```

题号 题目 知识点 难度 通过率

XM1 小米Git 树 困难 10.22%

XM2 懂二进制 中等 29.90%

XM3 风口的猪-中国牛市 动态规划 较难 18.73%

XM4 电话号码分身 中等 22.63%

XM5 句子反转 字符串 中等 26.09%

XM6 树的高度 中等 16.85%

XM7 进制间转换 中等 24.29%

XM8 最少立方数之和 动态规划 中等 24.28%

XM9 资产包打包 动态规划贪心 中等 33.55%

XM10 爬楼梯 递归动态规划 中等 14.17%

XM11 构建短字符串 动态规划字符串 中等 33.70%

XM12 序列模式匹配 字符串动态规划 中等 20.28%

XM13 密码破译 字符串模拟穷举动态规划 中等 28.49%

XM14 ipv4地址白名单 字符串数组 中等 32.28%

XM15 比赛名次 图排序 中等 28.34%

XM16 地鼠逃跑计划 动态规划队列 中等 38.55%

XM17 找出单向链表中的一个节点，该节点到尾指针的距离为K 数组链表 中等 24.22%

XM18 数组操作 排序数组 中等 21.34%

XM19 一组带数字编号的球，其中有两个编号只出现了一次，把它们找出来 中等 19.79%

XM20 旋转数组中的最小元素 数组排序 中等 20.62%

XM21 求数列第n项 模拟 中等 16.88%

XM22 计算题 贪心模拟栈 中等 21.04%

XM23 设计一个函数1 字符串数组模拟穷举 中等 27.65%

XM24 找&ldquo;异数&rdquo; 哈希数学 中等 15.04%

XM25 求整数的阶乘 模拟 中等 23.48%

XM26 最大新整数 动态规划贪心 中等 21.05%

XM27 计算原子的个数 字符串模拟栈 中等 33.45%

XM28 厨艺大赛奖金 贪心排序 中等 25.28%

XM29 扑克牌四则运算 模拟穷举 中等 23.71%

XM30 小米大礼包 动态规划 中等 16.00%

XM31 最优分割 动态规划贪心 中等 27.59%

XM32 如何添加运算符 穷举 中等 41.05%

XM33 小明的字符串 模拟字符串 中等 33.50%

XM34 分布式集群消息传递 图 中等 29.31%

XM35 CCNumber 模拟 中等 14.69%

XM36 获取n维数组的最大深度 字符串模拟栈 中等 36.13%

XM37 爬楼梯2 递归动态规划 中等 23.63%

XM38 设计一个函数2 字符串模拟树 中等 28.51%

XM39 集合合并 图数组模拟 中等 15.14%

XM40 升级蓄水池 动态规划贪心模拟 中等 13.83%

```

## XM1 小米Git

#### 题目描述

```

git是一种分布式代码管理工具，git通过树的形式记录文件的更改历史，比如： base'<--base<--A<--A' ^ | --- B<--B' 小米工程师常常需要寻找两个分支最近的分割点，即base.假设git 树是多叉树，请实现一个算法，计算git树上任意两点的最近分割点。 （假设git树节点数为n,用邻接矩阵的形式表示git树：字符串数组matrix包含n个字符串，每个字符串由字符'0'或'1'组成，长度为n。matrix[i][j]=='1'当且仅当git树种第i个和第j个节点有连接。节点0为git树的根节点。）

```

#### 示例1

#### 输入

```

[01011,10100,01000,10000,10000],1,2

```

#### 返回值

```

1

```

```java

public class Solution {

/\*\*

\* 返回git树上两点的最近分割点

\* \* @param matrix 接邻矩阵，表示git树，matrix[i][j] == '1' 当且仅当git树中第i个和第j个节点有连接，节点0为git树的跟节点

\* indexA 节点A的index

\* indexB 节点B的index

\* @return 整型

\*/

public void dfs(int node, int father, int dep, int[] path, int[] depth, String[] mat) {

depth[node] = dep;

path[node] = father;

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

if (i != father && mat[node].charAt(i) == '1') dfs(i, node, dep + 1, path, depth, mat);

}

}

public int getSplitNode(String[] matrix, int indexA, int indexB) {

int n = matrix.length;

int[] depth = new int[n];

int[] father = new int[n];

father[0] = -1;

dfs(0, -1, 0, father, depth, matrix);

while (depth[indexA] > depth[indexB]) {

indexA = father[indexA];

}

while (depth[indexB] > depth[indexA]) {

indexB = father[indexB];

}

while (indexA != indexB) {

indexA = father[indexA];

indexB = father[indexB];

}

return indexA;

}

}

```

## XM2 懂二进制

#### 题目描述

```

世界上有10种人，一种懂二进制，一种不懂。那么你知道两个int32整数m和n的二进制表达，有多少个位(bit)不同么？

```

#### 示例1

#### 输入

```

1999 2299

```

#### 返回值

```

7

```

```java

public class Solution {

/\*\*

\* 获得两个整形二进制表达位数不同的数量

\* \* @param m 整数m

\* @param n 整数n

\* @return 整型

\*/

public int countBitDiff(int m, int n) {

String sm = Integer.toBinaryString(m);

String sn = Integer.toBinaryString(n);

int length\_sm = sm.length();

int length\_sn = sn.length();

if (length\_sm != length\_sn) {

if (length\_sm < length\_sn) {

StringBuilder sb = new StringBuilder();

for (int i = 0; i < length\_sn - length\_sm; i++) {

sb.append("0");

}

sb.append(sm);

sm = sb.toString();

}

if (length\_sm > length\_sn) {

StringBuilder sb = new StringBuilder();

for (int i = 0; i < length\_sm - length\_sn; i++) {

sb.append("0");

}

sb.append(sn);

sn = sb.toString();

}

}

int count = 0;

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

if (sm.charAt(i) != sn.charAt(i)) count++;

}

return count;

}

}

```

## XM3 风口的猪-中国牛市

#### 题目描述

```

风口之下，猪都能飞。当今中国股市牛市，真可谓“错过等七年”。 给你一个回顾历史的机会，已知一支股票连续n天的价格走势，以长度为n的整数数组表示，数组中第i个元素（prices[i]）代表该股票第i天的股价。 假设你一开始没有股票，但有至多两次买入1股而后卖出1股的机会，并且买入前一定要先保证手上没有股票。若两次交易机会都放弃，收益为0。 设计算法，计算你能获得的最大收益。 输入数值范围：2<=n<=100,0<=prices[i]<=100

```

#### 示例1

#### 输入

```

3,8,5,1,7,8

```

#### 返回值

```

12

```

```java

import java.math.\*;

public class Solution {

/\*\*

\* 计算你能获得的最大收益

\* @param prices Prices[i]即第i天的股价

\* @return 整型

\*/

public int calculateMax(int[] prices) {

int firstbuy = Integer.MIN\_VALUE;

int secondbuy = Integer.MIN\_VALUE;

int firstsell = 0;

int secondsell = 0;

for (int price : prices) {

firstbuy = Math.max(firstbuy, -price);

firstsell = Math.max(firstsell, firstbuy + price);

secondbuy = Math.max(secondbuy, firstsell - price);

secondsell = Math.max(secondsell, secondbuy + price);

}

return secondsell;

}

}

```

## XM4 电话号码分身

#### 题目描述

```

继MIUI8推出手机分身功能之后，MIUI9计划推出一个电话号码分身的功能：首先将电话号码中的每个数字加上8取个位，然后使用对应的大写字母代替 （"ZERO", "ONE", "TWO", "THREE", "FOUR", "FIVE", "SIX", "SEVEN", "EIGHT", "NINE"）， 然后随机打乱这些字母，所生成的字符串即为电话号码对应的分身。

```

#### 输入描述:

```

第一行是一个整数T（1 ≤ T ≤ 100)表示测试样例数；接下来T行，每行给定一个分身后的电话号码的分身（长度在3到10000之间）。

```

#### 输出描述:

```

输出T行，分别对应输入中每行字符串对应的分身前的最小电话号码（允许前导0）。

```

#### 示例1

#### 输入

```

4

EIGHT

ZEROTWOONE

OHWETENRTEO

OHEWTIEGTHENRTEO

```

#### 输出

```

0

234

345

0345

```

```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));

char[] comArr = {'Z', 'O', 'W', 'T', 'U', 'F', 'X', 'S', 'G', 'I'};

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

String[] str = new String[n];

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

char[] charArr = br.readLine().toCharArray();

int[] count = new int[26];

int[] result = new int[10];

for (char c : charArr) count[c - 'A']++;

for (int j = 0; j < comArr.length; j += 2) {

result[(j + 2) % 10] = count[comArr[j] - 'A'];

}

result[3] = count['O' - 'A'] - result[6] - result[2] - result[4];

result[5] = count['T' - 'A'] - result[4] - result[0];

result[7] = count['F' - 'A'] - result[6];

result[9] = count['S' - 'A'] - result[8];

result[1] = count['I' - 'A'] - result[7] - result[0] - result[8];

StringBuilder sb = new StringBuilder();

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

for (int k = 0; k < result[j]; k++) sb.append(j);

}

System.out.println(sb);

}

}

}

```

## XM5 句子反转

#### 题目描述

```

给定一个句子（只包含字母和空格）， 将句子中的单词位置反转，单词用空格分割, 单词之间只有一个空格，前后没有空格。 比如： （1） “hello xiao mi”-> “mi xiao hello”

```

#### 输入描述:

```

输入数据有多组，每组占一行，包含一个句子(句子长度小于1000个字符)

```

#### 输出描述:

```

对于每个测试示例，要求输出句子中单词反转后形成的句子

```

#### 示例1

#### 输入

```

hello xiao mi

```

#### 输出

```

mi xiao hello

```

```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 brs = new BufferedReader(new InputStreamReader(System.in));

String inputString = brs.readLine();

String[] numbers = inputString.split(" ");

// 字符分割后，倒序输出即可

for (int i = numbers.length - 1; i >= 0; i--) {

if (i == 0) System.out.print(numbers[i]);

else System.out.print(numbers[i] + " ");

}

}

}

```

## XM6 树的高度

#### 题目描述

```

现在有一棵合法的树，树的节点都是用数字表示，现在给定这棵树上所有的父子关系，求这棵树的高度

```

#### 输入描述:

```

输入的第一行表示节点的个数n（1 ≤ n ≤ 1000，节点的编号为0到n-1）组成，

下面是n-1行，每行有两个整数，第一个数表示父节点的编号，第二个数表示子节点的编号

```

#### 输出描述:

```

输出树的高度，为一个整数

```

#### 示例1

#### 输入

```

5

0 1

0 2

1 3

1 4

```

#### 输出

```

3

```

```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));

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

int[] height = new int[n];

int[] getNodes = new int[n];

height[0] = 1;

int max = 0;

for (int i = 0; i < n - 1; i++) {

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

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

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

getNodes[parent]++;

if (getNodes[parent] < 3) height[child] = height[parent] + 1;

max = Math.max(max, height[child]);

}

System.out.println(max);

}

}

```

## XM7 进制间转换

#### 题目描述

```

设计一个函数， 可以将任意十进制的数， 转换成任意2到9的进制表示的形式

```

#### 输入描述:

```

需要转换的数字x(0<=x<=1e18) 转换使用的进制k(2<=k<=9)

```

#### 输出描述:

```

对应进制的结果

```

#### 示例1

#### 输入

```

33 2

```

#### 输出

```

100001

```

```java

import java.util.\*;

import java.io.\*;

public class Main {

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

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

String str = cin.readLine();

System.out.print(getConversion(str));

}

public static String getConversion(String str) {

String[] num = str.split(" ");

long x = Long.parseLong(num[0]);

long k = Long.parseLong(num[1]);

StringBuilder str1 = new StringBuilder();

do {

str1.append(x % k);

x = x / k;

} while (x >= k);

char[] ch = str1.toString().toCharArray();

StringBuilder str2 = new StringBuilder();

for (int i = ch.length - 1; i >= 0; i--) {

str2.append(ch[i]);

}

if (x == 0) return str2.toString();

else return x + "" + str2;

}

}

```

## XM8 最少立方数之和

#### 题目描述

```

给出一个数字N（0<N<1000000）,将N写成立方数和的形式，求出需要的最少立方数个数。

例如N=17，1+8+8 = 17，最少需要3个立方数，则输出3。

N= 28,1+1+1+1+8+8+8=28, 需要7个立方数，1+27=28,需要2个立方数，所以最少立方数为2，则输出2。

```

#### 输入描述:

```

一个数字N（0<N<1000000）

```

#### 输出描述:

```

最少立方数个数

```

#### 示例1

#### 输入

```

28

```

#### 输出

```

2

```

```java

import java.util.\*;

public class Main {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int N = sc.nextInt();

int[] dp = new int[N + 1];

dp[0] = 0;

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

int min = Integer.MAX\_VALUE;

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

min = Math.min(min, dp[i - j \* j \* j]);

}

dp[i] = min + 1;

}

System.out.println(dp[N]);

}

}

```

## XM9 资产包打包

#### 题目描述

```

在金融资产交易中，经常涉及到资产包的挑选打包。在资产包打包过程中，每种类型的资产有固定的数量与价值，需选择某几种资产打包，使得资产包总价值最大。打包时每种资产只能整体打包，不能分割。假设现有可容纳M条资产的资产包，另外有N种资产。资产Na数量为Ta条，总价值为Va元；资产Nb数量为Tb条，总价值为Vb元；资产Nc数量为Tc条，总价值为Vc元......；资产Nn数量为Tn，总价值为Vn。编写算法，挑选哪些类型资产放入资产包可使得资产包总价值最大？

```

#### 输入描述:

```

资产总量,资产种类,每类资产条数,每类资产价值(逗号分隔)；其中每类资产条数与每类资产价值为空格分隔。

总格式如下：

资产总量,资产种类,资产A条数 资产B条数 资产C条数,资产A价值 资产B价值 资产C价值！

举例，资产总量为12，资产种类3种，3种资产条数分别为4，5，7，三种资产价值分别是500元，600元，800元，输入如下：

12,3,4 5 7,500 600 800

资产总量和资产种类都不超过1000，资产条数不超过1000，资产价值不超过10000，所有数值均为正整数。

```

#### 输出描述:

```

资产包中资产最大总价值

```

#### 示例1

#### 输入

```

12,3,4 5 7,500 600 800

```

#### 输出

```

1400

```

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

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

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

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

String[] st = s[2].split(" ");

String[] sv = s[3].split(" ");

int[] t = new int[n];

int[] v = new int[n];

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

t[i] = Integer.parseInt(st[i]);

v[i] = Integer.parseInt(sv[i]);

}

int[] arr = new int[m + 1];

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

for (int j = m; j >= t[i - 1]; j--) {

arr[j] = Math.max(arr[j], arr[j - t[i - 1]] + v[i - 1]);

}

}

System.out.println(arr[m]);

}

}

```

## XM10 爬楼梯

#### 题目描述

```

在你面前有一个n阶的楼梯，你一步只能上1阶或2阶。

请问计算出你可以采用多少种不同的方式爬完这个楼梯。

```

#### 输入描述:

```

一个正整数n(n<=100)，表示这个楼梯一共有多少阶

```

#### 输出描述:

```

一个正整数，表示有多少种不同的方式爬完这个楼梯

```

#### 示例1

#### 输入

```

5

```

#### 输出

```

8

```

```java

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.math.BigInteger;

public class Main {

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

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

int i = Integer.parseInt(bufferedReader.readLine());

if (i < 3) {

System.out.println(i);

return;

}

BigInteger a = BigInteger.ONE;

BigInteger b = BigInteger.valueOf(2);

BigInteger c = BigInteger.ZERO;

for (int j = 3; j <= i; j++) {

c = a.add(b);

a = b;

b = c;

}

System.out.println(c);

}

}

```

## XM11 构建短字符串

#### 题目描述

```

给定任意一个较短的子串，和另一个较长的字符串，判断短的字符串是否能够由长字符串中的字符构建出来，且长串中的每个字符只能用一次。

```

#### 输入描述:

```

一行数据包括一个较短的字符串S和一个较长的字符串T，用一个空格分隔。保证1<=|S|<=|T|<=100000。

```

#### 输出描述:

```

如果短的字符串可以由长字符串中的字符构建出来，输出字符串 “true”，否则输出字符串 "false"。

```

#### 示例1

#### 输入

```

a b

```

#### 输出

```

false

```

#### 示例2

#### 输入

```

fj jfiejfiejfie

```

#### 输出

```

true

```

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

String s1 = s[0];

String s2 = s[1];

int[] count = new int[26];

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

count[s2.charAt(i) - 'a']++;

}

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

count[s1.charAt(i) - 'a']--;

}

for (int i : count) {

if (i < 0) {

System.out.println(false);

return;

}

}

System.out.println(true);

}

}

```

## XM12 序列模式匹配

#### 题目描述

```

给定文本text和待匹配字符串pattern，二者皆只包含小写字母，并且不为空。

在text中找出匹配pattern的最短字符串，匹配指按序包含pattern，但不要求pattern连续。

如text为abaacxbcbbbbacc，pattern为cbc，text中满足条件的是abaacxbcbbbbacc下划线部分。

```

#### 输入描述:

```

多行，每行一个text和一个pattern，用空格分隔。

保证1<=|text|,|pattern|<=1000，Σ|text|,Σ|pattern|<=10000。

```

#### 输出描述:

```

输出最短匹配序列起止位置（位置下标从0开始），用空格分隔。若有多个答案，输出起止位置最小的答案；若无满足条件的答案，则起止均为-1。

```

#### 示例1

#### 输入

```

abaacxbcbbbbacc cbc

abc x

aaabcac ac

```

#### 输出

```

4 7

-1 -1

5 6

```

```java

import java.io.\*;

public class Main {

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

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

String s;

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

String[] str = s.split(" ");

char[] text = str[0].toCharArray();

char[] pattern = str[1].toCharArray();

int[] ans = new int[]{-1, -1};

boolean b = true;

for (int i = 0; i <= text.length - pattern.length; i++) {

boolean match = true;

int j = i, k = 0;

if (text[j] != pattern[0]) continue;

while (j < text.length) {

if (text[j] == pattern[k]) k++;

j++;

if (k == pattern.length) {

if (b) {

ans = new int[]{i, j - 1};

b = false;

break;

} else {

if (j - 1 - i < ans[1] - ans[0]) ans = new int[]{i, j - 1};

}

break;

}

}

}

System.out.println(ans[0] + " " + ans[1]);

}

}

}

```

## XM13 密码破译

#### 题目描述

```

我们来做一个简单的密码破译游戏。破译的规则很简单，将数字转换为字母，1转化为a，2转化为b，依此类推，26转化为z。现在输入的密码是一串数字，输出的破译结果是该数字串通过转换规则所能产生的所有字符串。

```

#### 输入描述:

```

多行数据，每行为一个数字串。

保证数字串的总长度不超过1000，每行数据的答案至少有1个且不超过1000个。

```

#### 输出描述:

```

多行数据，每行对应输出通过数字串破译得到的所有字符串，并按照字符串顺序排列，字符串之间用单个空格分隔。每行开头和结尾不允许有多余的空格。

```

#### 示例1

#### 输入

```

1

12

123

```

#### 输出

```

a

ab l

abc aw lc

```

```java

import java.util.\*;

public class Main {

static List<String> ans;

static String[] maps = {"a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l", "m", "n", "o", "p", "q", "r", "s", "t", "u", "v", "w", "x", "y", "z"};

public static void main(String[] args) {

Scanner scan = new Scanner(System.in);

while (scan.hasNext()) {

String s = scan.nextLine();

ans = new ArrayList<>();

dfs(s, 0, "");

for (int i = 0; i < ans.size(); i++) {

if (i < ans.size() - 1) System.out.print(ans.get(i) + " ");

else System.out.println(ans.get(i));

}

}

}

// 怎么处理0的问题

public static void dfs(String s, int index, String tmp) {

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

ans.add(tmp);

} else {

if (index < s.length() - 1 && s.charAt(index + 1) == '0') {

dfs(s, index + 2, tmp + maps[Integer.parseInt(s.substring(index, index + 2)) - 1]);

} else {

// aa0的时候要注意

dfs(s, index + 1, tmp + maps[s.charAt(index) - '0' - 1]);

if (index < s.length() - 1 && Integer.parseInt(s.substring(index, index + 2)) <= 26) {

if (index >= s.length() - 2 || s.charAt(index + 2) != '0')

dfs(s, index + 2, tmp + maps[Integer.parseInt(s.substring(index, index + 2)) - 1]);

}

}

}

}

}

```

## XM14 ipv4地址白名单

#### 题目描述

```

我们的小齐同学是一名很辛苦的实习DBA，他每天的工作就是为一个帐号添加授权，今天给这200个ipv4添加授权，明天又要把这200个授权删掉，有一天小齐同学在删除授权的时候不小心把所有的授权都删了，被领导很批了一顿。痛定思痛，小齐同学开始反思他每天的工作，发现无非就是我每天要让那些ip访问数据库而已，他决定写一个效率很高的ip白名单，请帮小齐同学说一下实现思路，并用结构化编程语言（c/c++/python/golang/java等）写一个ip白名单吧，他需要这个白名单有添加ip的功能，删除ip的功能，查找这个ip在不在白名单中，以及打印白名单中的内容，以上四个功能中查找ip是否在白名单中效率一定要高。并帮小齐分析一下各个功能的时间复杂度，写的好小齐同学会请你吃饭哦。

```

#### 输入描述:

```

每行一条输入，格式为 type:ip

type 包括 i d s 分别表示添加，删除，查找

以 end 结尾

输入最多不超过100000行

```

#### 输出描述:

```

输出每行一条对应输入

如果是查找，成功请打印true,失败请打印false

如果是添加删除，请打印ok

```

#### 示例1

#### 输入

```

i:127.0.0.1

i:10.0.0.1

s:10.0.0.1

d:10.0.0.1

s:10.0.0.1

s:127.0.0.1

end

```

#### 输出

```

ok

ok

true

ok

false

true

```

#### 示例2

#### 输入

```

i:127.0.0.3

i:127.0.0.3

d:127.0.0.4

s:127.0.0.3

i:127.0.0.5

d:127.0.0.5

s:127.0.0.4

i:127.0.0.4

s:127.0.0.3

d:127.0.0.4

end

```

#### 输出

```

ok

ok

ok

true

ok

ok

false

ok

true

ok

```

```java

import java.io.\*;

import java.util.\*;

public class Main {

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

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

Map<String, Boolean> map = new HashMap<>();

while (true) {

String str = br.readLine();

if (str.equals("end")) return;

String s1 = str.substring(0, 1);

String s2 = str.substring(2, str.length());

if (s1.equals("i")) {

map.put(s2, true);

System.out.println("ok");

} else if (s1.equals("d")) {

map.put(s2, false);

System.out.println("ok");

} else if (s1.equals("s")) {

if (map.get(s2) != null && map.get(s2)) System.out.println("true");

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

}

}

}

}

```

## XM15 比赛名次

#### 题目描述

```

有N个比赛队（1<=N<=500），编号依次为1，2，3，。。。。，N进行比赛，比赛结束后，裁判委员会要将所有参赛队伍从前往后依次排名，但现在裁判委员会不能直接获得每个队的比赛成绩，只知道每场比赛的结果，即P1赢P2，用P1，P2表示，排名时P1在P2之前。现在请你编程序确定排名。

```

#### 输入描述:

```

输入有若干组，每组中的第一行为二个数N（1<=N<=500），M；其中N表示队伍的个数，M表示接着有M行的输入数据。接下来的M行数据中，每行也有两个整数P1，P2表示即P1队赢了P2队。

```

#### 输出描述:

```

给出一个符合要求的排名。输出时队伍号之间有空格，最后一名后面没有空格。

其他说明：符合条件的排名可能不是唯一的，此时要求输出时编号小的队伍在前；输入数据保证是正确的，即输入数据确保一定能有一个符合要求的排名。

```

#### 示例1

#### 输入

```

4 3

1 2

2 3

4 3

```

#### 输出

```

1 2 4 3

```

```java

// 最小堆 实现 和图的入度 实现 将入度为0的顶点压入最小堆 下一个顶点的选择 根据栈顶元素的边

import java.util.\*;

public class Main {

public static void main(String[] args) {

Scanner scan = new Scanner(System.in);

while (scan.hasNext()) {

int n = scan.nextInt(), m = scan.nextInt();

if (m == 0) {

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

if (i == n) System.out.println(i);

else System.out.print(i + " ");

}

} else {

// 怎么来 统计点的入读

int[] div = new int[n + 1]; // 要满足该点一定存在才行呀

int[][] edges = new int[n + 1][n + 1];

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

int p1 = scan.nextInt(), p2 = scan.nextInt();

div[p2]++;

edges[p1][p2] = 1; // 有向边

}

PriorityQueue<Integer> priorityQueue = new PriorityQueue<>();

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

if (div[i] == 0) priorityQueue.add(i);

}

List<Integer> ans = new ArrayList<>();

while (priorityQueue.size() != 0) {

int p = priorityQueue.poll();

ans.add(p);

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

if (edges[p][j] == 1) {

div[j]--;

edges[p][j] = 0;

if (div[j] == 0) priorityQueue.add(j);

}

}

}

for (int i = 0; i < ans.size(); i++) {

if (i != ans.size() - 1) System.out.print(ans.get(i) + " ");

else System.out.println(ans.get(i));

}

}

}

}

}

```

## XM16 地鼠逃跑计划

#### 题目描述

```

有一只地鼠不小心跑进了一个m\*n的矩形田地里，假设地鼠在这块田地的初始位置为（x,y），并且每次只能向相邻的上下左右四个方向移动一步，那么在最多移动K次的情况下，有多少条路径可以逃出这片田地（一旦出去田地的边界就不能再往回走）？

下面是样例示意图：

```

#### 输入描述:

```

输入数据包括五个参数：m,n,x,y,K

其中m和n的范围均为是[1,10]，K的范围是[0,10]。

0<=x<m,0<=y<n。

```

#### 输出描述:

```

输出成功逃跑的路径数量。

```

#### 示例1

#### 输入

```

2

3

0

1

2

```

#### 输出

```

6

```

```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 m = Integer.parseInt(br.readLine());

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

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

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

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

int[][][] dp = new int[m + 2][n + 2][step + 2];

dp[x + 1][y + 1][0] = 1;

int sum = 0;

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

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

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

dp[i + 1][j][k] += dp[i][j][k - 1];

dp[i - 1][j][k] += dp[i][j][k - 1];

dp[i][j + 1][k] += dp[i][j][k - 1];

dp[i][j - 1][k] += dp[i][j][k - 1];

}

}

for (int i = 0; i <= m + 1; i++) {

sum += dp[i][0][k] + dp[i][n + 1][k];

}

for (int i = 0; i <= n + 1; i++) {

sum += dp[0][i][k] + dp[m + 1][i][k];

}

}

System.out.println(sum);

}

}

```

## XM17 找出单向链表中的一个节点，该节点到尾指针的距离为K

#### 题目描述

```

找出单向链表中的一个节点，该节点到尾指针的距离为K。链表的倒数第0个结点为链表的尾指针。要求时间复杂度为O(n)。

链表结点定义如下：

struct ListNode

{

int m\_nKey;

ListNode\* m\_pNext;

}

链表节点的值初始化为1，2，3，4，5，6，7。

```

#### 输入描述:

```

该节点到尾指针的距离K

```

#### 输出描述:

```

返回该单向链表的倒数第K个节点，输出节点的值

```

#### 示例1

#### 输入

```

2

```

#### 输出

```

6

备注:

请自觉实现一个链表，将1到7依次加入链表，然后再寻找倒数第K个节点。要求加节点与找节点的操作复杂度均为O(n)。

```

```java

import java.util.\*;

import java.io.\*;

public class Main {

static class ListNode {

int val;

ListNode next;

public ListNode(int val) {

this.val = val;

this.next = null;

}

}

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

ListNode head = new ListNode(0);

ListNode p = head;

for (int i = 1; i <= 7; ++i) {

p.next = new ListNode(i);

p = p.next;

}

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

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

int n = dfs(head.next, k);

}

public static int dfs(ListNode head, int k) {

if (head == null) return 0;

int cnt = 1 + dfs(head.next, k);

if (cnt == k) System.out.println(head.val);

return cnt;

}

}

```

## XM18 数组操作

#### 题目描述

```

输入一个无序整数数组，调整数组中数字的顺序， 所有偶数位于数组的前半部分，使得所有奇数位于数组的后半部分。

要求时间复杂度为O(n)。

```

#### 输入描述:

```

给定无序数组。

长度不超过1000000。

```

#### 输出描述:

```

所有偶数位于数组的前半部分，所有奇数位于数组的后半部分。

如果有多个答案可以输出任意一个正确答案。

```

#### 示例1

#### 输入

```

2 4 5 7 8 1

```

#### 输出

```

2 4 8 7 5 1

备注:

请自觉使用O(n)的算法。

```

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

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

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

int x = 0;

int y = a.length - 1;

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

int b = Integer.parseInt(str[i]);

if (b % 2 == 0) a[x++] = b;

else a[y--] = b;

}

StringBuffer sb = new StringBuffer();

for (int i = 0; i < a.length - 1; i++) {

sb.append(a[i]);

sb.append(" ");

}

sb.append(a[a.length - 1]);

System.out.println(sb);

}

}

```

## XM19 一组带数字编号的球，其中有两个编号只出现了一次，把它们找出来

#### 题目描述

```

一组带数字编号的球里除了两个编号之外，其它的编号都出现了两次。

请写程序找出这两个只出现一次的编号。要求时间复杂度是O(n)，空间复杂度是O(1)。

```

#### 输入描述:

```

整形数组

长度不超过1000000

```

#### 输出描述:

```

输出数组中2个只出现了一次的数

先输出较小的数

```

#### 示例1

#### 输入

```

1

2

3

4

5

2

3

4

5

6

```

#### 输出

```

1 6

```

```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));

int[] nums = new int[1000001];

int len = 0;

String str;

int xor = 0;

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

nums[len] = Integer.parseInt(str);

xor ^= nums[len];

len++;

}

int mask = xor - (xor & (xor - 1));

int val1 = 0;

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

if ((nums[i] & mask) != 0) val1 ^= nums[i];

}

int val2 = val1 ^ xor;

System.out.println(Math.min(val1, val2) + " " + Math.max(val1, val2));

}

}

```

## XM20 旋转数组中的最小元素

#### 题目描述

```

把一个数组最开始的若干个元素搬到数组的末尾，我们称之为数组的旋转。输入一个排好序的数组的一个旋转，输出旋转数组的最小元素。例如数组{3, 4, 5, 1, 2}为{1, 2, 3, 4, 5}的一个旋转，该数组的最小值为1。

```

#### 输入描述:

```

一个排好序的数组的一个旋转

数组长度不超过1000000

```

#### 输出描述:

```

该数组的最小值

```

#### 示例1

#### 输入

```

3 4 5 1 2

```

#### 输出

```

1

```

```java

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

/\*

找到第一个元素小于前面的元素，该值为最小

\*/

public class Main {

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

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

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

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

for (int i = 0; i < str.length; i++) arr[i] = Integer.parseInt(str[i]);

int result = arr[0];

for (int i = 0; i < arr.length - 1; i++) {

if (arr[i + 1] < arr[i]) {

result = arr[i + 1];

break;

}

}

System.out.println(result);

}

}

```

## XM21 求数列第n项

#### 题目描述

```

米兔从兔米那里了解到有一个无限长的数字序列 1, 2，3，3，4，4，4, 5，5，5，5，5 ...,(已知此数列有一定规律，现将这些数字按不同数值堆叠，相同值的数字在同一层)。米兔想知道这个数字序列的第n个数所在的那一层之前的所有层里共有多少个数。

```

#### 输入描述:

```

n(n<=1e18)

```

#### 输出描述:

```

第n个数所在的那一层之前的所有层里共有多少个数

```

#### 示例1

#### 输入

```

6

```

#### 输出

```

4

```

```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));

String s;

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

long n = Long.parseLong(s);

if (n == 0 || n == 1) {

System.out.println(0);

return;

}

if (n == 2) {

System.out.println(1);

return;

}

long a = 1;

long b = 1;

long sum = a + b;

while (n > sum) {

long temp = b;

b = a + b;

a = temp;

sum += b;

}

System.out.println(sum - b);

}

}

}

```

## XM22 计算题

#### 题目描述

```

给定n个非负整数表示每个宽度为1的柱子的高度题，计算按此排列的柱子，下雨之后能接多少雨水。

```

#### 输入描述:

```

逗号分隔的整数，表示每根柱子的高度。

柱子数n<=1000000,每根柱子的高度不大于100000

```

#### 输出描述:

```

雨水量（高度和）

```

#### 示例1

#### 输入

```

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

```

#### 输出

```

6

```

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

if (strs.length <= 2) {

System.out.println(0);

return;

}

int leftId = 0;

int rightId = strs.length - 1;

int leftMax = Integer.parseInt(strs[leftId]);

int rightMax = Integer.parseInt(strs[rightId]);

long water = 0;

while (leftId < rightId) {

if (leftMax < rightMax) {

leftId++;

int leftNum = Integer.parseInt(strs[leftId]);

if (leftNum < leftMax) water += leftMax - leftNum;

else leftMax = leftNum;

} else {

rightId--;

int rightNum = Integer.parseInt(strs[rightId]);

if (rightNum < rightMax) water += rightMax - rightNum;

else rightMax = rightNum;

}

}

System.out.println(water);

}

}

```

## XM23 设计一个函数1

#### 题目描述

```

设计一个函数，两个参数，第一个参数为整数的数组，第二个参数为标杆值，取数组中任意符合两个数相加为标杆值的下标相加到一起的值

传入一串字符串（如下例子所示），转义为数组，除去数组中最后一位数字作为标杆值，取数组中任意符合两个数相加为标杆值的下标，输出所有符合要求的下标的和。

如下解释：

value：0,1,5,11,17,16,2,5,10,30,12

index：1 3 6 8输出结果为18

```

#### 输入描述:

```

一串数字，逗号分割，最后一个值为标杆值

数组长度不超过1000，所有数均为不超过1e9的正整数。

```

#### 输出描述:

```

结果值

```

#### 示例1

#### 输入

```

0,1,5,11,17,16,2,5,10,30,12

```

#### 输出

```

18

```

#### 示例2

#### 输入

```

1,1,1,2

```

#### 输出

```

6

说明

(0+1)+(0+2)+(1+2)=6

```

```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));

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

int[] a = new int[str.length - 1];

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

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

}

int s = Integer.parseInt(str[str.length - 1]);

int num = 0;

for (int i = 0; i < a.length - 1; i++) {

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

if (a[i] + a[j] == s) num = num + i + j;

}

}

System.out.println(num);

}

}

```

## XM24 找异数

#### 题目描述

```

定义：数值序列中包含2~16进制整数，如果序列中有一个数，与序列中其他任何一个数大小都不相等，则这个数叫做“异数”。请找出给定数值序列中所有的“异数”.

```

#### 输入描述:

```

输入数值序列i行(0<i)，每一行分别是进制和数值，以“#”分割。如：n#m, n是整数，代表n进制（1<n<17）,m是n进制下的数值.

输入序列以结束符”END”结束。

m的字符集为0-9和A-F，保证数值在十进制下不超过1e9，行数不超过100001行。

```

#### 输出描述:

```

输出j行(0<j<=i),每一行都是输入序列的“异数”。要求：

1.按照输入序列的原序输出;

2.如果没有”异数”,输出字符串”None”

3.结束符“END”不用输出

```

#### 示例1

#### 输入

```

10#15

4#32

4#33

8#17

END

```

#### 输出

```

4#32

```

```java

import java.util.Scanner;

import java.util.ArrayList;

import java.util.HashMap;

public class Main {

public static void main(String[] args) {

Scanner input = new Scanner(System.in);

yList<String> list = new ArrayList<>();

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

while (true) {

String str = input.nextLine();

if (str.equals("END")) break;

else list.add(str);

}

int[] num = new int[list.size()];

for (int i = 0; i < list.size(); i++) {

String[] str = list.get(i).split("#");

num[i] = Integer.parseInt(str[1], Integer.parseInt(str[0]));

map.put(num[i], map.getOrDefault(num[i], 0) + 1);

}

int index = -1;

for (int i = 0; i < list.size(); i++) {

if (map.get(num[i]) == 1) {

index = 1;

System.out.println(list.get(i));

}

}

if (index == -1) System.out.println("None");

input.close();

}

}

```

## XM25 求整数的阶乘

#### 题目描述

```

求任一正整数的阶乘（注意：是任意正整数）

该正整数不大于1000。

```

#### 输入描述:

```

输入一个正整数

```

#### 输出描述:

```

输出一个正整数

```

#### 示例1

#### 输入

```

3

```

#### 输出

```

6

```

#### 示例2

#### 输入

```

10

```

#### 输出

```

3628800

```

```java

import java.io.\*;

import java.math.\*;

public class Main {

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

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

String s = br.readLine();

int a = Integer.parseInt(s);

if (a == 1) System.out.println(a);

else {

BigInteger res = BigInteger.valueOf(1);

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

BigInteger big = BigInteger.valueOf(i);

res = res.multiply(big);

}

System.out.println(res);

}

}

}

```

## XM26 最大新整数

#### 题目描述

```

有一十进制正整数，移除其中的 K 个数，使剩下的数字是所有可能中最大的。

假设：

字符串的长度一定大于等于 K

字符串不会以 0 开头

```

#### 输入描述:

```

一行由正整数组成的数字字符串，和一个正整数 K，两个数据用空格隔开，如：1432219 3。

字符串长度不超过2000，K<=2000。

```

#### 输出描述:

```

移除 K 位后可能的最大的数字字符串。

如 1432219 移除 1, 2, 1 这 3 个数字后得到 4329，为所有可能中的最大值。

```

#### 示例1

#### 输入

```

1432219 3

```

#### 输出

```

4329

```

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

int K = Integer.parseInt(strs[1]);

StringBuilder sb = new StringBuilder();

sb.ensureCapacity(strs[0].length() - K);

int i;

for (i = 0; i < strs[0].length(); i++) {

if (K == 0) break;

char ch = strs[0].charAt(i);

while (K > 0 && sb.length() > 0 && sb.charAt(sb.length() - 1) < ch) {

sb.deleteCharAt(sb.length() - 1);

K--;

}

sb.append(ch);

}

while (i < strs[0].length()) {

sb.append(strs[0].charAt(i));

i++;

}

System.out.println(sb.substring(0, sb.length() - K));

}

}

```

## XM27 计算原子的个数

#### 题目描述

```

给出一个字符串格式的化学分子式，计算原子的个数

每个化学元素都是由一个大写字母，或者一个大写字母后跟着若干个小写字母组成，例如H是一个化学元素，Mg也是一个化学元素。

每个分子式中，原子的个数写在元素后面，如果原子个数是1，那么原子个数省略。例如H2O和H2O2都是有效的分子式，但H1O2不是有效分子式。

每个分子式中包含若干括号，为简单起见，分子式中只有小括号。

每次输入一个分子式，对每个给定的分子式，求出每个原子的个数，按照原子字母表的顺序排列，并输出。

```

#### 输入描述:

```

一行，一个字符串表示输入的分子式

```

#### 输出描述:

```

按要求输出答案

```

#### 示例1

#### 输入

```

H2O

```

#### 输出

```

H2O

```

#### 示例2

#### 输入

```

Mg(OH)2

```

#### 输出

```

H2MgO2

```

#### 示例3

#### 输入

```

K4(ON(SO3)2)2

```

#### 输出

```

K4N2O14S4

```

```java

import java.util.\*;

public class Main {

private static int index = 0;

public static void main(String[] args) {

Scanner input = new Scanner(System.in);

String line = input.nextLine();

TreeMap<String, Integer> count = parse(line);

for (Map.Entry<String, Integer> entry : count.entrySet()) {

String vulue = entry.getValue() == 1 ? "" : String.valueOf(entry.getValue());

System.out.print(entry.getKey() + vulue);

}

}

private static TreeMap<String, Integer> parse(String line) {

TreeMap<String, Integer> count = new TreeMap<>();

int n = line.length();

while (index < n && line.charAt(index) != ')') {

if (line.charAt(index) == '(') {

index++;

for (Map.Entry<String, Integer> entry : parse(line).entrySet()) {

count.put(entry.getKey(), count.getOrDefault(entry.getKey(), 0) + entry.getValue());

}

} else {

int start = index++;

while (index < n && Character.isLowerCase(line.charAt(index))) {

index++;

}

String name = line.substring(start, index);

start = index;

while (index < n && Character.isDigit(line.charAt(index))) {

index++;

}

int num = start < index ? Integer.parseInt(line.substring(start, index)) : 1;

count.put(name, count.getOrDefault(name, 0) + num);

}

}

int start = ++index;

while (index < n && Character.isDigit(line.charAt(index))) {

index++;

}

int num = start < index ? Integer.parseInt(line.substring(start, index)) : 1;

if (num != 1) multi(count, num);

return count;

}

private static void multi(TreeMap<String, Integer> count, int num) {

count.replaceAll((k, v) -> v \* num);

}

}

```

## XM28 厨艺大赛奖金

#### 题目描述

```

小米食堂每年都会举办一次厨艺大赛，假设参赛的厨师一共有n位（n < 1000），比赛结束后没有公布评分，但是站在领奖台上的一排厨师中每位厨师都能看到与自己相邻的厨师（左或者右）里评分比自己低（看不到比自己分数高的人的分数）的评分。比赛结束之后要发奖金，以1K为单位，每位厨师至少会发1K的奖金，另外，如果一个厨师发现自己的奖金没有高于比自己评分低的厨师的奖金，就会不满意，作为比赛组织方，小米食堂至少需要发放多少奖金才能让所有厨师满意。

```

#### 输入描述:

```

每组数据为n+1个正整数单空格分割，其中第一个数为参赛厨师的人数，后面n个数为每位厨师的得分（0-100）

```

#### 输出描述:

```

输出至少需要多少K的奖金

```

#### 示例1

#### 输入

```

10 60 76 66 76 85 55 61 71 84 62

```

#### 输出

```

20

```

```java

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

public class Main {

/\*\*

\* 分糖果类型题

\* 首先从左往右遍历一遍数组，我分数比你高，那我必须比你多一千块,这样便利一遍就能确保相对于左边满足了条件

\* 在从右边往左边遍历一遍，我比你分数高，但是我的钱跟你一样甚至比你少，我必须要比你多1000块

\* 两边遍历后就分配好了

\*/

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

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

String[] line1 = bf.readLine().split(" ");

int n = Integer.parseInt(line1[0]);

int[] score = new int[n];

int[] money = new int[n];

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

score[i] = Integer.parseInt(line1[i + 1]);

//从左往右遍历，如果i比i-1的分数高，那我的钱一定要比你多

if (i > 0 && score[i] > score[i - 1]) money[i] = money[i - 1] + 1;

}

//在从右往左遍历

int sum = n;

for (int i = n - 2; i >= 0; i--) {

if (score[i] > score[i + 1] && money[i] <= money[i + 1]) money[i] = money[i + 1] + 1;

}

for (int i = 0; i < n; i++) sum += money[i];

System.out.println(sum);

}

}

```

## XM29 扑克牌四则运算

#### 题目描述

```

现在有一幅扑克牌，去掉大小王52张牌。随机选出4张牌，可以任意改变扑克牌的顺序，并填入 + - \* / 四个运算符，不用考虑括号，除法按整数操作，计算过程中没有浮点数，问是否能够求值得到给定的数m。

```

#### 输入描述:

```

一行四个数字 （JQK 分别被替换成11，12，13）单空格分割，另一行输入 m

```

#### 输出描述:

```

可以输出1

否则输出0

```

#### 示例1

#### 输入

```

13 13 13 1324

```

#### 输出

```

0

备注:

提示：

注意运算符优先级

```

```java

import java.util.ArrayList;

import java.util.List;

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int[] cards = new int[4];

List<Integer> nums = new ArrayList<>();

for (int i = 0; i < 4; i++) {

nums.add(scanner.nextInt());

}

int m = scanner.nextInt();

boolean flag = solution(nums, m);

System.out.println(flag ? 1 : 0);

}

private static boolean solution(List<Integer> nums, int m) {

if (nums.size() == 0) return false;

if (nums.size() == 1) return nums.get(0) == m;

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

for (int j = 0; j < nums.size(); j++) {

if (i != j) {

List<Integer> nums2 = new ArrayList<>();

for (int k = 0; k < nums.size(); k++) {

if (k != i && k != j) nums2.add(nums.get(k));

}

for (int k = 0; k < 4; k++) {

if (k < 2 && j > i) continue;

if (k == 0) nums2.add(nums.get(i) + nums.get(j));

if (k == 1) nums2.add(nums.get(i) \* nums.get(j));

if (k == 2) nums2.add(nums.get(i) - nums.get(j));

if (k == 3) {

if (nums.get(j) != 0) nums2.add(nums.get(i) / nums.get(j));

else continue;

}

if (solution(nums2, m)) return true;

nums2.remove(nums2.size() - 1);

}

}

}

}

return false;

}

}

```

## XM30 小米大礼包

#### 题目描述

```

小米之家是成人糖果店。里面有很多便宜，好用，好玩的产品。中秋节快到了，小米之家想给米粉们准备一些固定金额大礼包。对于给定的一个金额，需要判断能不能用不同种产品（一种产品在礼包最多出现一次）组合出来这个金额。聪明的你来帮帮米家的小伙伴吧。

```

#### 输入描述:

```

输入 N （N 是正整数， N <= 200）

输入 N 个价格p（正整数, p <= 10000）用单空格分割

输入金额 M（M是正整数，M <= 100000 ）

```

#### 输出描述:

```

能组合出来输出 1

否则输出 0

```

#### 示例1

#### 输入

```

6

99 199 1999 10000 39 1499

10238

```

#### 输出

```

1

```

```java

import java.io.\*;

import java.util.\*;

public class Main {

private static boolean can(int[] a, int t) {

long[] dp = new long[t + 1];

dp[0] = 1;

for (int k : a) {

for (int j = t; j >= k; j--) dp[j] += dp[j - k];

if (dp[t] > 0) return true;

}

return false;

}

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

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

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

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

int[] a = new int[n];

for (int i = 0; i < n; i++) a[i] = Integer.parseInt(ss[i]);

int m = Integer.parseInt(bf.readLine());

if (can(a, m)) System.out.println(1);

else System.out.println(0);

}

}

```

## XM31 最优分割

#### 题目描述

```

依次给出n个正整数A1，A2，… ，An，将这n个数分割成m段，每一段内的所有数的和记为这一段的权重， m段权重的最大值记为本次分割的权重。问所有分割方案中分割权重的最小值是多少？

```

#### 输入描述:

```

第一行依次给出正整数n，m，单空格切分；(n <= 10000, m <= 10000, m <= n)

第二行依次给出n个正整数单空格切分A1，A2，… ，An (Ai <= 10000)

```

#### 输出描述:

```

分割权重的最小值

```

#### 示例1

#### 输入

```

5 3

1 4 2 3 5

```

#### 输出

```

5

说明

分割成 1 4 | 2 3 | 5 的时候，3段的权重都是5，得到分割权重的最小值。

```

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

String[] line1 = bf.readLine().split(" ");

int n = Integer.parseInt(line1[0]);

int m = Integer.parseInt(line1[1]);

String[] line2 = bf.readLine().split(" ");

int[] nums = new int[n];

int sum = 0;

int max = 0;

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

nums[i] = Integer.parseInt(line2[i]);

if (max < nums[i]) max = nums[i];

sum += nums[i];

}

System.out.println(binarySearch(nums, m, n, max, sum));

}

private static int binarySearch(int[] nums, int m, int n, int left, int right) {

int ans = right;

while (left <= right) {

int mid = (left + right) / 2;

int sum = 0;

int count = 1;//记录数组的个数

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

//直到当前子数组的和加上当前元素比mid还大，那必须将当前元素归为下一个子数组中，sum重新计算新子数组的和

if (sum + nums[i] > mid) {

count++;

sum = nums[i];

} else sum += nums[i]; //当前子数组的和比mid小，继续加

}

//如果分完之后组数小于等于m说明，mid还可以更小，即上面思路里说的x还能更小 右区间缩小到mid-1；

if (count <= m) {

ans = Math.min(ans, mid);

right = mid - 1;

} else left = mid + 1;

}

return ans;

}

}

```

## XM32 如何添加运算符

#### 题目描述

```

给出一个数字N，对于数字序列 1,2,3 ... N。现在在其中插入“+”, "-", " "，使得表达式的和为M。" "的含义是把相邻的两个数字组成一个数。例如：1 + 2 3 - 4，含义是：1 + 23 - 4 = 20。

给出N和M，求出所有合法的序列的个数。

```

#### 输入描述:

```

两个整数N,M ( 1 <= N <= 7, -100 <= M <= 100)

```

#### 输出描述:

```

合法序列的个数

```

#### 示例1

#### 输入

```

7 0

```

#### 输出

```

6

说明

样例中的六种合法序列

1+2-3+4-5-6+7

1+2-3-4+5+6-7

1-2 3+4+5+6+7

1-2 3-4 5+6 7

1-2+3+4-5+6-7

1-2-3-4-5+6+7

```

```java

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.io.IOException;

public class Main {

public static int count = 0;

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

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

String str;

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

String[] strNM = str.split(" ");

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

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

dfs(0, 1, n, m);

System.out.println(count);

}

}

public static void dfs(int sum, int start, int n, int target) {

if (sum == target && start == n + 1) count++;

int temp = 0;

for (int i = start; i <= n; i++, temp \*= 10) {

temp += i;

dfs(sum + temp, i + 1, n, target);

if (start != 1) dfs(sum - temp, i + 1, n, target);

}

}

}

```

## XM33 小明的字符串

#### 题目描述

```

小明同学需要对一个长度为 N 的字符串进行处理，他需要按照要求执行若干步骤，每个步骤都均为下面 2 种操作中的一种，2 种操作如下：

TYPE 1. 从字符串结尾开始算起，将第 X 个字符之前的字符移动到字符串末尾

TYPE 2. 输出字符串索引为 X 的字符

小明尝试了很久没能完成，你可以帮他解决这个问题吗？

```

#### 输入描述:

```

第一行，包含两个整数，字符串的长度 N 和操作次数T；

第二行为要操作的原始字符串；

之后每行都是要执行的操作类型 TYPE 和操作中 X 的值，均为整数。

输入范围：

字符串长度 N：1 <= N <= 10000

操作次数 T：1 <= T <= 10000

操作类型 TYPE：1 <= TYPE<= 2

变量 X：0 <= X < N

```

#### 输出描述:

```

操作的执行结果

```

#### 示例1

#### 输入

```

6 2

xiaomi

1 2

2 0

```

#### 输出

```

m

```

```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));

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

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

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

String string = br.readLine();

int offset = 0;

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

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

int type = Integer.parseInt(op[0]);

int index = Integer.parseInt(op[1]);

if (type == 1) offset = offset + n - index;

else if (type == 2) System.out.println(string.charAt((offset + index) % n));

}

}

}

```

## XM34 分布式集群消息传递

#### 题目描述

```

有一个分布式服务集群，集群内含有 N 个服务节点，分别标记为 1 到 N。

给予一个列表 times，表示消息从两个节点间有向传递需要的时间。 times[i] = (s, d, t)，其中 s 表示发出消息的源节点，d 表示接收到消息的目标节点， t 表示信息有向传递的时间。

现在 K 节点发送了一个信号，请问至少需要多少秒才能使所有的服务节点都收到该消息？如果消息不能传递给集群内全部节点，则返回-1。

```

#### 输入描述:

```

第一行：列表 times。分布式服务集群的图，图的结构为二维数组。例如： [[2,1,1],[2,3,1],[3,4,1]] ，表示集群4个节点，2到1的时间为1，2到3的时间为1，3到4的时间为1；

第二行：N值

第三行：K值

范围约束：

1. N 的范围在 [1, 100] 之间。

2. K 的范围在 [1, N] 之间。

3. times 的长度在 [1, 6000] 之间。

4. 所有的边 times[i] = (s, d, t) 都有 1 <= s, d <= N 且 1 <= t <= 100。

```

#### 输出描述:

```

至少需要多少秒才能使所有的服务节点都收到该消息？如果消息不能传递给集群内全部节点，则返回-1

```

#### 示例1

#### 输入

```

[[2,1,1],[2,3,1],[3,4,1]]

4

2

```

#### 输出

```

2

备注:

图可能存在重边或自环

```

```java

import java.util.\*;

import java.lang.Math;

public class Main {

public static void main(String[] args) {

Scanner in = new Scanner(System.in);

while (in.hasNext()) {

String rawStr = in.nextLine();

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

int K = Integer.parseInt(in.nextLine());

int[][] times = new int[N + 1][N + 1];

for (int[] time : times) Arrays.fill(time, 10001);

List<Integer> Sk = new ArrayList<>();

List<Integer> Uk = new ArrayList<>();

List<Integer> Uv = new ArrayList<>();

StringTokenizer stn = new StringTokenizer(rawStr, "[],");

while (stn.hasMoreTokens()) {

int x = Integer.parseInt(stn.nextToken()), y = Integer.parseInt(stn.nextToken()), z = Integer.parseInt(stn.nextToken());

times[x][y] = Math.min(times[x][y], z);

}

Sk.add(K);

for (int i = 1; i <= N; i++)

if (i != K) {

Uk.add(i);

Uv.add(times[K][i]);

}

long ret = -1;

while (Uk.size() > 0) {

int mindis = Collections.min(Uv);

if (mindis > 10000) break;

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

ret = mindis;

break;

}

int idx = Uv.indexOf(mindis);

int k = Uk.get(idx);

Uk.remove(idx);

Uv.remove(idx);

Sk.add(k);

for (int i = 0; i < Uk.size(); i++) {

Uv.set(i, Math.min(Uv.get(i), mindis + times[k][Uk.get(i)]));

}

}

System.out.println(ret);

}

}

}

```

## XM35 CCNumber

#### 题目描述

```

CC最近对一种整数比较感兴趣，我们暂且把这种整数称为C Number, C Number是指一个整数 {C0, C1 … Cn-1} (C0 > 0 , n >= 3)， 存在一个Cm（0<m<n-1）满足以下条件:

Ci-1 < Ci (0<i<=m), Ci代表这个整数中的第i位数字

Ci>Ci+1(m<=i<n-1)

如果一个整数里面有相邻的2个C Number的话，我们称这个整数为CC Number（2个C Number不可以有公用的数字Ci，并且2个C Number要紧紧相邻）。

请在[A,B]区间内找出找出score最大的CCNumber 并输出这个score.(score：CC Number中所有数字的和)

```

#### 输入描述:

```

第一行为数字N(N<=1000),后面有N行测试用例

每行用例有2个数字 A,B（0<=A<=B<2^64）,需要[A,B]区间内找出题干中描述的最大score。

```

#### 输出描述:

```

对于第N行的测试用例，输出“Case N: S”, S为最大的score，如果区间内没有CC Number的话 S为0。

```

#### 示例1

#### 输入

```

4

12121 12121

120010 120010

121121 121121

1211121 1211121

```

#### 输出

```

Case 1: 0

Case 2: 0

Case 3: 8

Case 4: 0

略，没有Java版

```

## XM36 获取n维数组的最大深度

#### 题目描述

```

输入参数为字符串型的n维数组，数组的每一项值为数组 或 int型数字。请实现一个函数，可以获取列表嵌套列表的最大深度为多少。

```

#### 输入描述:

```

输入参数为字符串型的 n维数组，列表的每一项值为数组 或 int型数字。数组内的数组，每一项值，也可以是数组 或 int型数字。

```

#### 输出描述:

```

int型数字，表示数组嵌套的深度。

```

#### 示例1

#### 输入

```

[[1], [2,3,4], [5,[2,3]], [7], [0,[1,2,3,4],3,5], [1,3], [3,2,4]]

```

#### 输出

```

3

说明

n维数组的深度为3

```

```java

import java.util.\*;

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();

int count = 0, res = 0;

for (char c : s.toCharArray()) {

if (c == '[') {

count++;

res = Math.max(count, res);

} else if (c == ']') count--;

}

System.out.println(res);

}

}

```

## XM37 爬楼梯2

#### 题目描述

```

在你面前有一个n阶的楼梯(n>=100且n<500)，你一步只能上1阶或3阶。

请问计算出你可以采用多少种不同的方式爬完这个楼梯（到最后一层为爬完）。

(注意超大数据)

```

#### 输入描述:

```

一个正整数，表示这个楼梯一共有多少阶

```

#### 输出描述:

```

一个正整数，表示有多少种不同的方式爬完这个楼梯

```

#### 示例1

#### 输入

```

100

```

#### 输出

```

24382819596721629

备注:

注意时间限制

```

```java

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.math.BigInteger;

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());

BigInteger[] count = new BigInteger[n];

count[0] = new BigInteger("1");

count[1] = new BigInteger("1");

count[2] = new BigInteger("2");

for (int i = 3; i < n; i++) {

count[i] = count[i - 1].add(count[i - 3]);

}

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

}

}

```

## XM38 设计一个函数2

#### 题目描述

```

设计一个函数，传入一个可序列化为树结构的字符串，将含有多个子节点的节点以数组的形式输出。

```

#### 输入描述:

```

{ node: 'root', next: [ { node: 'second\_root' }, { node: 'second\_child', next: [{ node: 'second\_child\_1', next: { node: 'second\_child\_1\_1' } }, { node: 'second\_child\_2' }] }, { node: 'third\_root', next: { node: 'third\_child' , next: [{ node: 'third\_child\_1', next: { node: 'third\_child\_1\_1' } }, { node: 'third\_child\_2' }] } } ] }

```

#### 输出描述:

```

数组

输出规范

1）数组应被左右中括号括起；

2）数组的元素间由','相隔；

3）各节点在数组中的顺序应和其在输入中出现的次序一致；

4）节点名保证为不超过30个字符的字符串，仅含大小写字母、数字及下划线，输出时应用双引号括起；

5）输出的字符串不应有多余的空格。

```

#### 示例1

#### 输入

```

{ node: 'root', next: [ { node: 'second\_root' }, { node: 'second\_child', next: [{ node: 'second\_child\_1', next: { node: 'second\_child\_1\_1' } }, { node: 'second\_child\_2' }] }, { node: 'third\_root', next: { node: 'third\_child' , next: [{ node: 'third\_child\_1', next: { node: 'third\_child\_1\_1' } }, { node: 'third\_child\_2' }] } } ] }

```

#### 输出

```

["root","second\_child","third\_child"]

```

```java

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.ArrayList;

public class Main {

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

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

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

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

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

if (sp[i].contains("[")) {

String s = sp[i - 1];

String[] t = s.split(" ");

String st = t[t.length - 1].substring(1, t[t.length - 1].length() - 1);

al.add("\"" + st + "\"");

}

}

System.out.print("[");

for (int i = 0; i < al.size(); i++) {

System.out.print(al.get(i));

if (i != al.size() - 1) System.out.print(",");

}

System.out.print("]");

}

}

```

## XM39 集合合并

#### 题目描述

```

给定若干个32位int数字集合，每个集合中的数字无重复，譬如：

{1,2,3} {2,5,6} {8}

将其中交集不为空的集合合并，保证合并完成后所有集合之间无交集，输出合并后的集合个数以及最大集合中元素的个数。

```

#### 输入描述:

```

输入格式：

1. 第一行为一个数字N，表示集合数。

2. 接下来N行，每行一个非空集合，包含若干个数字，数字之间用空格分开。

假设第i个集合的大小为Si，数据满足N<=100000,ΣSi<=500000

```

#### 输出描述:

```

输出格式：

1. 第一行为合并后的集合个数。

2. 第二个为最大集合中元素的个数。

```

#### 示例1

#### 输入

```

3

1 2 3

2 5 6

8

```

#### 输出

```

2

5

```

```java

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.HashMap;

import java.util.HashSet;

public class Main {

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

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

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

int total = 20000000;

int[] b = new int[n + 1];

int[] c = new int[total];

int k = n;

int max = 0;

int[] a = new int[n + 1];

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

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

for (String s : sp) {

int pa = Integer.parseInt(s);

if (c[pa] == 0) {

c[pa] = i;

++a[i];

max = Math.max(max, a[i]);

} else {

int t = c[pa];

c[pa] = i;

while (b[t] != 0) {

if (t == b[t]) break;

int tt = t;

t = b[t];

b[tt] = i;

}

if (t != i) {

a[i] += a[t];

max = Math.max(max, a[i]);

--k;

b[t] = i;

}

}

}

}

System.out.println(k);

System.out.println(max);

}

}

```

## XM40 升级蓄水池

#### 题目描述

```

在米兔生活的二维世界中，建造蓄水池非常简单。

一个蓄水池可以用n个坐标轴上的非负整数表示，代表区间为【0-n】范围内宽度为1的墙壁的高度。

如下图1，黑色部分是墙壁，墙壁的高度是[0,1,0,2,1,0,1,3,2,1,2,1] ，蓝色部分是蓄水的面积，可以看出蓄水池最大蓄水容量是6。

现在米兔想通过增加某些墙壁的高度对蓄水池扩容，但是经费有限，最多只能增加最多m的高度，增加高度只能在【0-n】范围内，高度为0的区域也是可以增加的，为了追求最大的性价比，米兔想要找到一种最优方案，使扩容后蓄水池的容量最大，你能帮帮他么？

提示：

对于样例，图2，图3，是样例可能的两种扩容方案，显然图2是比图3更优的方案

```

#### 输入描述:

```

第一行为一个数字n

接下来n行，每行一个数字，代表n个墙壁的高度

最后一行为一个数字m

```

#### 输出描述:

```

一个数字，表示扩容之后蓄水池能达到的最大容量

```

#### 示例1

#### 输入

```

12

0

1

0

2

1

0

1

3

2

1

2

1

2

```

#### 输出

```

12

备注:

关于题目数据范围

20%的数据， n<10,m<10,蓄水池最大高度小于10

50%的数据， n<100,m<100,蓄水池最大高度小于100

100%的数据， n<1000,m<1000,蓄水池最大高度小于1000

```

```java

import java.util.\*;

public class Main {

static int n = 0;

static int m = 0;

static int[] arr;

static int maxHeight = 0;

static int leftIndex = 0;

static int rightIndex = 0;

static int[] maxLeft;

static int[] maxRight;

public static void main(String[] args) {

Scanner input = new Scanner(System.in);

n = input.nextInt();

arr = new int[n];

maxHeight = 0;

leftIndex = 0;

rightIndex = 0;

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

arr[i] = input.nextInt();

if (maxHeight < arr[i]) {

maxHeight = arr[i];

leftIndex = i;

}

if (maxHeight == arr[i]) rightIndex = i;

}

maxLeft = new int[n];

maxRight = new int[n];

int max = 0;

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

maxLeft[i] = max;

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

}

max = 0;

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

maxRight[i] = max;

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

}

m = input.nextInt();

System.out.println(countArea(arr) + solve());

}

public static int solve() {

int maxArea = 0;

int totArea = 0;

//前

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

int tmp = Math.min(maxHeight - arr[i], m);

totArea = 0;

totArea += add(arr, i, tmp);

if (tmp != m) {

arr[i] += tmp;

int temp = leftIndex;

leftIndex = i;

totArea += Math.max(count(arr, 2, m - tmp), count(arr, 1, m - tmp)); //后或后中

leftIndex = temp;

arr[i] -= tmp;

}

maxArea = Math.max(maxArea, totArea);

}

//后

for (int i = n - 1; i > rightIndex; i--) {

int tmp = Math.min(maxHeight - arr[i], m);

totArea = 0;

totArea += add(arr, i, tmp);

if (tmp != m) {

arr[i] += tmp;

int temp = rightIndex;

rightIndex = i;

totArea += Math.max(count(arr, 2, m - tmp), count(arr, 0, m - tmp)); //前或前中

rightIndex = temp;

arr[i] -= tmp;

}

maxArea = Math.max(maxArea, totArea);

}

//中

if (leftIndex != rightIndex) maxArea = Math.max(maxArea, count(arr, 2, m)); //中

return maxArea;

}

public static int count(int[] arr, int type, int tempM) {

if (tempM == 0) return 0;

int maxArea = 0;

int totArea = 0;

if (type <= 1) { //前后

int st = type == 1 ? rightIndex + 1 : 0;

int en = type == 1 ? n : leftIndex;

for (int i = st; i < en; i++) {

totArea = 0;

int tmp = Math.min(maxHeight - arr[i], tempM);

totArea += add(arr, i, tmp);

if (tmp != tempM) {

arr[i] += tmp; //这里可剪枝

int temp = 0;

if (type == 0) {

temp = leftIndex;

leftIndex = i;

} else {

temp = rightIndex;

rightIndex = i;

}

totArea += count(arr, 2, tempM - tmp);

if (type == 0) leftIndex = temp;

else rightIndex = temp;

arr[i] -= tmp;

}

maxArea = Math.max(maxArea, totArea);

}

} else { //中

int tmp1 = tempM / 2;

if (tmp1 == 0 && leftIndex == 0 && rightIndex == n - 1) return 0;

totArea = add(arr, leftIndex, tmp1);

arr[leftIndex] += tmp1;

arr[rightIndex] += tmp1;

int has = tempM - 2 \* tmp1;

totArea += Math.max(count(arr, 1, has), count(arr, 0, has));

arr[rightIndex] -= tmp1;

arr[leftIndex] -= tmp1;

maxArea = Math.max(maxArea, totArea);

}

return maxArea;

}

public static int add(int[] arr, int index, int change) {

int max = arr[index];

int tot = 0;

int sum = arr[index] + change;

if (index < leftIndex) { //有误

max = Math.max(max, maxLeft[index]);

for (int i = index + 1; i < leftIndex && arr[i] <= sum; i++) {

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

tot += sum > max ? (sum - max) : 0;

}

} else if (index > rightIndex) {

max = Math.max(max, maxRight[index]);

for (int i = index - 1; i > rightIndex && arr[i] <= sum; i--) {

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

tot += sum > max ? (sum - max) : 0;

}

} else tot = (rightIndex - leftIndex - 1) \* change;

return tot;

}

public static int countArea(int[] arr) {

int area = 0;

int i = 0, j = arr.length - 1;

int leftMax = arr[i], rightMax = arr[j];

while (i < j) {

if (leftMax > rightMax) {

area += rightMax - arr[j];

j--;

rightMax = Math.max(rightMax, arr[j]);

} else {

area += leftMax - arr[i];

i++;

leftMax = Math.max(leftMax, arr[i]);

}

}

return area;

}

}

```