.out.println(builder.substring(0, builder.length() - 1));

}

private static Integer sum(List<Integer> list) {

list.sort(Integer::compareTo);

int sum = 0;

for (int i = list.size() - 1; i >= list.size() - 3; i--) {

sum += list.get(i);

}

return sum;

}

}

# ****【算法题】数组连续和****

## **题目描述**

****给定一个含有N个正整数的数组, 求出有多少个连续区间（包括单个正整数）, 它们的和大于等于x。****

## **题目要求**

* 第一行两个整数N x（0 < N <= 100000, 0 <= x <= 10000000)

第二行有N个正整数（每个正整数小于等于100)

输出 一个整数，表示所求的个数

### **示例1**

输入  
3 7  
3 4 7  
输出  
4  
说明  
3+4 4+7 3+4+7 7这四组数据都是大于等于7的，所以答案为4

### **示例2**

输入  
10 10000000  
1 2 3 4 5 6 7 8 9 10  
输出  
0

## **JAVA解法**

import java.util.Scanner;

/\*\*

\* 标题：数组连续和 | 时间限制：1秒 | 内存限制：65536K | 语言限制：不限

\* 给定一个含有N个正整数的数组, 求出有多少个连续区间（包括单个正整数）, 它们的和大于等于x。

\*

\* @since 2022年4月18日

\*/

public class ContinuousArraySum {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

String line = scanner.nextLine();

String numStr = scanner.nextLine();

// 取出两个数N x

int N = Integer.parseInt(line.split(" ")[0]);

int x = Integer.parseInt(line.split(" ")[1]);

// 得到所有的正整数

int[] ints = new int[N];

String[] strings = numStr.split(" ");

for (int i = 0; i < numStr.split(" ").length; i++) {

ints[i] = Integer.parseInt(strings[i]);

}

int count = 0;

int sum = 0;

// 比较各种连续和

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

for (int i = j; i < N; i++) {

sum += ints[i];

if (sum >= x) {

count++;

}

}

sum = 0;

}

System.out.println(count);

}

}

# ****【算法题】分班****

## **题目描述**

****幼儿园两个班的小朋友在排队时混在了一起，每位小朋友都知道自己是否与前面一位小朋友是否同班，请你帮忙把同班的小朋友找出来。  
小朋友的编号为整数，与前一位小朋友同班用Y表示，不同班用N表示。****

## **题目要求**

输入为：空格分开的小朋友编号和是否同班标志。  
比如：6/N 2/Y 3/N 4/Y，表示共4位小朋友，2和6同班，3和2不同班，4和3同班。  
其中，小朋友总数不超过999，每个小朋友编号大于0，小于等于999。  
不考虑输入格式错误问题。

输出为：两行，每一行记录一个班小朋友的编号，编号用空格分开。且：

* 编号需要按照大小升序排列，分班记录中第一个编号小的排在第一行。
* 若只有一个班的小朋友，第二行为空行。
* 若输入不符合要求，则直接输出字符串ERROR。

#### **示例1**

输入  
1/N 2/Y 3/N 4/Y  
输出  
1 2  
3 4  
说明  
2的同班标记为Y，因此和1同班。  
3的同班标记为N，因此和1、2不同班。  
4的同班标记为Y，因此和3同班。  
所以1、2同班，3、4同班，输出为  
1 2  
3 4

## **JAVA解法**

import java.util.ArrayList;

import java.util.Comparator;

import java.util.Scanner;

/\*\*

\* @since 2022年4月18日

\*/

public class DivideClass {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

String teamStr = scanner.nextLine();

String[] childs = teamStr.split(" ");

if (childs.length > 999) {

System.out.println("ERROR");

}

// 分开两个班级

ArrayList<String> classA = new ArrayList<>();

ArrayList<String> classB = new ArrayList<>();

String classNum = "";

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

if (i == 0) {

String first = childs[0];

classA.add(first.split("/")[0]);

classNum = "A";

continue;

}

String follow = childs[i];

boolean isSameClass = follow.endsWith("Y");

String stu = follow.split("/")[0];

if ("A".equals(classNum)) {

if (isSameClass) {

classA.add(stu);

} else {

classB.add(stu);

}

classNum = isSameClass ? "A" : "B";

} else {

if (isSameClass) {

classB.add(stu);

} else {

classA.add(stu);

}

classNum = isSameClass ? "B" : "A";

}

}

classA.sort(Comparator.comparingInt(Integer::parseInt));

classB.sort(Comparator.comparingInt(Integer::parseInt));

System.out.println(classA.toString().replace("[", "").replace("]", "").replace(",", " "));

System.out.println(classB.toString().replace("[", "").replace("]", "").replace(",", " "));

}

}

# ****【算法题】流水线****

## **题目描述**

****一个工厂有m条流水线，来并行完成n个独立的作业，该工厂设置了一个调度系统，在安排作业时，总是优先执行处理时间最短的作业。  
现给定流水线个数m，需要完成的作业数n, 每个作业的处理时间分别为t1,t2…tn。请你编程计算处理完所有作业的耗时为多少？  
当n>m时，首先处理时间短的m个作业进入流水线，其他的等待，当某个作业完成时，依次从剩余作业中取处理时间最短的进入处理。****

## **题目要求**

输入描述:  
第一行为2个整数（采用空格分隔），分别表示流水线个数m和作业数n；  
第二行输入n个整数（采用空格分隔），表示每个作业的处理时长t1,t2…tn。  
(0< m，n<100，0<t1,t2…tn<100）  
注：保证输入都是合法的。

输出描述:  
输出处理完所有作业的总时长

#### **示例1**

输入  
3 5  
8 4 3 2 10  
输出  
13  
说明

1. 先安排时间为2、3、4的3个作业。
2. 第一条流水线先完成作业，然后调度剩余时间最短的作业8。
3. 第二条流水线完成作业，然后调度剩余时间最短的作业10。
4. 总工耗时就是第二条流水线完成作业的时间13（3+10）。

## **JAVA解法**

import java.util.\*;

import java.util.stream.Collectors;

/\*\*

\* @since 2022年4月19日

\*/

public class AssembleLine {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

String m\_n = scanner.nextLine();

String timeStr = scanner.nextLine();

// 取出m条流水线 n个作业数

int m = Integer.parseInt(m\_n.split(" ")[0]);

int n = Integer.parseInt(m\_n.split(" ")[1]);

// 所有作业数的时间集合

List<Integer> ts = Arrays.stream(timeStr.split(" ")).map(Integer::parseInt).sorted(Comparator.comparingInt(o -> o)).collect(Collectors.toList());

// 由于他们可以同时开工

// 如果流水线多于任务数，那么就取时间最上的那个任务时间

if (n <= m) {

System.out.println(ts.get(n - 1));

}

ArrayList<Integer> res = new ArrayList<>();

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

res.add(ts.get(i));

}

for (int i = m; i < ts.size(); i++) {

int index = i % m; // 换个方式

// Integer min = res.stream().sorted().iterator().next(); // 这一快很有意思，无限制取，和平常了解的迭代器不同

// int index = res.indexOf(min);

res.set(index, res.get(index) + ts.get(i));

}

Integer maxTime = res.stream().max(Integer::compareTo).get();

System.out.println(maxTime);

}

}

# ****【算法题】服务器广播****

## **题目描述**

****服务器连接方式包括直接连接，间接连接。A和B直接连接，B和C直接连接，则A和C间接连接。直接连接和间接连接都可以发送广播。  
给出一个NN数组，代表N个服务器，matrix[i][j] == 1，则代表i和j直接连接；不等于1时，代表i和j不直接连接。matrix[i][i] == 1，即自己和自己直接连接。matrix[i][j] == matrix[j][i]。  
计算初始需要给几台服务器广播，才可以使每个服务器都收到广播。****

## **题目要求**

输入描述:  
输入为N行，每行有N个数字，为0或1，由空格分隔，构成N\*N的数组，N的范围为 1<=N<=40

输出描述:  
输出一个数字，为需要广播的服务器的数量

#### **示例1**

输入  
1 0 0  
0 1 0  
0 0 1  
输出  
3  
说明  
3台服务器互不连接，所以需要分别广播这3台服务器

#### **示例2**

输入  
1 1  
1 1  
输出  
1  
说明  
2台服务器相互连接，所以只需要广播其中一台服务器

## **JAVA解法**

import java.util.\*;

/\*\*

\* @since 2022年4月20日

\*/

public class ServerBroadcast {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

String[] servers = scanner.nextLine().split(" ");

int N = servers.length;

int count = 0;

int[][] arrServer = new int[N][N];

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

arrServer[count][i] = Integer.parseInt(servers[i]);

}

while (++count < N) {

servers = scanner.nextLine().split(" ");

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

arrServer[count][i] = Integer.parseInt(servers[i]);

}

}

// 打印一下二维数组

// System.out.println(Arrays.deepToString(arrServer));

Map<Integer, Set<Integer>> map = new HashMap<>();

//把直接相连的服务器存储起来

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

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

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

boolean isSame = arrServer[i][j] == 1;

if (isSame) {

set.add(j);

map.put(i, set);

}

}

}

//取出间接相连的

for (Map.Entry<Integer, Set<Integer>> entry : map.entrySet()) {

int key = entry.getKey();

Set<Integer> broadcastSet = entry.getValue();

for (Integer broadcast : broadcastSet) {

if (key != broadcast) {

map.put(broadcast, broadcastSet);

}

}

}

Set<Set<Integer>> ret = new HashSet<>();

for (Map.Entry<Integer, Set<Integer>> entry : map.entrySet()) {

ret.add(entry.getValue());

}

System.out.println(ret.size());

}

}

# ****【算法题】找终点****

## **题目描述**

****给定一个正整数数组，设为nums，最大为100个成员，求从第一个成员开始，正好走到数组最后一个成员，所使用的最少步骤数。  
要求：  
1、第一步必须从第一元素开始，且1<=第一步的步长<len/2;（len为数组的长度，需要自行解析）。  
2、从第二步开始，只能以所在成员的数字走相应的步数，不能多也不能少, 如果目标不可达返回-1，只输出最少的步骤数量。  
3、只能向数组的尾部走，不能往回走。****

## **题目要求**

输入描述:  
由正整数组成的数组，以空格分隔，数组长度小于100，请自行解析数据数量。

输出描述:  
正整数，表示最少的步数，如果不存在输出-1

#### **示例1**

输入  
7 5 9 4 2 6 8 3 5 4 3 9  
输出  
2  
说明  
第一步： 第一个可选步长选择2，从第一个成员7开始走2步，到达9；第二步： 从9开始，经过自身数字9对应的9个成员到最后。

#### **示例2**

输入  
1 2 3 7 1 5 9 3 2 1  
输出  
-1

## **JAVA解法**

import java.util.Scanner;

/\*\*

\* @since 2022年4月21日

\*/

public class GetFinal {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

String line = scanner.nextLine();

String[] strArr = line.split(" ");

int stepSmallest = 0;

for (int i = 1; i < strArr.length / 2; i++) {

// 第一步走到索引 i

int index = i;

int step = 0;

while (true) {

if (index >= strArr.length) {

stepSmallest = -1;

break;

}

step++;

int current = Integer.parseInt(strArr[index]);

if (strArr.length - 1 == index) {

if (stepSmallest <= 0) {

stepSmallest = step;

} else if (stepSmallest <= step) {

} else {

stepSmallest = step;

}

break;

}

index += current;

}

}

System.out.println(stepSmallest);

}

}

# ****【算法题】火星文计算****

## **题目描述**

****已知火星人使用的运算符为#、$，其与地球人的等价公式如下：  
x#y = 2**x+3**y+4  
x$y = 3\*x+y+2  
1、其中x、y是无符号整数  
2、地球人公式按C语言规则计算  
3、火星人公式中，$的优先级高于#，相同的运算符，按从左到右的顺序计算  
现有一段火星人的字符串报文，请你来翻译并计算结果。****

## **题目要求**

* 输入描述:  
  火星人字符串表达式（结尾不带回车换行）  
  输入的字符串说明： 字符串为仅由无符号整数和操作符（#、$）组成的计算表达式。例如：123#4$5#67$78。
* 用例保证字符串中，操作数与操作符之间没有任何分隔符。
* 用例保证操作数取值范围为32位无符号整数。
* 保证输入以及计算结果不会出现整型溢出。
* 保证输入的字符串为合法的求值报文，例如：123#4$5#67$78

保证不会出现非法的求值报文，例如类似这样字符串：**\***

* + - #4$5 //缺少操作数
    - 4$5# //缺少操作数
    - 4#$5 //缺少操作数
    - 4 $5 //有空格
    - 3+4-5\*6/7 //有其它操作符
    - 12345678987654321$54321 //32位整数计算溢出

输出描述:  
根据输入的火星人字符串输出计算结果（结尾不带回车换行）

#### **示例1**

输入  
7#6$5#12  
输出  
226

示例：

7#6$5#12

=7#(3\*6+5+2)#12

=7#25#12

=(2\*7+3\*25+4)#12

=93#12

=2\*93+3\*12+4

=226

## **JAVA解法**

import java.util.Scanner;

/\*\*

\* @since 2022年4月22日

\*/

public class MartianText {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

String line = scanner.nextLine();

String[] arr\_jin = line.split("#");

int sum = dealWithArr\_jin(arr\_jin[0]);

for (int i = 1; i < arr\_jin.length; i++) {

sum = calculate\_jin(sum, dealWithArr\_jin(arr\_jin[i]));

}

System.out.print(sum); // 注意！！！不带回车

}

private static Integer dealWithArr\_jin(String arr\_jin) {

String[] arr\_dollar = arr\_jin.split("\\$");

if (arr\_dollar.length < 2) {

return Integer.parseInt(arr\_jin);

}

int temp = Integer.parseInt(arr\_dollar[0]);

for (int j = 1; j < arr\_dollar.length; j++) {

temp = calculate\_dollar(temp, Integer.parseInt(arr\_dollar[j]));

}

return temp;

}

private static int calculate\_dollar(int x, int y) {

return 3 \* x + y + 2;

}

private static int calculate\_jin(int x, int y) {

return 2 \* x + 3 \* y + 4;

}

}

# ****【算法题】找单词****

## **题目描述**

给一个字符串和一个二维字符数组，如果该字符串存在于该数组中，则按字符串的字符顺序输出字符串每个字符所在单元格的位置下标字符串，如果找不到返回字符串"N"。

1. 需要按照字符串的字符组成顺序搜索，且搜索到的位置必须是相邻单元格，其中“相邻单元格”是指那些水平相邻或垂直相邻的单元格。
2. 同一个单元格内的字母不允许被重复使用。
3. 假定在数组中最多只存在一个可能的匹配。

## **题目要求**

输入描述:

* 1. 第1行为一个数字（N）指示二维数组在后续输入所占的行数。
  2. 第2行到第N+1行输入为一个二维大写字符数组，每行字符用半角,分割。
  3. 第N+2行为待查找的字符串，由大写字符组成。
  4. 二维数组的大小为N\*N，0<N<=100。
  5. 单词长度K，0<K<1000。

输出描述:

输出一个位置下标字符串，拼接格式为：第1个字符行下标+","+第1个字符列下标+","+第2个字符行下标+","+第2个字符列下标...+","+第N个字符行下标+","+第N个字符列下标

#### **示例1**

输入

4  
A,C,C,F  
C,D,E,D  
B,E,S,S  
F,E,C,A  
ACCESS

输出

0,0,0,1,0,2,1,2,2,2,2,3

说明

ACCESS分别对应二维数组的[0,0] [0,1] [0,2] [1,2] [2,2] [2,3]下标位置

## **JAVA解法**

import java.util.Scanner;

/\*\*

\* @since 2022年4月23日

\*/

public class FindWord {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int N = Integer.parseInt(scanner.nextLine());

char[][] twodim = new char[N][N];

int count = 0;

while (count < N) {

String line = scanner.nextLine();

String replacedLine = line.replace(",", "");

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

twodim[count][i] = replacedLine.charAt(i);

}

count++;

}

String lastLine = scanner.nextLine();

StringBuilder str = new StringBuilder();

boolean result = false;

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

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

if (lastLine.charAt(0) == twodim[i][j]) {

StringBuilder resultStr = new StringBuilder();

if (find(twodim, lastLine, i, j, 0, resultStr)) {

str = resultStr;

}

}

}

}

System.out.println(str.reverse());

}

private static boolean find(char[][] twodim, String lastLine, int i, int j, int indexOfStr, StringBuilder resultStr) {

if (indexOfStr >= lastLine.length()) {

return true;

}

if (i < 0 || i >= twodim.length || j < 0 || j >= twodim.length || twodim[i][j] != lastLine.charAt(indexOfStr)) {

return false;

}

twodim[i][j] += 300; // 防止该位置被重复判断

boolean result =

find(twodim, lastLine, i - 1, j, indexOfStr + 1, resultStr)

|| find(twodim, lastLine, i + 1, j, indexOfStr + 1, resultStr)

|| find(twodim, lastLine, i, j - 1, indexOfStr + 1, resultStr)

|| find(twodim, lastLine, i, j + 1, indexOfStr + 1, resultStr);

twodim[i][j] -= 300;

if (result) {

if (indexOfStr == 0) {

resultStr.append(j).append(",").append(i);

}

if (indexOfStr != 0) {

resultStr.append(j).append(",").append(i).append(",");

}

}

return result;

}

}

# ****【算法题】HJ92 在字符串中找出连续最长的数字串****

## **题目**

****输入一个字符串，返回其最长的数字子串，以及其长度。若有多个最长的数字子串，则将它们全部输出（按原字符串的相对位置）  
本题含有多组样例输入。  
数据范围：字符串长度 1 \le n \le 200 \1≤n≤200 ， 保证每组输入都至少含有一个数字****

## **描述:**

输入：输入一个字符串。1<=len(字符串)<=200  
输出：输出字符串中最长的数字字符串和它的长度，中间用逗号间隔。如果有相同长度的串，则要一块儿输出（中间不要输出空格）。

### **示例1**

输入  
abcd12345ed125ss123058789  
a8a72a6a5yy98y65ee1r2

输出  
123058789,9  
729865,2

说明：  
样例一最长的数字子串为123058789，长度为9  
样例二最长的数字子串有72,98,65，长度都为2

## **JAVA解法**

import java.util.Scanner;

/\*\*

\* @since 2022-04-24

\*/

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

while (scanner.hasNextLine()) {

String line = scanner.nextLine();

StringBuilder digitalStr = new StringBuilder();

StringBuilder digitalStr1 = findDigitalStr(line, digitalStr, 0, 0);

System.out.println(digitalStr1.toString());

}

}

private static StringBuilder findDigitalStr(String line, StringBuilder digitalStr, int index, int count) {

if (index >= line.length()) {

return digitalStr.append(",").append(count);

}

StringBuilder temp = new StringBuilder();

for (int i = index; i < line.length(); i++) {

index++;

char c = line.charAt(i);

if (47 < c && c < 58) {

temp.append(c);

continue;

}

break;

}

if (temp.length() > 0 && temp.length() == count) {

digitalStr.append(temp);

}

if (temp.length() > count) {

count = temp.length();

digitalStr = temp;

}

return findDigitalStr(line, digitalStr, index, count);

}

}

# ****【算法题】最远足迹****

## **题目**

某探险队负责对地下洞穴进行探险。探险队成员在进行探险任务时，随身携带的记录器会不定期地记录自身的坐标，但在记录的间隙中也会记录其他数据。探索工作结束后，探险队需要获取到某成员在探险过程中相对于探险队总部的最远的足迹位置。

1. 仪器记录坐标时，坐标的数据格式为(x,y)，如(1,2)、(100,200)，其中0<x<1000，0<y<1000。同时存在非法坐标，如(01,1)、(1,01)，(0,100)属于非法坐标。
2. 设定探险队总部的坐标为(0,0)，某位置相对总部的距离为：x**x+y**y。
3. 若两个座标的相对总部的距离相同，则第一次到达的坐标为最远的足迹。
4. 若记录仪中的坐标都不合法，输出总部坐标（0,0）。  
   备注：不需要考虑双层括号嵌套的情况，比如sfsdfsd((1,2))。

## **描述:**

输入：字符串，表示记录仪中的数据。  
如：ferga13fdsf3(100,200)f2r3rfasf(300,400)

输出：字符串，表示最远足迹到达的坐标。  
如： (300,400)

#### **示例1**

输入  
ferg(3,10)a13fdsf3(3,4)f2r3rfasf(5,10)  
输出  
(5,10)  
说明  
记录仪中的合法坐标有3个： (3,10)， (3,4)， (5,10)，其中(5,10)是相距总部最远的坐标， 输出(5,10)。

#### **示例2**

输入  
asfefaweawfaw(0,1)fe  
输出  
(0,0)  
说明  
记录仪中的坐标都不合法，输出总部坐标（0,0）

## **JAVA解法**

import java.util.ArrayList;

import java.util.Scanner;

/\*\*

\* 合法坐标

\*

\* @since 2022年4月25日

\*/

public class LegalCoordinates {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

String line = scanner.nextLine();

// 拿到所有坐标

ArrayList<String> coordinates = new ArrayList<>();

StringBuilder builder = new StringBuilder();

boolean left = false;

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

char c = line.charAt(i);

if (c == '(') {

left = true;

continue;

}

if (left) {

if (c == ')') {

left = false;

coordinates.add(builder.toString());

builder.setLength(0);

continue;

}

builder.append(c);

}

}

// 处理坐标

ArrayList<String> legalCoordinates = new ArrayList<>();

String legalStr = "";

int len = 0;

for (String coordinate : coordinates) {

String[] split = coordinate.split(",");

if (split.length != 2) {

continue;

}

if (split[0].length() >= 4

|| split[1].length() >= 4

|| split[0].startsWith("0")

|| split[1].startsWith("0")) {

continue;

}

int result = calculate(split[0], split[1]);

if (result > len) {

len = result;

legalStr = '(' + coordinate + ')';

}

}

System.out.println("".equals(legalStr) ? "(0,0)" : legalStr);

}

private static int calculate(String left, String right) {

int le = Integer.parseInt(left);

int ri = Integer.parseInt(right);

return le \* le + ri \* ri;

}

}

# ****【算法题】LeeCode 回文子字符串的个数****

## **题目**

给定一个字符串 s ，请计算这个字符串中有多少个回文子字符串。  
具有不同开始位置或结束位置的子串，即使是由相同的字符组成，也会被视作不同的子串。

### **提示:**

1 <= s.length <= 1000  
s 由小写英文字母组成

#### **示例1**

输入：s = "abc"  
输出：3  
解释：三个回文子串: "a", "b", "c"

#### **示例2**

输入：s = "aaa"  
输出：6  
解释：6个回文子串: "a", "a", "a", "aa", "aa", "aaa"

## **JAVA解法**

import java.util.Scanner;

/\*\*

\* @since 2022年4月26日

\*/

public class LeeCode\_020 {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

String line = scanner.nextLine();

System.out.println(countSubstrings(line));

}

public static int countSubstrings(String line) {

char[] chars = line.toCharArray();

// StringBuilder builder = new StringBuilder();

int count = 0;

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

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

if (j < i || chars[i] != chars[j]) {

continue;

}

int l = i; // 定义最左边指针

int r = j; // 定义最右边指针

// 思路：两边向中间移动指针，判断回文字符

while (l <= r) {

// 一旦出现不对称就跳出，说明不符合回文规则

if (chars[l] != chars[r]) {

r = -2;

break;

}

l++;

r--;

}

// -2就是个标记,预示本区间字符串不符合回文规则

if (r != -2) {

count++;

// builder.append(line, i, temp + 1).append(", ");

}

}

}

// System.out.println(builder);

return count;

}

}

# ****【算法题】LeeCode 最长回文子串****

## **题目**

给你一个字符串 s，找到 s 中最长的回文子串。

### **提示:**

1 <= s.length <= 1000  
s 由小写英文字母组成  
回文串是指正反两个方向都一样的单词或短语。排列是指字母的重新排列。

#### **示例1**

输入：s = "babad"  
输出："bab"  
解释："aba" 同样是符合题意的答案。

#### **示例2**

输入：s = "cbbd"  
输出："bb"

## **JAVA解法**

import java.util.Scanner;

/\*\*

\* 5. 最长回文子串

\*

\* @since 2022年4月25日

\*/

public class LeeCode\_5 {

public static String longestPalindrome(String line) {

int[] strs = {0, 0};

int longest = 0;

char[] chars = line.toCharArray();

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

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

// 找到右边的相同字符索引

if (j < i || chars[i] != chars[j]) {

continue;

}

int l = i; // 定义最左侧指针

int r = j; // 定义最右侧指针

// 从两侧向中间移动，如果出现不相等，就判定不符合回文规则

while (l <= r) {

if (chars[l] != chars[r]) {

r = -2; // 定义一个标记，表示本次区间不符合回文规则

break;

}

l++;

r--;

}

int tempLong;

if (r == -2) {

continue;

} else {

tempLong = j - i;

}

if (tempLong >= longest) {

longest = tempLong; // 最长的回文字符串长度标记一下

strs[0] = i; // 记录回文串左索引

strs[1] = j + 1; // 记录回文串右侧索引

}

}

}

return line.substring(strs[0], strs[1]);

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

String line = scanner.nextLine();

System.out.println(longestPalindrome(line));

}

}

# ****【算法题】LeeCode 回文排列****

## **题目**

给定一个字符串，编写一个函数判定其是否为某个回文串的排列之一。  
回文串是指正反两个方向都一样的单词或短语。排列是指字母的重新排列。  
回文串不一定是字典当中的单词。

### **提示:**

1 <= s.length <= 1000  
s 由小写英文字母组成

#### **示例1**

输入："tactcoa"  
输出：true（排列有"tacocat"、"atcocta"，等等）

## **JAVA解法**

import java.util.Scanner;

public class LeeCode\_01\_04 {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

String line = scanner.nextLine();

System.out.println(canPermutePalindrome(line));

}

public static boolean canPermutePalindrome(String line) {

char[] chars = line.toCharArray();

int oneSingle = 0; // 奇数变量的个数

int temp = 256; // 定义一个临时变量存储独一个变量

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

// 重复的奇数字符不再计入

if (temp == chars[i]) {

continue;

}

// 计数各个字符的个数

int count = 0;

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

if (chars[i] == chars[j]) {

count++;

}

}

if (count % 2 != 0) {

oneSingle++;

temp = chars[i];

}

}

return oneSingle <= 1;

}

}

# ****【算法题】判断一组不等式是否满足约束并输出最大差****

import java.util.Arrays;  
import java.util.Scanner;  
import java.util.stream.Collectors;  
  
/\*\*  
 \* 标题：判断一组不等式是否满足约束并输出最大差 | 时间限制：1秒 | 内存限制：65536K | 语言限制：不限  
 \* 给定一组不等式，判断是否成立并输出不等式的最大差(输出浮点数的整数部分)，要求：  
 \* 1）不等式系数为double类型，是一个二维数组；  
 \* 2）不等式的变量为int类型，是一维数组；  
 \* 3）不等式的目标值为double类型，是一维数组；  
 \* 4）不等式约束为字符串数组，只能是：">",">=","<","<=","="，例如,不等式组：  
 \* a11\*x1+a12\*x2+a13\*x3+a14\*x4+a15\*x5<=b1;  
 \* a21\*x1+a22\*x2+a23\*x3+a24\*x4+a25\*x5<=b2;  
 \* a31\*x1+a32\*x2+a33\*x3+a34\*x4+a35\*x5<=b3;  
 \* <p>  
 \* 最大差=max{ (a11\*x1+a12\*x2+a13\*x3+a14\*x4+a15\*x5-b1), (a21\*x1+a22\*x2+a23\*x3+a24\*x4+a25\*x5-b2), (a31\*x1+a32\*x2+a33\*x3+a34\*x4+a35\*x5-b3) }，类型为整数(输出浮点数的整数部分)  
 \* <p>  
 \* 输入描述:  
 \* 1）不等式组系数(double类型)：  
 \* a11,a12,a13,a14,a15  
 \* a21,a22,a23,a24,a25  
 \* a31,a32,a33,a34,a35  
 \* 2）不等式变量(int类型)：  
 \* x1,x2,x3,x4,x5  
 \* 3）不等式目标值(double类型)：b1,b2,b3  
 \* 4)不等式约束(字符串类型):<=,<=,<=  
 \* <p>  
 \* 输入：a11,a12,a13,a14,a15;a21,a22,a23,a24,a25;a31,a32,a33,a34,a35;x1,x2,x3,x4,x5;b1,b2,b3;<=,<=,<=  
 \* <p>  
 \* 输出描述:  
 \* true 或者 false, 最大差  
 \* 示例1  
 \* 输入  
 \* 2.3,3,5.6,7,6;11,3,8.6,25,1;0.3,9,5.3,66,7.8;1,3,2,7,5;340,670,80.6;<=,<=,<=  
 \* 输出  
 \* false 458  
 \* 示例2  
 \* 输入  
 \* 2.36,3,6,7.1,6;1,30,8.6,2.5,21;0.3,69,5.3,6.6,7.8;1,13,2,17,5;340,67,300.6;<=,>=,<=  
 \* 输出  
 \* false 758  
 \*  
 \* @since 2022年4月28日  
 \*/  
public class M\_N\_T\_14 {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.in);  
 String line = scanner.nextLine();  
  
 String[] strings = line.split(";");  
  
 // 不等式组系数(double类型)  
 double[][] a = new double[strings[strings.length - 1].split(",").length][strings[0].split(",").length];  
 for (int i = 0; i < a.length; i++) {  
 String[] les = strings[i].split(",");  
 for (int j = 0; j < les.length; j++) {  
 a[i][j] = Double.parseDouble(les[j]);  
 }  
 }  
  
 // 不等式变量(int类型)  
 Integer[] x = Arrays.stream(strings[a.length].split(",")).map(Integer::parseInt).collect(Collectors.toList()).toArray(new Integer[strings[0].split(",").length]);  
  
 // 不等式目标值(double类型)  
 Double[] b = Arrays.stream(strings[strings.length - 2].split(",")).map(Double::parseDouble).collect(Collectors.toList()).toArray(new Double[a.length]);  
  
 // 不等式约束(字符串类型)  
 String[] con = Arrays.stream(strings[strings.length - 1].split(",")).collect(Collectors.toList()).toArray(new String[a.length]);  
  
 // 开始比较  
 boolean bool = true;  
 double max = 0;  
 for (int i = 0; i < con.length; i++) {  
 boolean bo;  
 double temp;  
 switch (con[i]) {  
 case ">":  
 temp = getLeft(a[i], x) - b[i];  
 bo = temp > 0;  
 break;  
 case ">=":  
 temp = getLeft(a[i], x) - b[i];  
 bo = temp >= 0;  
 break;  
 case "<=":  
 temp = getLeft(a[i], x) - b[i];  
 bo = temp <= 0;  
 break;  
 case "<":  
 temp = getLeft(a[i], x) - b[i];  
 bo = temp < 0;  
 break;  
 default:  
 temp = getLeft(a[i], x) - b[i];  
 bo = temp == 0;  
 }  
  
 max = Math.max(temp, max);  
 if (bool) {  
 bool = bo;  
 }  
 }  
  
 System.out.println(bool + " " + (int) max);  
 }  
  
 public static double getLeft(double[] aRight, Integer[] x) {  
 double d = 0;  
 for (int i = 0; i < x.length; i++) {  
 d += aRight[i] \* x[i];  
 }  
 return d;  
 }  
}

# ****【算法题】字符串比较****

import java.util.Scanner;  
  
/\*\*  
 \* 标题：字符串比较 | 时间限制：1秒 | 内存限制：65536K | 语言限制：不限  
 \* 给定字符串A、B和正整数V，A的长度与B的长度相等， 请计算A中满足如下条件的最大连续子串的长度：  
 \* 1、该连续子串在A和B中的位置和长度均相同。  
 \* 2、该连续子串|A[i] – B[i]|之和小于等于V。其中|A[i] – B[i]|表示两个字母ASCII码之差的绝对值。  
 \* <p>  
 \* 第一行为字符串A，仅包含小写字符，1 <= A.length <=1000。  
 \* 第二行为字符串B，仅包含小写字符，1 <= B.length <=1000。  
 \* 第三行为正整数V，0<= V <= 10000。  
 \* 输出描述:  
 \* 字符串最大连续子串的长度，要求该子串|A[i] – B[i]|之和小于等于V。  
 \* <p>  
 \* 示例1  
 \* 输入  
 \* xxcdefg  
 \* cdefghi  
 \* 5  
 \* 输出  
 \* 2  
 \* 说明  
 \* 字符串A为xxcdefg，字符串B为cdefghi，V=5。  
 \* 它的最大连续子串可以是cd->ef,de->fg,ef->gh,fg->hi，所以最大连续子串是2。  
 \*  
 \* @since 2022年4月30日  
 \*/  
public class M\_N\_T\_15 {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.in);  
 String A = scanner.nextLine();  
 String B = scanner.nextLine();  
 String V = scanner.nextLine();  
  
 char[] As = A.toCharArray();  
 char[] Bs = B.toCharArray();  
 int v = Integer.parseInt(V);  
  
 int first = 0;  
 int len = 0;  
 int cha = 0;  
 for (int i = 0; i < As.length - 1; i++) {  
 int tempLast = 0;  
 if (As[i + 1] == As[i] + 1) {  
 tempLast = i + 1;  
 } else {  
 first = i + 1;  
 continue;  
 }  
  
 for (int j = first; j <= tempLast - 1; j++) {  
 if (Bs[j + 1] != Bs[j] + 1) {  
 cha = 0;  
 break;  
 }  
  
 if (j == first) {  
 cha += Math.abs(As[j] - Bs[j]);  
 }  
 cha += Math.abs(As[j + 1] - Bs[j + 1]);  
 }  
  
 int tempLen = tempLast - first + 1;  
 if (cha <= v && tempLen > len) {  
 len = tempLen;  
 cha = 0;  
 }  
 }  
  
 System.out.println(len);  
 }  
}

# ****【算法题】消消乐游戏****

#### **2022年6月9日23点14分 更新**

import java.util.Scanner;  
import java.util.Stack;  
  
*/\*\*  
 \* 标题：消消乐游戏 | 时间限制：1秒 | 内存限制：65536K | 语言限制：不限  
 \* 游戏规则：输入一个只包含英文字母的字符串，字符串中的两个字母如果相邻且相同，就可以消除。  
 \* <p>  
 \* 在字符串上反复执行消除的动作，直到无法继续消除为止，此时游戏结束。  
 \* 输出最终得到的字符串长度。  
 \* <p>  
 \* 大小写敏感， str 长度不超过100。  
 \* <p>  
 \* 输出描述:  
 \* 输出游戏结束后，最终得到的字符串长度  
 \* <p>  
 \* 示例1  
 \* 输入  
 \* gg  
 \* 输出  
 \* 0  
 \* 说明  
 \* gg 可以直接消除，得到空串，长度为0  
 \* 示例2  
 \* 输入  
 \* mMbccbc  
 \* 输出  
 \* 3  
 \* 说明  
 \* 在 mMbccbc 中，可以先消除 cc ；此时字符串变成 mMbbc ，可以再消除 bb ；此时字符串变成 mMc ，此时没有相邻且相同的字符，无法继续消除。最终得到的字符串为 mMc ，长度为3  
 \* 备注:  
 \* 输入中包含 非大小写英文字母 时，均为异常输入，直接返回 0  
 \* <p>  
 \* aaccbbdcskiuhgvsdffdsvcj 10  
 \* scdxcckkjjnhbbvvfgv 9  
 \*  
 \** ***@since*** *2022年6月9日  
 \*/*public class M\_N\_T\_16 {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.*in*);  
 String line = scanner.nextLine();  
  
 char[] chars = line.toCharArray();  
  
 *extracted\_2*(chars);  
  
 }  
  
 private static void extracted\_2(char[] chars) {  
 Stack<Character> stack = new Stack<>(); // 利用栈特性  
 for (char aChar : chars) {  
 if (!Character.*isLetter*(aChar)) { // 遇到非字母字符直接返回0  
 System.*out*.println(0);  
 return;  
 }  
  
 if (stack.size() == 0) { // 注意第一个字符或者栈中已有字符被消除完了情况，要先插入  
 stack.push(aChar);  
 continue;  
 }  
  
 if (aChar == stack.peek()) { // 相同就会弹栈，反之入栈 peek()获取最近一个入栈的字符而先不弹出  
 stack.pop();  
 } else {  
 stack.push(aChar);  
 }  
  
 }  
  
 System.*out*.println(stack.size()); // 判断最后留下的  
 }  
}

原解法

import java.util.Scanner;  
  
/\*\*  
 \* 标题：消消乐游戏 | 时间限制：1秒 | 内存限制：65536K | 语言限制：不限  
 \* 游戏规则：输入一个只包含英文字母的字符串，字符串中的两个字母如果相邻且相同，就可以消除。  
 \* <p>  
 \* 在字符串上反复执行消除的动作，直到无法继续消除为止，此时游戏结束。  
 \* 输出最终得到的字符串长度。  
 \* <p>  
 \* 大小写敏感， str 长度不超过100。  
 \* <p>  
 \* 输出描述:  
 \* 输出游戏结束后，最终得到的字符串长度  
 \* <p>  
 \* 示例1  
 \* 输入  
 \* gg  
 \* 输出  
 \* 0  
 \* 说明  
 \* gg 可以直接消除，得到空串，长度为0  
 \* 示例2  
 \* 输入  
 \* mMbccbc  
 \* 输出  
 \* 3  
 \* 说明  
 \* 在 mMbccbc 中，可以先消除 cc ；此时字符串变成 mMbbc ，可以再消除 bb ；此时字符串变成 mMc ，此时没有相邻且相同的字符，无法继续消除。最终得到的字符串为 mMc ，长度为3  
 \* 备注:  
 \* 输入中包含 非大小写英文字母 时，均为异常输入，直接返回 0  
 \* <p>  
 \* aaccbbdcskiuhgvsdffdsvcj 10  
 \* scdxcckkjjnhbbvvfgv 9  
 \*  
 \* @since 2022年4月30日  
 \*/  
public class M\_N\_T\_16 {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.in);  
 String line = scanner.nextLine();  
  
 char[] chars = line.toCharArray();  
  
 StringBuilder sb = new StringBuilder();  
 merge(chars,0, 0, 0, 1, sb);  
  
 System.out.println(sb.length());  
 }  
  
 private static void merge(char[] chars, int start, int preIndex, int le, int ri, StringBuilder sb) {  
 if (le > chars.length || ri > chars.length) {  
 return;  
 }  
  
 if (ri == chars.length) {  
 sb.append(chars[ri - 1]);  
 return;  
 }  
  
 if (!(((chars[le] >= 65 && chars[le] <= 90) || (chars[le] >= 97 && chars[le] <= 122)) &&  
 ((chars[ri] >= 65 && chars[ri] <= 90) || (chars[ri] >= 97 && chars[ri] <= 122)))) {  
 sb.setLength(0);  
 return;  
 }  
  
 if (chars[le] == chars[ri]) {  
 start = le == start ? ri + 1 : start;  
 le--;  
 if (le < start) {  
 le = ri + 1;  
 ri++;  
 } else {  
 le = Math.min(le, preIndex);  
 sb.deleteCharAt(sb.length() - 1);  
 }  
 } else {  
 sb.append(chars[le]);  
 preIndex = le;  
 le = ri;  
 }  
 ri++;  
  
 merge(chars, start, preIndex, le, ri, sb);  
 }  
}

# ****【算法题】字符串分割****

import java.util.Scanner;  
  
/\*\*  
 \* 标题：字符串分割 | 时间限制：1秒 | 内存限制：262144K | 语言限制：不限  
 \* 给定一个非空字符串S，其被N个‘-’分隔成N+1的子串，给定正整数K，要求除第一个子串外，其余的子串每K个字符组成新的子串，并用‘-’分隔。  
 \* 对于新组成的每一个子串，如果它含有的小写字母比大写字母多，则将这个子串的所有大写字母转换为小写字母；  
 \* 反之，如果它含有的大写字母比小写字母多，则将这个子串的所有小写字母转换为大写字母；大小写字母的数量相等时，不做转换。  
 \* <p>  
 \* 输入为两行，第一行为参数K，第二行为字符串S。  
 \* 输出描述:  
 \* 输出转换后的字符串。  
 \* <p>  
 \* 示例1  
 \* 输入  
 \* 3  
 \* 12abc-abCABc-4aB@  
 \* 输出  
 \* 12abc-abc-ABC-4aB-@  
 \* 说明  
 \* 子串为12abc、abCABc、4aB@，第一个子串保留，后面的子串每3个字符一组为abC、ABc、4aB、@，  
 \* abC中小写字母较多，转换为abc，ABc中大写字母较多，转换为ABC，4aB中大小写字母都为1个，不做转换，@中没有字母，连起来即12abc-abc-ABC-4aB-@  
 \* <p>  
 \* 示例2  
 \* 输入  
 \* 12  
 \* 12abc-abCABc-4aB@  
 \* 输出  
 \* 12abc-abCABc4aB@  
 \* 说明  
 \* 子串为12abc、abCABc、4aB@，第一个子串保留，后面的子串每12个字符一组为abCABc4aB@，这个子串中大小写字母都为4个，不做转换，连起来即12abc-abCABc4aB@  
 \*  
 \* @since 2022年4月30日  
 \*/  
public class M\_N\_T\_17 {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.in);  
 String K = scanner.nextLine();  
 String S = scanner.nextLine();  
  
 String[] strings = S.split("-");  
  
 StringBuilder stringBuilder = new StringBuilder();  
 for (int i = 1; i < strings.length; i++) {  
 stringBuilder.append(strings[i]);  
 }  
  
 char[] chars = stringBuilder.toString().toCharArray();  
  
 int k = Integer.parseInt(K);  
 int num = chars.length / k;  
 int mo = chars.length % k;  
  
 StringBuilder finalStr = new StringBuilder();  
 int len = 0;  
 for (int i = 0; i <= num; i++) {  
 len += (i + 1) \* k <= num \* k ? k : mo;  
 StringBuilder tempStr = new StringBuilder();  
 int countUp = 0;  
 int countDo = 0;  
 for (int j = i \* k; j < len; j++) {  
 if (65 <= chars[j] && chars[j] <= 90) {  
 countUp++;  
 }  
 if (97 <= chars[j] && chars[j] <= 122) {  
 countDo++;  
 }  
 tempStr.append(chars[j]);  
 }  
  
 if (countUp > countDo) {  
 finalStr.append("-").append(tempStr.toString().toUpperCase());  
 } else if (countUp == countDo) {  
 finalStr.append("-").append(tempStr);  
 } else {  
 finalStr.append("-").append(tempStr.toString().toLowerCase());  
 }  
 }  
  
 System.out.println(strings[0] + finalStr);  
 }  
}

# ****【算法题】找单词（二刷）****

import java.util.Scanner;  
  
/\*\*  
 \* 标题：找单词 | 时间限制：1秒 | 内存限制：32768K | 语言限制：不限  
 \* 给一个字符串和一个二维字符数组，如果该字符串存在于该数组中，则按字符串的字符顺序输出字符串每个字符所在单元格的位置下标字符串，如果找不到返回字符串"N"。  
 \* <p>  
 \* 1.需要按照字符串的字符组成顺序搜索，且搜索到的位置必须是相邻单元格，其中“相邻单元格”是指那些水平相邻或垂直相邻的单元格。  
 \* 2.同一个单元格内的字母不允许被重复使用。  
 \* 3.假定在数组中最多只存在一个可能的匹配。  
 \* <p>  
 \* 输入描述:  
 \* 1.第1行为一个数字（N）指示二维数组在后续输入所占的行数。  
 \* <p>  
 \* 2.第2行到第N+1行输入为一个二维大写字符数组，每行字符用半角,分割。  
 \* <p>  
 \* 3.第N+2行为待查找的字符串，由大写字符组成。  
 \* <p>  
 \* 4.二维数组的大小为N\*N，0<N<=100。  
 \* <p>  
 \* 5.单词长度K，0<K<1000。  
 \* 输出描述:  
 \* 输出一个位置下标字符串，拼接格式为：第1个字符行下标+","+第1个字符列下标+","+第2个字符行下标+","+第2个字符列下标...+","+第N个字符行下标+","+第N个字符列下标  
 \* 示例1  
 \* 输入  
 \* 4  
 \* A,C,C,F  
 \* C,D,E,D  
 \* B,E,S,S  
 \* F,E,C,A  
 \* ACCESS  
 \* 输出  
 \* 0,0,0,1,0,2,1,2,2,2,2,3  
 \* 说明  
 \* ACCESS分别对应二维数组的[0,0] [0,1] [0,2] [1,2] [2,2] [2,3]下标位置  
 \*  
 \* @since 2022年5月1日  
 \*/  
public class M\_N\_T\_18 {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.in);  
 int n = Integer.parseInt(scanner.nextLine()); // 二维数组在后续输入所占的行数  
  
 char[][] chars = new char[n][n];  
 for (int i = 0; i < n; i++) {  
 String[] strs = scanner.nextLine().split(",");  
 for (int j = 0; j < n; j++) {  
 chars[i][j] = strs[j].charAt(0);  
 }  
 }  
  
 char[] ks = scanner.nextLine().toCharArray();// 待查找的字符串  
  
// System.out.println(chars);  
  
 StringBuilder sb = new StringBuilder();  
  
 if (findWord(chars, ks, 0, 0, 0, sb)) {  
 System.out.println(sb.reverse().substring(1));  
 } else {  
 System.out.println("N");  
 }  
  
  
 }  
  
 public static boolean findWord(char[][] chars, char[] k, int le, int ri, int index, StringBuilder sb) {  
 if (index >= k.length) return true; // 如果达到给定字符串的长度就结束  
 if (le < 0 || le >= chars.length || ri < 0 || ri >= chars.length || chars[le][ri] != k[index]) {  
 return false;  
 }  
  
 chars[le][ri] += 300;  
 // 上下左右寻找 递归  
 index++;  
 if (findWord(chars, k, le - 1, ri, index, sb) ||  
 findWord(chars, k, le + 1, ri, index, sb) ||  
 findWord(chars, k, le, ri - 1, index, sb) ||  
 findWord(chars, k, le, ri + 1, index, sb)) {  
 sb.append(ri).append(",").append(le).append(",");  
 return true;  
 }  
 chars[le][ri] -= 300;  
  
 return false;  
 }  
}

# ****【算法题】补种未成活胡杨****

import java.util.Scanner;  
  
/\*\*  
 \* 标题：补种未成活胡杨 | 时间限制：1秒 | 内存限制：262144K  
 \* 近些年来，我国防沙治沙取得显著成果。某沙漠新种植N棵胡杨（编号1-N），排成一排。一个月后，有M棵胡杨未能成活。  
 \* 现可补种胡杨K棵，请问如何补种（只能补种，不能新种），可以得到最多的连续胡杨树？  
 \* 输入描述:  
 \* N 总种植数量 1<=N<=100000 M 未成活胡杨数量 1<=M<=N M 个空格分隔的数，按编号从小到大排列 K 最多可以补种的数量 0<=K<=M  
 \* 输出描述:  
 \* 最多的连续胡杨棵树  
 \* 示例1  
 \* 输入  
 \* 5  
 \* 2  
 \* 2 4  
 \* 1  
 \* 输出  
 \* 3  
 \* 说明  
 \* 补种到2或4结果一样，最多的连续胡杨棵树都是3  
 \* 示例2  
 \* 输入  
 \* 10  
 \* 3  
 \* 2 4 7  
 \* 1  
 \* 输出  
 \* 6  
 \* 说明  
 \* 补种第7棵树，最多的连续胡杨棵树为6(5,6,7,8,9,10)  
 \*  
 \* @since 2022年5月1日  
 \*/  
public class M\_N\_T\_19 {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.in);  
 int N = Integer.parseInt(scanner.nextLine());  
 int M = Integer.parseInt(scanner.nextLine());  
  
 String[] Ms = scanner.nextLine().split(" ");  
 int[] ints\_M = new int[M];  
 for (int i = 0; i < Ms.length; i++) {  
 ints\_M[i] = Integer.parseInt(Ms[i]);  
 }  
  
 int K = Integer.parseInt(scanner.nextLine());  
  
 // 开始补种 滑动窗口，首先知道补种也必须是连续的才有可能获得最大连续棵树，然后设定左右指针，保证中间有K棵树补种  
 int max = 0;  
 for (int i = 0; i <= ints\_M.length - K; i++) {  
 int le = 0;  
 int ri = N;  
 if (i > 0) {  
 le = ints\_M[i - 1];  
 }  
  
 if (i + K < ints\_M.length) {  
 ri = ints\_M[i + K] - 1;  
 }  
  
 int temp = ri - le;  
 if (temp > max) {  
 max = temp;  
 }  
 }  
  
 System.out.println(max);  
 }  
}

# ****【算法题】找朋友****

import java.util.Scanner;  
  
/\*\*  
 \* 标题：找朋友 | 时间限制：1秒 | 内存限制：262144K | 语言限制：不限  
 \* 在学校中，N个小朋友站成一队， 第i个小朋友的身高为height[i]，  
 \* 第i个小朋友可以看到的第一个比自己身高更高的小朋友j，那么j是i的好朋友(要求j > i)。  
 \* 请重新生成一个列表，对应位置的输出是每个小朋友的好朋友位置，如果没有看到好朋友，请在该位置用0代替。  
 \* 小朋友人数范围是 [0, 40000]。  
 \* 输入描述:  
 \* 第一行输入N，N表示有N个小朋友  
 \* 第二行输入N个小朋友的身高height[i]，都是整数  
 \* 输出描述:  
 \* 输出N个小朋友的好朋友的位置  
 \* 示例1  
 \* 输入  
 \* 2  
 \* 100 95  
 \* 输出  
 \* 0 0  
 \* 说明  
 \* 第一个小朋友身高100，站在队尾位置，向队首看，没有比他身高高的小朋友，所以输出第一个值为0。  
 \* 第二个小朋友站在队首，前面也没有比他身高高的小朋友，所以输出第二个值为0。  
 \* 示例2  
 \* 输入  
 \* 8  
 \* 123 124 125 121 119 122 126 123  
 \* 输出  
 \* 1 2 6 5 5 6 0 0  
 \* 说明  
 \* 123的好朋友是1位置上的124  
 \* 124的好朋友是2位置上的125  
 \* 125的好朋友是6位置上的126  
 \* 以此类推  
 \*  
 \* @since 2022年5月1日  
 \*/  
public class M\_N\_T\_20 {  
 public static void main(String[] args) {  
  
 Scanner scanner = new Scanner(System.in);  
 int num = Integer.parseInt(scanner.nextLine());  
 String[] heights = scanner.nextLine().split(" ");  
 int[] friends = new int[num];  
 StringBuilder stringBuilder = new StringBuilder();  
 for (int i = 0; i < heights.length; i++) {  
 int init = Integer.parseInt(heights[i]);  
 friends[i] = 0;  
 for (int j = i + 1; j < heights.length; j++) {  
 int next = Integer.parseInt(heights[j]);  
 if (next > init) {  
 friends[i] = j;  
 break;  
 }  
 }  
 if (i == num - 1) {  
 stringBuilder.append(friends[i]);  
 break;  
 }  
 stringBuilder.append(friends[i]).append(" ");  
 }  
  
 System.out.println(stringBuilder);  
 }  
}

# ****【算法题】求满足条件的最长子串的长度****

import java.util.ArrayList;  
import java.util.Scanner;  
  
/\*\*  
 \* 标题：求满足条件的最长子串的长度 | 时间限制：1秒 | 内存限制：262144K | 语言限制：不限  
 \* 给定一个字符串，只包含字母和数字，按要求找出字符串中的最长（连续）子串的长度，字符串本身是其最长的子串，子串要求：  
 \* 1、 只包含1个字母(a~z, A~Z)，其余必须是数字；  
 \* 2、 字母可以在子串中的任意位置；  
 \* 如果找不到满足要求的子串，如全是字母或全是数字，则返回-1。  
 \* 输入描述:  
 \* 字符串(只包含字母和数字)  
 \* 输出描述:  
 \* 子串的长度  
 \* 示例1  
 \* 输入  
 \* abC124ACb  
 \* 输出  
 \* 4  
 \* 说明 满足条件的最长子串是C124或者124A，长度都是4  
 \* 示例2  
 \* 输入  
 \* a5  
 \* 输出  
 \* 2  
 \* 说明 字符串自身就是满足条件的子串，长度为2  
 \* 示例3  
 \* 输入  
 \* aBB9  
 \* 输出  
 \* 2  
 \* 说明 满足条件的子串为B9，长度为2  
 \* 示例4  
 \* 输入  
 \* abcdef  
 \* 输出  
 \* -1  
 \* 说明 没有满足要求的子串，返回-1  
 \*/  
public class M\_N\_T\_21 {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.in);  
 String line = scanner.nextLine();  
  
 char[] chars = line.toCharArray();  
  
 ArrayList<Integer> integers = new ArrayList<>();  
 integers.add(-1); // 关键点：第一个初始值，防止出现只有一个字母或者没有字母，无法进入循环的情况  
 for (int i = 0; i < chars.length; i++) {  
 char aChar = chars[i];  
 if ((65 <= aChar && aChar <= 90) || (97 <= aChar && aChar <= 122)) {  
 integers.add(i); // 存放所有字母出现的索引值  
 }  
 }  
 integers.add(chars.length); // 关键点：第一个初始值，防止出现只有一个字母或者没有字母，无法进入循环的情况  
  
 int len = -1;  
 for (int i = 0; i < integers.size() - 2; i++) {  
 int temp = integers.get(i + 2) - integers.get(i); // 每隔一个字母差值，即可判断可以放多少个数字  
 if (temp > 2) {  
 if (temp - 1 > len) {  
 len = temp - 1; // 出现更长的长度时才替换  
 }  
 }  
 }  
  
 System.out.println(len);  
 }  
}

# ****【算法题】构成的正方形数量****

import java.util.Scanner;  
  
/\*\*  
 \* 标题：构成的正方形数量 | 时间限制：1秒 | 内存限制：262144K | 语言限制：不限  
 \* 输入N个互不相同的二维整数坐标, 求这N个坐标可以构成的正方形数量。(内积为零的两个向量垂直)  
 \* 输入描述:  
 \* 第一行输入为 N，N 代表坐标数量，N为正整数。N <= 100  
 \* 之后的 K 行输入为坐标 x y以空格分隔，x, y 为整数, -10<=x, y <= 10  
 \* 输出描述:  
 \* 输出可以构成的正方形数量  
 \* 示例1  
 \* 输入  
 \* 3  
 \* 1 3  
 \* 2 4  
 \* 3 1  
 \* 输出  
 \* 0  
 \* 说明 3个点不足以构成正方形  
 \* 示例2  
 \* 输入  
 \* 4  
 \* 0 0  
 \* 1 2  
 \* 3 1  
 \* 2 -1  
 \* 输出  
 \* 1  
 \* 说明 此4点可构成正方形  
 \*/  
public class M\_N\_T\_22 {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.in);  
 int N = Integer.parseInt(scanner.nextLine());  
  
 if (N < 4) {  
 System.out.println(0);  
 return;  
 }  
  
 int[][] zuobiaos = new int[N][2];  
 for (int i = 0; i < zuobiaos.length; i++) {  
 String[] strings = scanner.nextLine().split(" ");  
 zuobiaos[i][0] = Integer.parseInt(strings[0]);  
 zuobiaos[i][1] = Integer.parseInt(strings[1]);  
 }  
  
 int count = 0;  
 for (int i = 0; i < zuobiaos.length - 3; i++) {  
 for (int j = i + 1; j < zuobiaos.length - 2; j++) {  
 for (int k = j + 1; k < zuobiaos.length - 1; k++) {  
 for (int l = k + 1; l < zuobiaos.length; l++) {  
 if (canBeSquare(zuobiaos[i], zuobiaos[j], zuobiaos[k], zuobiaos[l])) {  
 count++;  
 }  
 }  
 }  
 }  
 }  
  
 System.out.println(count);  
 }  
  
 public static boolean canBeSquare(int[] i, int[] j, int[] k, int[] l) {  
 return calculate(new int[]{i[0] - j[0], i[1] - j[1]}, new int[]{k[0] - l[0], k[1] - l[1]}) ||  
 calculate(new int[]{i[0] - k[0], i[1] - k[1]}, new int[]{j[0] - l[0], j[1] - l[1]}) ||  
 calculate(new int[]{i[0] - l[0], i[1] - l[1]}, new int[]{j[0] - k[0], j[1] - k[1]});  
 }  
  
 // 关键点： 判断正方形,需要其对角线垂直且长度相等  
 public static boolean calculate(int[] vec1, int[] vec2) {  
 return vec1[0] \* vec2[0] + vec1[1] \* vec2[1] == 0 && (vec1[0] \* vec1[0] + vec1[1] \* vec1[1]) == (vec2[0] \* vec2[0] + vec2[1] \* vec2[1]);  
 }  
}

# ****【算法题】字符串变换最小字符串****

import java.util.Scanner;  
  
/\*\*  
 \* 标题：字符串变换最小字符串 | 时间限制：1秒 | 内存限制：262144K | 语言限制：不限  
 \* 给定一个字符串s，最多只能进行一次变换，返回变换后能得到的最小字符串（按照字典序进行比较）。  
 \* 变换规则：交换字符串中任意两个不同位置的字符。  
 \* 输入描述:  
 \* 一串小写字母组成的字符串s  
 \* 输出描述:  
 \* 按照要求进行变换得到的最小字符串  
 \* 示例1  
 \* 输入  
 \* abcdef  
 \* 输出  
 \* abcdef  
 \* 说明  
 \* abcdef已经是最小字符串，不需要交换  
 \* 示例2  
 \* 输入  
 \* bcdefa  
 \* 输出  
 \* acdefb  
 \* 说明  
 \* a和b进行位置交换，可以等到最小字符串  
 \* 备注:  
 \* s是都是小写字符组成  
 \* 1<=s.length<=1000  
 \*/  
public class M\_N\_T\_26 {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.in);  
 String line = scanner.nextLine();  
  
 char[] chars = line.toCharArray();  
  
 char ch = chars[chars.length - 1];  
 int index = chars.length - 1;  
  
 for (int i = chars.length - 1; i >= 0; i--) {  
 if (chars[i] < ch) {  
 ch = chars[i];  
 index = i;  
 }  
 }  
  
 for (int i = 0; i < chars.length; i++) {  
 if (ch < chars[i] && index > i) {  
 chars[index] = chars[i];  
 chars[i] = ch;  
 break;  
 }  
 }  
  
 for (int i = 0; i < chars.length; i++) {  
 if (i == chars.length - 1) {  
 System.out.println(chars[i]);  
 break;  
 }  
 System.out.print(chars[i]);  
 }  
  
 }  
}

#### **OD考试遇到，挂了，死因找到了**

import java.util.Scanner;  
  
public class HW\_OD\_02 {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.*in*);  
  
 String line = scanner.nextLine();  
 char[] chars = line.toCharArray();  
  
 for (int i = 0; i < chars.length; i++) {  
  
 int index = -1;  
 char temp\_char = chars[i];  
 // 应该从后面向前面查找，把倒数第一个最小的与前面替换。才能保证最小（死因找到了！！！！）  
 for (int j = chars.length - 1; j > i; j--) {  
  
 if (chars[j] < temp\_char) {  
 index = j;  
 temp\_char = chars[j];  
 }  
  
 }  
  
 if (index != -1) {  
 chars[index] = chars[i];  
 chars[i] = temp\_char;  
 break;  
 }  
 }  
  
 for (char aChar : chars) {  
 System.*out*.print(aChar);  
 }  
  
 }  
}

# ****【算法题】停车场车辆统计****

import java.util.ArrayList;  
import java.util.Scanner;  
  
/\*\*  
 \* 标题：停车场车辆统计 | 时间限制：1秒 | 内存限制：262144K | 语言限制：不限  
 \* 特定大小的停车场，数组cars[]表示，其中1表示有车，0表示没车。车辆大小不一，小车占一个车位（长度1），货车占两个车位（长度2），卡车占三个车位（长度3），统计停车场最少可以停多少辆车，返回具体的数目。  
 \* 输入描述:  
 \* 整型字符串数组cars[]，其中1表示有车，0表示没车，数组长度小于1000。  
 \* 输出描述:  
 \* 整型数字字符串，表示最少停车数目。  
 \* 示例1  
 \* 输入  
 \* 1,0,1  
 \* 输出  
 \* 2  
 \* 说明 1个小车占第1个车位 第二个车位空 1个小车占第3个车位 最少有2辆车  
 \* 示例2  
 \* 输入  
 \* 1,1,0,0,1,1,1,0,1  
 \* 输出  
 \* 3  
 \* 说明 1个货车占第1、2个车位 第3、4个车位空 1个卡车占第5、6、7个车位 第8个车位空 1个小车占第9个车位 最少3辆车  
 \*/  
public class M\_N\_T\_23 {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.in);  
 String line = scanner.nextLine();  
  
 String[] strings = line.split(",");  
  
 ArrayList<Integer> offCars = new ArrayList<>();  
 offCars.add(-1);  
 for (int i = 0; i < strings.length; i++) {  
 if ("0".equals(strings[i])) {  
 offCars.add(i);  
 }  
 }  
 offCars.add(strings.length);  
  
 int count = 0;  
 for (int i = 0; i < offCars.size() - 1; i++) {  
 int offset = offCars.get(i + 1) - offCars.get(i) - 1;  
 int kache = offset / 3;  
 int huoche = (offset - 3 \* kache) / 2;  
 int qiche = offset - 3 \* kache - 2 \* huoche;  
 count += kache;  
 count += huoche;  
 count += qiche;  
 }  
  
 System.out.println(count);  
  
 }  
}

# ****【算法题】最长方连续方波信号****

#### **2022年6月9日23点39分 更新**

import java.util.Scanner;  
import java.util.regex.Matcher;  
import java.util.regex.Pattern;  
  
*/\*\*  
 \* 标题：最长方连续方波信号 | 时间限制：1秒 | 内存限制：262144K | 语言限制：不限  
 \* 输入一串方波信号，求取最长的完全连续交替方波信号，并将其输出，如果有相同长度的交替方波信号，输出任一即可，方波信号高位用1标识，低位用0标识，如图：  
 \* 说明：  
 \* 1） 一个完整的信号一定以0开始然后以0结尾，即010是一个完整信号，但101，1010，0101不是  
 \* 2）输入的一串方波信号是由一个或多个完整信号组成  
 \* 3） 两个相邻信号之间可能有0个或多个低位，如0110010，011000010  
 \* 4） 同一个信号中可以有连续的高位，如01110101011110001010，前14位是一个具有连续高位的信号  
 \* 5） 完全连续交替方波是指10交替，如01010是完全连续交替方波，0110不是  
 \* 输入描述: 输入信号字符串（长度>=3且<=1024）： 0010101010110000101000010 注：输入总是合法的，不用考虑异常情况  
 \* 输出描述: 输出最长的完全连续交替方波信号串： 01010 若不存在完全连续交替方波信号串，输出 -1  
 \* 示例1  
 \* 输入  
 \* 00101010101100001010010  
 \* 输出  
 \* 01010  
 \* 备注:  
 \* 输入信号串中有三个信号：0 010101010110(第一个信号段) 00 01010(第二个信号段) 010(第三个信号段)  
 \* <p>  
 \* 第一个信号虽然有交替的方波信号段，但出现了11部分的连续高位，不算完全连续交替方波，在剩下的连续方波信号串中01010最长  
 \*/*public class M\_N\_T\_24 {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.*in*);  
 String line = scanner.nextLine();  
  
 *extracted\_2*(line);  
 }  
  
 private static void extracted\_2(String line) {  
 Pattern pattern = Pattern.*compile*("00\*0"); // 匹配出方波信号之间的间隔点  
 Matcher matcher = pattern.matcher(line);  
 String replace\_line = matcher.replaceAll("0 0"); // 替换为空格隔开  
  
 String[] strings = replace\_line.split(" "); // 切割  
  
 int len = 0;  
 String lenStr = "";  
 for (String string : strings) {  
 boolean b = string.matches("0(10)+"); // 判断得到的方波信号是否为完全连续交替方波信号  
  
 if (b && string.length() > len) {  
 len = strings.length;  
 lenStr = string;  
 }  
 }  
  
 System.*out*.println(lenStr.equals("") ? -1 : str);  
 }  
}

原解法：

import java.util.ArrayList;  
import java.util.Scanner;  
  
/\*\*  
 \* 标题：最长方连续方波信号 | 时间限制：1秒 | 内存限制：262144K | 语言限制：不限  
 \* 输入一串方波信号，求取最长的完全连续交替方波信号，并将其输出，如果有相同长度的交替方波信号，输出任一即可，方波信号高位用1标识，低位用0标识，如图：  
 \* 说明：  
 \* 1） 一个完整的信号一定以0开始然后以0结尾，即010是一个完整信号，但101，1010，0101不是  
 \* 2）输入的一串方波信号是由一个或多个完整信号组成  
 \* 3） 两个相邻信号之间可能有0个或多个低位，如0110010，011000010  
 \* 4） 同一个信号中可以有连续的高位，如01110101011110001010，前14位是一个具有连续高位的信号  
 \* 5） 完全连续交替方波是指10交替，如01010是完全连续交替方波，0110不是  
 \* 输入描述: 输入信号字符串（长度>=3且<=1024）： 0010101010110000101000010 注：输入总是合法的，不用考虑异常情况  
 \* 输出描述: 输出最长的完全连续交替方波信号串： 01010 若不存在完全连续交替方波信号串，输出 -1  
 \* 示例1  
 \* 输入  
 \* 00101010101100001010010  
 \* 输出  
 \* 01010  
 \* 备注:  
 \* 输入信号串中有三个信号：0 010101010110(第一个信号段) 00 01010(第二个信号段) 010(第三个信号段)  
 \* <p>  
 \* 第一个信号虽然有交替的方波信号段，但出现了11部分的连续高位，不算完全连续交替方波，在剩下的连续方波信号串中01010最长  
 \*/  
public class M\_N\_T\_24 {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.in);  
 String line = scanner.nextLine();  
  
 ArrayList<Integer> integers = new ArrayList<>();  
 integers.add(-1);  
 for (int i = 0; i < line.length(); i++) {  
 if (line.charAt(i) == 49) {  
 integers.add(i);  
 }  
 }  
 integers.add(line.length());  
  
 int le = -1;  
 int ri = -1;  
 int len = 0;  
 boolean isPer = true;  
 String str = "";  
 for (int i = 0; i < integers.size() - 1; i++) {  
 int sumOf\_0 = integers.get(i + 1) - integers.get(i) - 1;  
  
 if (i == 0) {  
 if (sumOf\_0 >= 1) {  
 le = integers.get(i + 1) - 1;  
 }  
 }  
  
 if (i == integers.size() - 2) {  
 if (sumOf\_0 >= 1) {  
 ri = integers.get(i) + 1;  
 if (isPer && ri - le + 1 > len) {  
 len = ri - le + 1;  
 str = line.substring(le, ri + 1);  
 }  
 isPer = true;  
 }  
 }  
  
 if (i > 0 && i < integers.size() - 2) {  
 if (sumOf\_0 == 0) {  
 isPer = false;  
 }  
 if (sumOf\_0 >= 2) {  
 ri = integers.get(i) + 1;  
 if (isPer && ri - le + 1 > len) {  
 len = ri - le + 1;  
 str = line.substring(le, ri + 1);  
 }  
 le = integers.get(i + 1) - 1;  
 isPer = true;  
 }  
 }  
 }  
  
 System.out.println(str.equals("") ? -1 : str);  
 }  
}

# ****【算法题】英文输入法****

import java.util.ArrayList;  
import java.util.List;  
import java.util.Scanner;  
import java.util.stream.Collectors;  
  
/\*\*  
 \* 标题：英文输入法 | 时间限制：1秒 | 内存限制：262144K | 语言限制：不限  
 \* 主管期望你来实现英文输入法单词联想功能。需求如下：  
 \* 依据用户输入的单词前缀，从已输入的英文语句中联想出用户想输入的单词，按字典序输出联想到的单词序列，如果联想不到，请输出用户输入的单词前缀。  
 \* 注意：  
 \* 1. 英文单词联想时，区分大小写  
 \* 2. 缩略形式如”don't”，判定为两个单词，”don”和”t”  
 \* 3. 输出的单词序列，不能有重复单词，且只能是英文单词，不能有标点符号  
 \* 输入描述: 输入为两行。首行输入一段由英文单词word和标点符号组成的语句str； 接下来一行为一个英文单词前缀pre。 0 < word.length() <= 20 0 < str.length <= 10000 0 < pre <= 20  
 \* 输出描述: 输出符合要求的单词序列或单词前缀，存在多个时，单词之间以单个空格分割  
 \* 示例1  
 \* 输入  
 \* I love you  
 \* He  
 \* 输出  
 \* He  
 \* 说明  
 \* 从用户已输入英文语句”I love you”中提炼出“I”、“love”、“you”三个单词，接下来用户输入“He”，从已输入信息中无法联想到任何符合要求的单词，因此输出用户输入的单词前缀。  
 \* 示例2  
 \* 输入  
 \* The furthest distance in the world, Is not between life and death, But when I stand in front of you, Yet you don't know that I love you.  
 \* f  
 \* 输出  
 \* front furthest  
 \* 说明  
 \* 从用户已输入英文语句”The furthestdistance in the world, Is not between life and death, But when I stand in frontof you, Yet you dont know that I love you.”中提炼出的单词，  
 \* 符合“f”作为前缀的，有“furthest”和“front”，按字典序排序并在单词间添加空格后输出，结果为“frontfurthest”。  
 \*/  
public class M\_N\_T\_25 {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.in);  
 String line = scanner.nextLine();  
 char[] chars = line.toCharArray();  
 ArrayList<String> strings = new ArrayList<>();  
 StringBuilder sb = new StringBuilder();  
 for (int i = 0; i < chars.length; i++) {  
 char aChar = chars[i];  
 if ((65 <= aChar && aChar <= 90) || (97 <= aChar && aChar <= 122)) {  
 sb.append(aChar);  
 if (i == chars.length - 1) {  
 strings.add(sb.toString());  
 }  
 } else {  
 if (i != 0 && (65 <= chars[i - 1] && chars[i - 1] <= 90) || (97 <= chars[i - 1] && chars[i - 1] <= 122)) {  
 strings.add(sb.toString());  
 sb.setLength(0);  
 }  
 }  
 }  
  
// System.out.println(strings);  
  
 String inputStr = scanner.nextLine();  
  
 ArrayList<String> lianxiangs = new ArrayList<>();  
 for (int i = 0; i < strings.size(); i++) {  
 String str = strings.get(i);  
 if (str.length() < inputStr.length()) {  
 continue;  
 }  
 boolean isMatch = true;  
 for (int j = 0; j < inputStr.length(); j++) {  
 if (inputStr.charAt(j) != str.charAt(j)) {  
 isMatch = false;  
 break;  
 }  
 }  
  
 if (isMatch) {  
 lianxiangs.add(str);  
 }  
 }  
  
 if (lianxiangs.size() == 0) {  
 System.out.println(inputStr);  
 return;  
 }  
  
 // 排序  
 List<String> stringList = lianxiangs.stream().distinct().sorted((o1, o2) -> {  
 for (int i = inputStr.length(); i < Math.min(o1.length(), o2.length()); i++) {  
 if (o1.charAt(i) != o2.charAt(i)) {  
 return o1.charAt(i) - o2.charAt(i);  
 }  
 }  
 return o1.length() < o2.length() ? -1 : 1;  
 }).collect(Collectors.toList());  
 for (int i = 0; i < stringList.size(); i++) {  
 if (i == stringList.size() - 1) {  
 System.out.println(stringList.get(i));  
 break;  
 }  
 System.out.print(stringList.get(i) + " ");  
 }  
 }  
}

# ****【算法题】信道分配****

import java.util.HashMap;  
import java.util.Map;  
import java.util.Scanner;  
  
/\*\*  
 \* 标题：信道分配 | 时间限制：1秒 | 内存限制：262144K | 语言限制：不限  
 \* 算法工程师小明面对着这样一个问题，需要将通信用的信道分配给尽量多的用户：  
 \* 信道的条件及分配规则如下：  
 \* 1) 所有信道都有属性：”阶”。阶为r的信道的容量为2^r比特；  
 \* 2) 所有用户需要传输的数据量都一样：D比特；  
 \* 3) 一个用户可以分配多个信道，但每个信道只能分配给一个用户；  
 \* 4) 只有当分配给一个用户的所有信道的容量和>=D，用户才能传输数据；  
 \* 给出一组信道资源，最多可以为多少用户传输数据？  
 \* 输入描述:  
 \* 第一行，一个数字R。R为最大阶数。  
 \* 0<=R<20  
 \* 第二行，R+1个数字，用空格隔开。 代表每种信道的数量Ni。按照阶的值从小到大排列。  
 \* 0<=i<=R, 0<=Ni<1000.  
 \* 第三行，一个数字D。 D为单个用户需要传输的数据量。  
 \* 0<D<1000000  
 \* 输出描述:  
 \* 一个数字，代表最多可以供多少用户传输数据。  
 \* 示例1  
 \* 输入  
 \* 5  
 \* 10 5 0 1 3 2  
 \* 30  
 \* 输出  
 \* 4  
 \* 说明  
 \* 最大阶数为5.  
 \* 信道阶数： 0 1 2 3 4 5  
 \* 信道容量： 1 2 4 8 16 32  
 \* 信道个数：10 5 0 1 3 2  
 \* 单个用户需要传输的数据量为30  
 \* 可能存在很多分配方式，举例说明：  
 \* 分配方式1：  
 \* 1) 32\*1 = 32  
 \* 2) 32\*1 = 32  
 \* 3) 16\*2 = 32  
 \* 4) 16\*1 + 8\*1 + 2\*3 = 30  
 \* 剩下2\*2 + 1\*10=14不足以再分一个用户了。  
 \* 分配方式2：  
 \* 1) 16\*1 + 8\*1 + 2\*3 = 30  
 \* 2) 16\*1 + 2\*2 + 1\*10 = 30  
 \* 3) 32\*1 = 32  
 \* 4) 32\*1 = 32  
 \* 剩下16\*1=16不足以再分一个用户了。  
 \* 分配方式3：  
 \* 1) 16\*1 + 8\*1 + 2\*3 = 30  
 \* 2) 16\*1 + 2\*2 + 1\*10 = 30  
 \* 3) 16\*1 + 32\*1 = 48  
 \* 4) 32\*1 = 32  
 \* 恰好用完。  
 \* 虽然每种分配方式剩下的容量不同，但服务的用户数量是一致的。因为这个问题中我们只关心服务的用户数，所以我们认为这些分配方式等效。  
 \*/  
public class M\_N\_T\_27 {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.in);  
 int R = Integer.parseInt(scanner.nextLine()); // 最大阶数  
  
 String[] NiStrs = scanner.nextLine().split(" ");  
 int[][] Nis = new int[R + 1][2]; // 每种信道的数量Ni。按照阶的值从小到大排列  
 for (int i = 0; i < Nis.length; i++) {  
 Nis[i][0] = (int) Math.pow(2, i);  
 Nis[i][1] = Integer.parseInt(NiStrs[i]);  
 }  
  
 int D = Integer.parseInt(scanner.nextLine()); // 单个用户需要传输的数据量  
  
 int count = 0;  
 Map<Integer, Integer> shizhi = new HashMap<>();  
 for (int[] ni : Nis) {  
 int Ni = ni[0];  
 int Ni\_count = ni[1];  
  
 if (Ni\_count == 0) {  
 continue;  
 }  
  
 if (Ni >= D) {  
 count += Ni\_count;  
 continue;  
 }  
  
 int mo = D % Ni;  
 int shang = D / Ni;  
 int need\_count = mo == 0 ? shang : shang + 1;  
  
 count += Ni\_count / need\_count;  
  
 // 剩余个数  
 int Ni\_count\_left = Ni\_count - (Ni\_count / need\_count) \* need\_count;  
  
 if (shizhi.size() != 0) {  
 int temp = 0;  
 for (Map.Entry<Integer, Integer> en : shizhi.entrySet()) {  
 temp += en.getKey() \* en.getValue();  
 }  
  
 // 单个用户还剩多少数据量  
 int D\_left = D - temp;  
 int mo\_left = D\_left % Ni;  
 int shang\_left = D\_left / Ni;  
 int need\_count\_left = mo\_left == 0 ? shang\_left : shang\_left + 1;  
  
 if (Ni\_count\_left >= need\_count\_left) {  
 shizhi.clear();  
 count++;  
 Ni\_count\_left -= need\_count\_left;  
 }  
  
 }  
  
 if (Ni\_count\_left > 0) {  
 shizhi.put(Ni, Ni\_count\_left);  
 }  
 }  
  
 System.out.println(count);  
 }  
}

# ****【算法题】矩阵最大值****

import java.util.ArrayList;  
import java.util.Scanner;  
  
/\*\*  
 \* 标题：矩阵最大值 | 时间限制：1秒 | 内存限制：32768K | 语言限制：不限  
 \* 给定一个仅包含0和1的N\*N二维矩阵，请计算二维矩阵的最大值，计算规则如下：  
 \* 1、 每行元素按下标顺序组成一个二进制数（下标越大越排在低位），二进制数的值就是该行的值。矩阵各行值之和为矩阵的值。  
 \* 2、允许通过向左或向右整体循环移动每行元素来改变各元素在行中的位置。  
 \* 比如： [1,0,1,1,1]向右整体循环移动2位变为[1,1,1,0,1]，二进制数为11101，值为29。  
 \* [1,0,1,1,1]向左整体循环移动2位变为[1,1,1,1,0]，二进制数为11110，值为30。  
 \* 输入描述:  
 \* 1、输入的第一行为正整数，记录了N的大小，0 < N <= 20。  
 \* 2、输入的第2到N+1行为二维矩阵信息，行内元素边角逗号分隔。  
 \* 输出描述:  
 \* 矩阵的最大值。  
 \* 示例1  
 \* 输入  
 \* 5  
 \* 1,0,0,0,1  
 \* 0,0,0,1,1  
 \* 0,1,0,1,0  
 \* 1,0,0,1,1  
 \* 1,0,1,0,1  
 \* 输出  
 \* 122  
 \* 说明  
 \* 第一行向右整体循环移动1位，得到本行的最大值[1,1,0,0,0]，二进制值为11000，十进制值为24。  
 \* 第二行向右整体循环移动2位，得到本行的最大值[1,1,0,0,0]，二进制值为11000，十进制值为24。  
 \* 第三行向左整体循环移动1位，得到本行的最大值[1,0,1,0,0]，二进制值为10100，十进制值为20。  
 \* 第四行向右整体循环移动2位，得到本行的最大值[1,1,1,0,0]，二进制值为11100，十进制值为28。  
 \* 第五行向右整体循环移动1位，得到本行的最大值[1,1,0,1,0]，二进制值为11010，十进制值为26。  
 \* 因此，矩阵的最大值为122。  
 \*/  
public class M\_N\_T\_28 {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.in);  
 int N = Integer.parseInt(scanner.nextLine());  
  
 int[][] ints = new int[N][N];  
 for (int i = 0; i < ints.length; i++) {  
 String[] strings = scanner.nextLine().split(",");  
 for (int j = 0; j < strings.length; j++) {  
 ints[i][j] = Integer.parseInt(strings[j]);  
 }  
 }  
  
 int sum\_max = 0;  
 ArrayList<Integer> integers = new ArrayList<>();  
 for (int i = 0; i < ints.length; i++) {  
 int[] anInt = ints[i];  
 int temp\_sum = transfer\_10(anInt);  
 for (int j = 0; j < (N - 1); j++) {  
 int temp = anInt[0];  
 for (int k = 0; k < N - 1; k++) {  
 anInt[k] = anInt[k + 1];  
 }  
 anInt[N - 1] = temp;  
  
 // 计算  
 int temp\_sum\_new = transfer\_10(anInt);  
 if (temp\_sum < temp\_sum\_new) {  
 temp\_sum = temp\_sum\_new;  
 }  
 }  
  
 sum\_max += temp\_sum;  
 }  
  
 System.out.println(sum\_max);  
 }  
  
 public static int transfer\_10(int[] anIntNew) {  
 int temp = 0;  
 int index = 0;  
 for (int i = anIntNew.length - 1; i > -1; i--) {  
 temp += Math.pow(2, index++) \* anIntNew[i];  
 }  
 return temp;  
 }  
}

# ****【算法题】敏感字段加密****

import java.util.ArrayList;  
import java.util.Scanner;  
  
*/\*\*  
 \* 标题：敏感字段加密 | 时间限制：1秒 | 内存限制：262144K | 语言限制：不限  
 \* 【敏感字段加密】给定一个由多个命令字组成的命令字符串：  
 \* 1、字符串长度小于等于127字节，只包含大小写字母，数字，下划线和偶数个双引号；  
 \* 2、命令字之间以一个或多个下划线\_进行分割；  
 \* 3、可以通过两个双引号""来标识包含下划线\_的命令字或空命令字（仅包含两个双引号的命令字），双引号不会在命令字内部出现；  
 \* 请对指定索引的敏感字段进行加密，替换为\*\*\*\*\*\*（6个\*），并删除命令字前后多余的下划线\_。如果无法找到指定索引的命令字，输出字符串ERROR。  
 \* <p>  
 \* 输入描述:  
 \* 输入为两行，第一行为命令字索引K（从0开始），第二行为命令字符串S。  
 \* 输出描述:  
 \* 输出处理后的命令字符串，如果无法找到指定索引的命令字，输出字符串ERROR  
 \* 示例1  
 \* 输入  
 \* 1  
 \* password\_\_a12345678\_timeout\_100  
 \* 输出  
 \* password\_\*\*\*\*\*\*\_timeout\_100  
 \* 示例2  
 \* 输入  
 \* 2  
 \* aaa\_password\_"a12\_45678"\_timeout\_\_100\_""\_  
 \* 输出  
 \* aaa\_password\_\*\*\*\*\*\*\_timeout\_100\_""  
 \*/*public class M\_N\_T\_29 {  
 public static void main(String[] args) {  
  
 Scanner scanner = new Scanner(System.*in*);  
 int K = Integer.*parseInt*(scanner.nextLine());  
  
 String line = scanner.nextLine();  
  
 char[] chars = line.toCharArray();  
 ArrayList<Integer> int\_shuang\_ying\_hao = new ArrayList<>();  
 for (int i = 0; i < chars.length; i++) {  
 if (chars[i] == '"') {  
 int\_shuang\_ying\_hao.add(i);  
 }  
 }  
  
 if (int\_shuang\_ying\_hao.size() == 0) {  
 int\_shuang\_ying\_hao.add(chars.length);  
 int\_shuang\_ying\_hao.add(chars.length + 1);  
 }  
  
 ArrayList<String> strings = new ArrayList<>();  
 int left = 0;  
 for (int i = 0; i < int\_shuang\_ying\_hao.size() - 1; i++) {  
 int le = int\_shuang\_ying\_hao.get(i);  
 int ri = int\_shuang\_ying\_hao.get(i + 1);  
  
 StringBuilder sb = new StringBuilder();  
 if (le > left) {  
 for (int j = left; j < le; j++) {  
 if (chars[j] != '\_') {  
 sb.append(chars[j]);  
 } else {  
 if (sb.length() != 0) {  
 strings.add(sb.toString());  
 sb.setLength(0);  
 }  
 }  
 }  
 }  
  
 for (int j = le + 1; j < ri; j++) {  
 sb.append(chars[j]);  
 }  
 strings.add(sb.toString());  
 sb.setLength(0);  
  
 if (i == int\_shuang\_ying\_hao.size() - 2 && ri < chars.length - 1) {  
 for (int j = ri + 1; j < chars.length; j++) {  
 if (chars[j] != '\_') {  
 sb.append(chars[j]);  
 } else {  
 if (sb.length() != 0) {  
 strings.add(sb.toString());  
 sb.setLength(0);  
 }  
 }  
 }  
 }  
  
  
 left = ri + 1;  
  
 i++;  
 }  
  
 StringBuilder sb = new StringBuilder();  
 boolean flag = false;  
 for (int i = 0; i < strings.size(); i++) {  
 if (i == 0) {  
 sb.append(strings.get(i));  
 continue;  
 }  
 if (i == K) {  
 sb.append("\_").append("\*\*\*\*\*\*");  
 flag = true;  
 continue;  
 }  
 sb.append("\_").append("".equals(strings.get(i)) ? "\"\"" : strings.get(i));  
 }  
 System.*out*.println(flag ? sb : "ERROR");  
 }

# ****【算法题】报数游戏****

import java.util.ArrayList;  
import java.util.Scanner;  
  
*/\*\*  
 \* 标题：报数游戏 | 时间限制：1秒 | 内存限制：262144K | 语言限制：不限  
 \* 100个人围成一圈，每个人有一个编码，编号从1开始到100。他们从1开始依次报数，报到为M的人自动退出圈圈，然后下一个人接着从1开始报数，直到剩余的人数小于M。请问最后剩余的人在原先的编号为多少？  
 \* 输入描述:  
 \* 输入一个整数参数M  
 \* 输出描述:  
 \* 如果输入参数M小于等于1或者大于等于100，输出“ERROR!”；否则按照原先的编号从小到大的顺序，以英文逗号分割输出编号字符串  
 \* 示例1  
 \* 输入  
 \* 3  
 \* 输出  
 \* 58,91  
 \* 说明  
 \* 输入M为3，最后剩下两个人  
 \* 示例2  
 \* 输入  
 \* 4  
 \* 输出  
 \* 34,45,97  
 \* 说明  
 \* 输入M为4，最后剩下三个人  
 \*  
 \** ***@since*** *2022年5月11日  
 \*/*public class M\_N\_T\_37 {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.*in*);  
 int M = scanner.nextInt();  
  
 if (M <= 1 || M >= 100) {  
 System.*out*.println("ERROR!");  
 return;  
 }  
  
// int[] rens = new int[200];  
  
 ArrayList<String> strings = new ArrayList<>();  
  
 for (int i = 0; i < 100; i++) {  
// rens[i] = i + 1;  
 strings.add(i + 1 + "");  
 }  
  
// extracted(rens, 0, 0, 0);  
  
 *extracted*(strings, M - 1, M);  
  
 StringBuilder stringBuilder = new StringBuilder();  
 for (String string : strings) {  
 if (stringBuilder.length() == 0) {  
 stringBuilder.append(string);  
 continue;  
 }  
 stringBuilder.append(",").append(string);  
 }  
  
 System.*out*.println(stringBuilder);  
 }  
  
 private static void extracted(ArrayList<String> strings, int begin, int M) {  
 if (strings.size() < M) {  
 return;  
 }  
 int index = begin;  
 for (int i = 0; i < strings.size(); i++) {  
 if (i != index) {  
 continue;  
 }  
 strings.remove(i);  
 strings.add(i, "-1");  
 index += M;  
 }  
  
 int next\_begin = index - strings.size();  
 strings.removeIf("-1"::equals);  
 *extracted*(strings, next\_begin, M);  
 }  
}

# ****【算法题】最大括号深度****

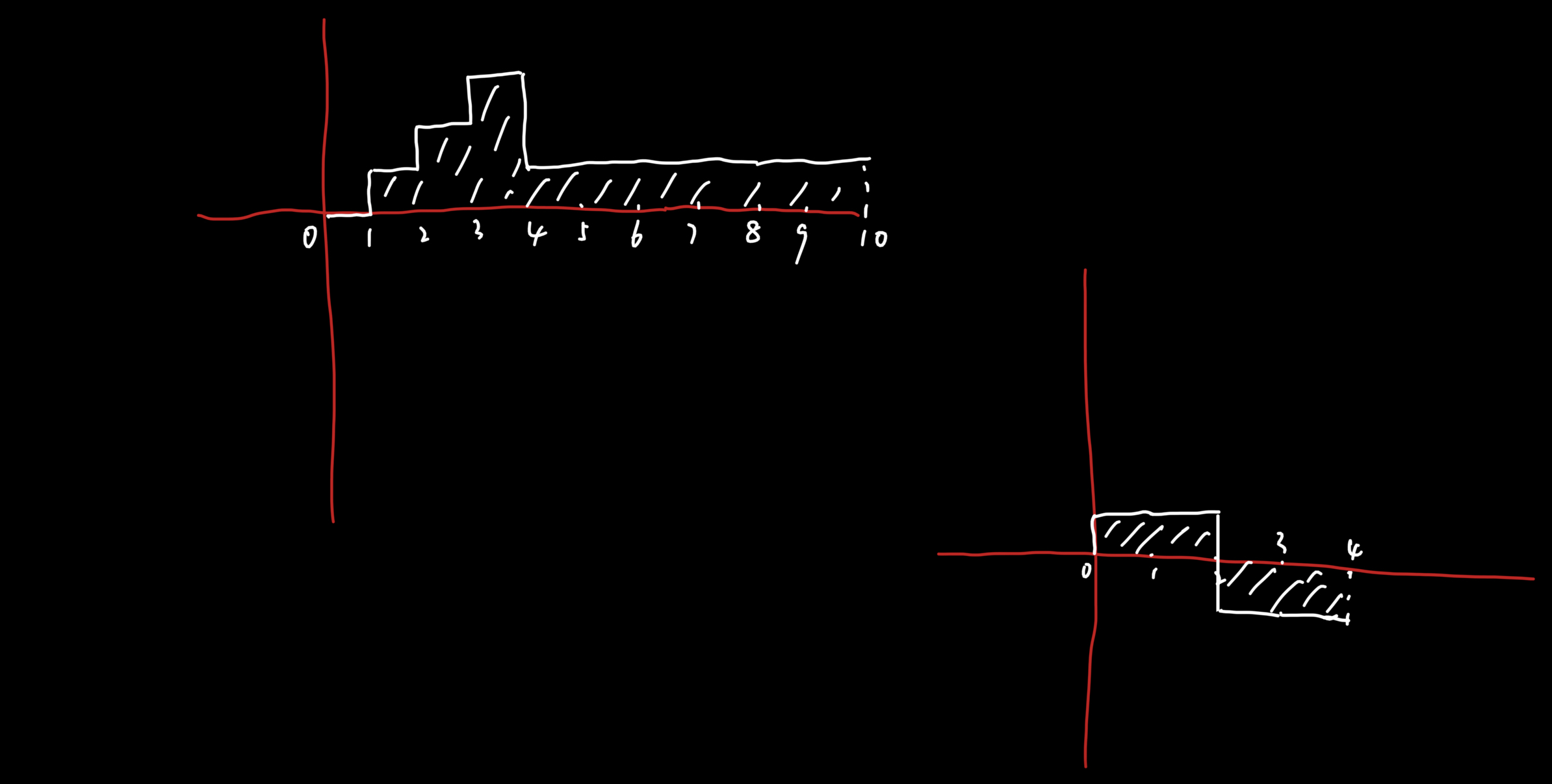
import java.util.Scanner;  
import java.util.Stack;  
  
/\*\*  
 \* 标题：最大括号深度 | 时间限制：1秒 | 内存限制：262144K | 语言限制：不限  
 \* 现有一字符串仅由 '('，')'，'{'，'}'，'['，']'六种括号组成。  
 \* 若字符串满足以下条件之一，则为无效字符串：  
 \* ①任一类型的左右括号数量不相等；  
 \* ②存在未按正确顺序（先左后右）闭合的括号。  
 \* 输出括号的最大嵌套深度，若字符串无效则输出0。  
 \* 0≤字符串长度≤100000  
 \* 输入描述:  
 \* 一个只包括 '('，')'，'{'，'}'，'['，']'的字符串  
 \* 输出描述:  
 \* 一个整数，最大的括号深度  
 \* 示例1  
 \* 输入  
 \* []  
 \* 输出  
 \* 1  
 \* 说明  
 \* 有效字符串，最大嵌套深度为1  
 \* 示例2  
 \* 输入  
 \* ([]{()})  
 \* 输出  
 \* 3  
 \* 说明  
 \* 有效字符串，最大嵌套深度为3  
 \* 示例3  
 \* 输入  
 \* (]  
 \* 输出  
 \* 0  
 \* 说明  
 \* 无效字符串，有两种类型的左右括号数量不相等  
 \* 示例4  
 \* 输入  
 \* ([)]  
 \* 输出  
 \* 0  
 \* 说明  
 \* 无效字符串，存在未按正确顺序闭合的括号  
 \* 示例5  
 \* 输入  
 \* )(  
 \* 输出  
 \* 0  
 \* 说明  
 \* 无效字符串，存在未按正确顺序闭合的括号  
 \*/  
public class M\_N\_T\_38 {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.in);  
 String line = scanner.nextLine();  
 char[] chars = line.toCharArray();  
  
 deal(chars);  
 }  
  
 private static void deal(char[] chars) {  
 Stack<Character> stack = new Stack<>();  
 int deep\_max = 0;  
 for (char aChar : chars) {  
 if (aChar == '(' || aChar == '[' || aChar == '{') {  
 stack.push(aChar);  
 deep\_max = Math.max(stack.size(), deep\_max);  
 continue;  
 }  
  
 if (stack.size() == 0) {  
 System.out.println(0);  
 return;  
 }  
  
 Character character = stack.pop();  
  
 if (aChar == ')') {  
 if (character == '(') {  
 continue;  
 }  
 System.out.println(0);  
 return;  
 }  
  
 if (aChar == ']') {  
 if (character == '[') {  
 continue;  
 }  
 System.out.println(0);  
 return;  
 }  
  
 if (aChar == '}') {  
 if (character == '{') {  
 continue;  
 }  
 System.out.println(0);  
 return;  
 }  
 }  
  
 System.out.println(stack.size() == 0 ? deep\_max : 0);  
 }  
}

# ****【算法题】输出字符串中包含所有整数的最小和****

import java.util.Scanner;  
  
*/\*\*  
 \* 输入字符串s输出s中包含所有整数的最小和  
 \* 说明：  
 \* 字符串s只包含az,AZ,+,-，  
 \* 合法的整数包括正整数，一个或者多个0-9组成，如：0,2,3,002,102  
 \* 负整数，负号开头，数字部分由一个或者多个0-9组成，如-2,-012,-23,-00023  
 \* 输入描述：  
 \* 包含数字的字符串  
 \* 输出描述：  
 \* 所有整数的最小和  
 \* 示例  
 \* 输入：  
 \* bb1234aa  
 \* 输出：  
 \* 10  
 \* 输入：  
 \* bb12-34aa  
 \* 输出：  
 \* -31  
 \* 说明：  
 \* 1+2-(34)=-31  
 \*/*public class OD\_Z\_T\_01 {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.*in*);  
 String line = scanner.nextLine();  
 char[] chars = (line + "+").toCharArray();  
  
 boolean flag = false; // 定义一个标记,是否是‘-’后面的数字，如果是的话就把数字串起来最大  
 StringBuilder sb = new StringBuilder(); // 用来拼接负数字符  
 int sum\_min = 0;  
 for (char aChar : chars) {  
 if (Character.*isDigit*(aChar)) { //判断是否为数字  
 if (flag) {  
 sb.append(aChar); // 如果负数就先拼接  
 } else {  
 sum\_min += aChar - 48; // 其他符号后面的直接计算，才能得到最小正数  
 }  
  
 continue;  
 }  
  
 if (flag && sb.length() != 0) { // 把前面拼接好的字符串计算出来，并且置空  
 sum\_min -= Integer.*parseInt*(sb.toString());  
 sb.setLength(0);  
 }  
  
 flag = aChar == '-'; // 遇到负号改变标记  
 }  
  
 System.*out*.println(sum\_min);  
 }  
}

# ****【算法题】计算面积****

import java.util.Scanner;  
  
*/\*\*  
 \* '  
 \* 标题：计算面积 | 时间限制：1秒 | 内存限制：262144K | 语言限制：不限  
 \* 绘图机器的绘图笔初始位置在原点（0, 0），机器启动后其绘图笔按下面规则绘制直线：  
 \* 1）尝试沿着横向坐标轴正向绘制直线，直到给定的终点值E。  
 \* 2）期间可通过指令在纵坐标轴方向进行偏移，并同时绘制直线，偏移后按规则1 绘制直线；指令的格式为X offsetY，表示在横坐标X 沿纵坐标方向偏移，offsetY为正数表示正向偏移，为负数表示负向偏移。  
 \* <p>  
 \* 给定了横坐标终点值E、以及若干条绘制指令，请计算绘制的直线和横坐标轴、以及 X=E 的直线组成图形的面积。  
 \* <p>  
 \* 输入描述:  
 \* 首行为两个整数 N E，表示有N条指令，机器运行的横坐标终点值E。  
 \* 接下来N行，每行两个整数表示一条绘制指令X offsetY，用例保证横坐标X以递增排序方式出现，且不会出现相同横坐标X。  
 \* 取值范围：0 < N <= 10000, 0 <= X <= E <=20000, -10000 <= offsetY <= 10000。  
 \* 输出描述:  
 \* 一个整数，表示计算得到的面积，用例保证，结果范围在0~4294967295内  
 \* 示例1  
 \* 输入  
 \* 4 10  
 \* 1 1  
 \* 2 1  
 \* 3 1  
 \* 4 -2  
 \* 输出  
 \* 12  
 \* 示例2  
 \* 输入  
 \* 2 4  
 \* 0 1  
 \* 2 -2  
 \* 输出  
 \* 4  
 \*/*



public class Python\_M\_N\_T\_5 {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.*in*);  
 String line = scanner.nextLine();  
 String[] strings = line.split(" ");  
  
 int N = Integer.*parseInt*(strings[0]);  
 int E = Integer.*parseInt*(strings[1]);  
  
 int Y = 0;  
 int sum = 0;  
 int[][] ints = new int[N][2];  
 for (int i = 0; i < ints.length; i++) {  
 String[] splits = scanner.nextLine().split(" ");  
 ints[i][0] = Integer.*parseInt*(splits[0]);  
 ints[i][1] = Integer.*parseInt*(splits[1]);  
 }  
  
 for (int i = 0; i < E; i++) {  
  
 for (int[] anInt : ints) {  
 if (anInt[0] == i) {  
 Y += anInt[1];  
 break;  
 }  
 }  
  
 sum += Math.*abs*(Y);  
  
 }  
  
 System.*out*.println(sum);  
  
  
 }  
}

# ****【算法题】最大矩阵和****

*/\*\*  
 \* 标题：最大矩阵和 | 时间限制：1秒 | 内存限制：262144K | 语言限制：不限  
 \* 给定一个二维整数矩阵，要在这个矩阵中选出一个子矩阵，使得这个子矩阵内所有的数字和尽量大，我们把这个子矩阵称为和最大子矩阵，子矩阵的选取原则是原矩阵中一块相互连续的矩形区域。  
 \* 输入的第一行包含2个整数n, m(1 <= n, m <= 10)，表示一个n行m列的矩阵，下面有n行，每行有m个整数，同一行中，每2个数字之间有1个空格，最后一个数字后面没有空格，所有的数字的在[-1000, 1000]之间。  
 \* 输出描述:  
 \* 输出一行一个数字，表示选出的和最大子矩阵内所有的数字和。  
 \* 示例1  
 \* 输入  
 \* 3 4  
 \* -3 5 -1 5  
 \* 2 4 -2 4  
 \* -1 3 -1 3  
 \* 输出  
 \* 20  
 \* 说明  
 \* 一个3\*4的矩阵中，后面3列的子矩阵求和加起来等于20，和最大。  
 \*/*public class M\_N\_T\_32 {  
 //这一题的思路是：由一维的连续子数组最大和，然后扩展到上下界的遍历，则可计算二维问题。//复杂度O(N^3)  
 public static void main(String[] args) {  
 Scanner sc = new Scanner(System.*in*);  
 int rows = sc.nextInt();//行数  
 int cols = sc.nextInt();//列数  
 int res = Integer.*MIN\_VALUE*;  
 int[][] matrix = new int[rows][cols];  
 for (int i = 0; i <= rows - 1; ++i) {  
 for (int j = 0; j <= cols - 1; ++j) {  
 matrix[i][j] = sc.nextInt();  
 }  
 }  
  
  
 for (int begin = 0; begin <= rows - 1; ++begin) {//上边界  
 int[] line = new int[cols];//每一列之和 //修改上边界之后，要清空重新开始  
 for (int end = begin; end <= rows - 1; ++end) {//下边界  
 //计算列元素和  
 for (int j = 0; j <= cols - 1; ++j) {  
 line[j] += matrix[end][j];//下边界每向下一行，就计算更新下line[]数组的值  
 }  
 res = Math.*max*(res, line[0]);  
 int sum = 0;  
 for (int j = 0; j <= cols - 1; ++j) {  
 sum += line[j];  
 res = Math.*max*(res, sum);//取最大  
 if (sum < 0) sum = 0;//小于零，置零 //这样意味着最终的值可能是负数  
 }  
 }  
 }  
 System.*out*.println(res);  
 }  
}