```

题号 题目 知识点 难度 通过率

KS1 获得最多的奖金 数组贪心 中等 21.23%

KS2 将满二叉树转换为求和树 递归模拟 简单 25.24%

KS3 搭积木 排序动态规划 较难 16.35%

KS4 最少数量货物装箱问题 动态规划贪心 中等 31.94%

KS5 回文子串 字符串模拟动态规划 较难 36.79%

KS6 字符串长度最大乘积 字符串模拟 简单 29.19%

KS7 今年的第几天 模拟 简单 38.80%

KS8 数字序列第n位的值 简单 43.40%

KS9 字符串排序 排序字符串 入门 31.93%

KS10 回文字符串 字符串动态规划 中等 26.79%

KS11 latex爱好者 中等 16.81%

KS12 游戏海报 字符串模拟 入门 45.05%

KS13 合并数组 数组模拟 中等 18.09%

KS14 字符串包含 字符串模拟 中等 26.04%

KS15 魔法深渊 动态规划 中等 25.14%

KS16 善变的同伴 动态规划 中等 8.89%

KS17 字符串归一化 模拟字符串 入门 35.63%

KS18 a/b 模拟 中等 28.34%

KS19 最小代价爬楼梯 动态规划 中等 25.06%

KS20 字符串压缩 字符串模拟 简单 32.19%

KS21 解析加减法运算 字符串数组模拟 简单 32.59%

KS22 求连续子数组的最大和 动态规划数组贪心 中等 32.23%

KS23 非递减序列 排序数组穷举 中等 44.40%

KS24 求x到y的最少计算次数 队列 中等 27.13%

KS25 阶乘末尾非零数字 数学 中等 13.63%

KS26 字符串最小变换次数 动态规划字符串 中等 36.35%

KS27 二进制中有多少个1 位运算 简单 44.04%

KS28 计算斐波那契数最小差值 穷举 中等 36.68%

KS29 查找无重复最长子串 字符串哈希 中等 30.23%

KS30 情报 递归图动态规划 中等 25.95%

KS31 最大公共子串 字符串动态规划 较难 33.83%

KS32 找缺失数字 穷举字符串 中等 33.49%

KS33 寻找奇数 穷举 简单 22.80%

KS34 计算器 字符串模拟栈 中等 38.67%

KS35 机器人移动范围 图数组 中等 18.27%

KS36 判断一棵满二叉树是否为二叉搜索树 递归树 简单 13.65%

```

### KS1 获得最多的奖金

#### 题目描述

```

小明在越南旅游，参加了当地的娱乐活动。小明运气很好，拿到了大奖， 到了最后的拿奖金环节。小明发现桌子上放着一列红包，每个红包上写着奖金数额。

现在主持人给要求小明在这一列红包之间“切”2刀，将这一列红包“切”成3组，并且第一组的奖金之和等于最后一组奖金和（允许任意一组的红包集合是空）。最终第一组红包的奖金之和就是小明能拿到的总奖金。小明想知道最多能拿到的奖金是多少，你能帮他算算吗。

举例解释：桌子上放了红包 1, 2, 3, 4, 7, 10。小明在“4,7”之间、“7,10” 之间各切一刀，将红包分成3组 [1, 2, 3, 4] [7] [10]，其中第一组奖金之和=第三组奖金之和=10，所以小明可以拿到10越南盾。

```

#### 输入描述:

```

第一行包含一个正整数n，(1<=n<= 200 000)，表示有多少个红包。

第二行包含n个正整数d[i]，表示每个红包包含的奖金数额。其中1<= d[i] <= 1000 000 000

```

#### 输出描述:

```

小明可以拿到的总奖金

```

#### 示例1

#### 输入

```

5

1 3 1 1 4

```

#### 输出

```

5

```

#### 说明

```

[1,3,1] [ ] [1,4] ，其中第一组奖金和是5，等于第三组奖金和。所以小明可以拿到5越南盾

```

#### 示例2

#### 输入

```

5

1 3 2 1 4

```

#### 输出

```

4

```

#### 说明

```

[1,3] [2,1] [4]，小明可以拿到4越南盾

####示例3

```

#### 输入

```

3

4 1 2

```

#### 输出

```

0

```

#### 说明

```

[ ] [4, 1, 2] [ ] ，小明没办法，为了保证第一组第三组相等，只能都分成空的。所以小明只能拿到0越南盾。

```

```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));

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

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

int[] nums = new int[n];

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

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

}

int left = 0, right = n - 1;

long left\_sum = nums[left], right\_sum = nums[right], max\_sum = 0;

while (left < right) {

if (left\_sum == right\_sum) {

max\_sum = left\_sum;

left\_sum += nums[++left];

right\_sum += nums[--right];

} else if (left\_sum > right\_sum) {

right\_sum += nums[--right];

} else {

left\_sum += nums[++left];

}

}

System.out.println(max\_sum);

}

}

```

### KS2 将满二叉树转换为求和树

#### 题目描述

```

给出满二叉树，编写算法将其转化为求和树

什么是求和树：二叉树的求和树， 是一颗同样结构的二叉树，其树中的每个节点将包含原始树中的左子树和右子树的和。

二叉树：

10

/ \

-2 6

/ \ / \

8 -4 7 5

求和树：

20(4-2+12+6)

/ \

4(8-4) 12(7+5)

/ \ / \

0 0 0 0

二叉树给出前序和中序输入，求和树要求中序输出；

所有处理数据不会大于int；

```

#### 输入描述:

```

2行整数，第1行表示二叉树的前序遍历，第2行表示二叉树的中序遍历，以空格分割

```

#### 输出描述:

```

1行整数，表示求和树的中序遍历，以空格分割

```

#### 示例1

#### 输入

```

10 -2 8 -4 6 7 5

8 -2 -4 10 7 6 5

```

#### 输出

```

0 4 0 20 0 12 0

```

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

String s1 = bufferedReader.readLine();

String s2 = bufferedReader.readLine();

String[] preOrder = s1.split(" ");

String[] inOrder = s2.split(" ");

int[] res = new int[inOrder.length];

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

res[i] = Integer.parseInt(inOrder[i]);

}

dfs(res, 0, inOrder.length - 1);

for (int r : res) System.out.print(r + " ");

}

private static int dfs(int[] res, int left, int right) {

if (left == right) {

int temp = res[left];

res[left] = 0;

return temp;

}

int mid = (right - left) / 2 + left;

int leftSum = dfs(res, left, mid - 1);

int rightSum = dfs(res, mid + 1, right);

int temp = res[mid];

res[mid] = leftSum + rightSum;

return res[mid] + temp;

}

}

```

### KS3 搭积木

#### 题目描述

```

小明有一袋子长方形的积木，如果一个积木A的长和宽都不大于另外一个积木B的长和宽，则积木A可以搭在积木B的上面。好奇的小明特别想知道这一袋子积木最多可以搭多少层，你能帮他想想办法吗？

定义每一个长方形的长L和宽W都为正整数，并且1 <= W <= L <= INT\_MAX, 袋子里面长方形的个数为N, 并且 1 <= N <= 1000000.

假如袋子里共有5个积木分别为 (2, 2), (2, 4), (3, 3), (2, 5), (4, 5), 则不难判断这些积木最多可以搭成4层, 因为(2, 2) < (2, 4) < (2, 5) < (4, 5)。

```

#### 输入描述:

```

第一行为积木的总个数 N

之后一共有N行，分别对应于每一个积木的宽W和长L

```

#### 输出描述:

```

输出总共可以搭的层数

```

#### 示例1

#### 输入

```

5

2 2

2 4

3 3

2 5

4 5

```

#### 输出

```

4

```

```java

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.\*;

public class Main {

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

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

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

int[][] arr = new int[n][2];

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

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

arr[i][0] = Integer.parseInt(s[0]);

arr[i][1] = Integer.parseInt(s[1]);

}

bf.close();

if (n == 1) {

System.out.println(1);

return;

}

Arrays.sort(arr, Comparator.comparingInt(o -> o[0]));

int[] dp = new int[n];

dp[0] = 1;

int count = 0;

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

int val = arr[i][1];

int l = 0;

int r = count;

while (l < r) {

int mid = l + (r - l) / 2;

if (dp[mid] > val)

r = mid;

else

l = mid + 1;

}

if (l == count)

count++;

dp[l] = val;

}

System.out.println(count);

}

}

```

### KS4 最少数量货物装箱问题

#### 题目描述

```

有重量分别为3，5，7公斤的三种货物，和一个载重量为X公斤的箱子（不考虑体积等其它因素，只计算重量）

需要向箱子内装满X公斤的货物，要求使用的货物个数尽可能少（三种货物数量无限）

```

#### 输入描述:

```

输入箱子载重量X(1 <= X <= 10000)，一个整数。

```

#### 输出描述:

```

如果无法装满，输出 -1。

如果可以装满，输出使用货物的总个数。

```

#### 示例1

#### 输入

```

4

```

#### 输出

```

-1

```

#### 说明

```

无法装满

```

#### 示例2

#### 输入

```

8

```

#### 输出

```

2

```

#### 说明

```

使用1个5公斤，1个3公斤货物

```

```java

import java.io.\*;

public class Main {

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

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

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

int[] w = new int[]{3, 5, 7};

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

for (int k : w) {

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

if (j % k == 0) {

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

} else if (dp[j - k] != 0) {

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

}

}

}

System.out.println(dp[X] == 0 ? -1 : dp[X]);

}

}

```

### KS5 回文子串

#### 题目描述

```

给定一个字符串，你的任务是计算这个字符串中有多少个回文子串。

("回文串”是一个正读和反读都一样的字符串，比如“level”或者“noon”等等就是回文串。)

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，也会被计为是不同的子串。

可用C++,Java,C#实现相关代码逻辑

```

#### 输入描述:

```

输入一个字符串S 例如“aabcb”(1 <= |S| <= 50), |S|表示字符串S的长度。

```

#### 输出描述:

```

符合条件的字符串有"a","a","aa","b","c","b","bcb"

所以答案:7

```

#### 示例1

#### 输入

```

aabcb

```

#### 输出

```

7

```

```java

import java.io.\*;

public class Main {

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

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

String s = bf.readLine();

int count = 0, len = s.length();

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

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

if (same(s.substring(i, j))) {

count++;

}

}

}

System.out.print(count);

}

// 判断是否为回文串

private static boolean same(String s) {

char[] chars = s.toCharArray();

int left = 0, right = chars.length - 1;

while (left < right) {

if (chars[left++] != chars[right--]) {

return false;

}

}

return true;

}

}

```

### KS6 字符串长度最大乘积

#### 题目描述

```

已知一个字符串数组words，要求寻找其中两个没有重复字符的字符串，使得这两个字符串的长度乘积最大，输出这个最大的乘积。如：

words=["abcd","wxyh","defgh"], 其中不包含重复字符的两个字符串是"abcd"和"wxyh"，则输出16

words=["a","aa","aaa","aaaa"], 找不到满足要求的两个字符串，则输出0

```

#### 输入描述:

```

Input:

["a","ab","abc","cd","bcd","abcd"]

```

#### 输出描述:

```

Output:

4

```

#### 示例1

#### 输入

```

["a","ab","abc","cd","bcd","abcd"]

```

#### 输出

```

4

```

#### 备注:

```

Input中，不包含相同字符的有三对：

"ab"和"cd"

"a"和"cd"

"a"和"bcd"

所以字符串长度乘积的最大值是4

```

```java

import java.io.\*;

public class Main {

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

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

String inputStr = reder.readLine();

String[] strs = inputStr.substring(1, inputStr.length() - 1).split(",");

int max = 0;

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

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

max = Math.max(compute(strs[i], strs[j]), max);

}

}

System.out.println(max);

}

public static int compute(String strs1, String strs2) {

for (int i = 1; i < strs1.length() - 1; i++) {

for (int j = 1; j < strs2.length() - 1; j++) {

if (strs1.charAt(i) == strs2.charAt(j)) {

return 0;

}

}

}

return (strs1.length() - 2) \* (strs2.length() - 2);

}

}

```

### KS7 今年的第几天

#### 题目描述

```

输入年、月、日，计算该天是本年的第几天。

输入：

包括三个整数年(1<=Y<=3000)、月(1<=M<=12)、日(1<=D<=31)。

输出：

输入可能有多组测试数据，对于每一组测试数据，

输出一个整数，代表Input中的年、月、日对应本年的第几天。

```

#### 输入描述:

```

输入：1990 9 20

```

#### 输出描述:

```

输入：263

```

#### 示例1

#### 输入

```

2000 5 1

```

#### 输出

```

122

####备注:

注意闰年的判定方式

```

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

int year = Integer.parseInt(a[0]);

int month = Integer.parseInt(a[1]);

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

int[] md = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

if ((year % 4 == 0 && year % 100 != 0) || year % 400 == 0) {

md[1]++;

}

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

day += md[i];

}

System.out.println(day);

}

}

```

### KS8 数字序列第n位的值

#### 题目描述

```

有一个无限长的数字序列1，2，2，3，3，3，4，4，4，4，5，5，5，5，5。。。（数字序列从1开始递增，且数字k在该序列中正好出现k次），求第n项是多少

```

#### 输入描述:

```

输入为一个整数n

```

#### 输出描述:

```

输出一个整数，即第n项的值

```

#### 示例1

#### 输入

```

4

```

#### 输出

```

3

```

#### 备注:

```

如：输入为3，有序数列第3项的值为2，则输出为2

```

```java

import java.io.\*;

import java.util.\*;

public class Main {

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

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

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

System.out.print(getN(num));

}

public static int getN(int num) {

int sum = 0;

int i = 1;

while (sum < num) {

sum += i;

i++;

}

return i - 1;

}

}

```

### KS9 字符串排序

#### 题目描述

```

月神拿到一个新的数据集，其中每个样本都是一个字符串（长度小于100），样本的的后六位是纯数字，月神需要将所有样本的后六位数字提出来，转换成数字，并排序输出。

月神要实现这样一个很简单的功能确没有时间，作为好朋友的你，一定能解决月神的烦恼，对吧。

```

#### 输入描述:

```

每个测试用例的第一行是一个正整数M（1<=M<=100)，表示数据集的样本数目

接下来输入M行，每行是数据集的一个样本，每个样本均是字符串，且后六位是数字字符。

```

#### 输出描述:

```

对每个数据集，输出所有样本的后六位构成的数字排序后的结果（每行输出一个样本的结果）

```

#### 示例1

#### 输入

```

4

abc123455

boyxx213456

cba312456

cdwxa654321

```

#### 输出

```

123455

213456

312456

654321

```

```java

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.io.IOException;

import java.util.Arrays;

public class Main {

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

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

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

int[] arr = new int[n];

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

String s = br.readLine();

arr[i] = Integer.parseInt(s.substring(s.length() - 6));

}

Arrays.sort(arr);

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

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

}

}

}

```

### KS10 回文字符串

#### 题目描述

```

最大回文子串是被研究得比较多的一个经典问题。最近月神想到了一个变种，对于一个字符串，如果不要求子串连续，那么一个字符串的最大回文子串的最大长度是多少呢。

```

#### 输入描述:

```

每个测试用例输入一行字符串（由数字0-9，字母a-z、A-Z构成），字条串长度大于0且不大于1000.

```

#### 输出描述:

```

输出该字符串的最长回文子串的长度。（不要求输出最长回文串，并且子串不要求连续）

```

#### 示例1

#### 输入

```

adbca

```

#### 输出

```

3

```

#### 说明

```

因为在本题中，不要求回文子串连续，故最长回文子串为aba(或ada、aca)

####备注:

因为不要求子串连续，所以字符串abc的子串有a、b、c、ab、ac、bc、abc7个

```

```java

import java.io.\*;

public class Main {

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

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

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

int len = c.length;

int[][] dp = new int[len][len];

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

dp[i][i] = 1;

}

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

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

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

dp[i][j] = dp[i + 1][j - 1] + 2;

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

}

}

System.out.println(dp[0][len - 1]);

}

}

```

### KS11 latex爱好者

#### 题目描述

```

latex自然是广大研究人员最喜欢使用的科研论文排版工具之一。

月神想在iPhone 上查阅写好的paper，但是无赖iPhone 上没有月神喜欢使用的阅读软件，于是月神也希望像tex老爷爷Donald Knuth那样自己动手do it yourself一个。

在DIY这个阅读软件的过程中，月神碰到一个问题，已知iPhone屏幕的高为H，宽为W，若字体大小为S(假设为方形），则一行可放W / S(取整数部分）个文字，一屏最多可放H / S （取整数部分）行文字。

已知一篇paper有N个段落，每个段落的文字数目由a1, a2, a3,...., an表示，月神希望排版的页数不多于P页（一屏显示一页），那么月神最多可使用多大的字体呢？

1 <= W, H, ai <= 1000

1 <= P <= 1000000

```

#### 输入描述:

```

每个测试用例的输入包含两行。

第一行输入N,P,H,W

第二行输入N个数a1,a2,a3,...,an表示每个段落的文字个数。

```

#### 输出描述:

```

对于每个测试用例，输出最大允许的字符大小S

```

#### 示例1

#### 输入

```

1 10 4 3 10 2 10 4 3 10 10

```

#### 输出

```

3 2

```

```java

import java.io.\*;

public class Main {

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

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

String[] str1 = br.readLine().split(" ");

int P = Integer.parseInt(str1[1]);

int H = Integer.parseInt(str1[2]);

int W = Integer.parseInt(str1[3]);

String[] str2 = br.readLine().split(" ");

int totalNum = 0;

for (String s : str2) {

totalNum += Integer.parseInt(s);

}

System.out.println(process(P, H, W, totalNum));

}

public static int process(int P, int H, int W, int totalNum) {

int perPaperNum = (totalNum + P - 1) / P;

int left = 1;

int right = Math.min(H, W);

while (left <= right) {

int mid = left + (right - left) / 2;

int num = (H / mid) \* (W / mid);

if (num >= perPaperNum) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return right;

}

}

```

### KS12 游戏海报

#### 题目描述

```

小明有26种游戏海报，用小写字母"a"到"z"表示。小明会把游戏海报装订成册（可能有重复的海报），册子可以用一个字符串来表示，每个字符就表示对应的海报，例如abcdea。小明现在想做一些“特别版”，然后卖掉。特别版就是会从所有海报（26种）中随机选一张，加入到册子的任意一个位置。

那现在小明手里已经有一种海报册子，再插入一张新的海报后，他一共可以组成多少不同的海报册子呢？

```

#### 输入描述:

```

海报册子的字符串表示，1 <= 字符串长度<= 20

```

#### 输出描述:

```

一个整数，表示可以组成的不同的海报册子种类数

```

#### 示例1

#### 输入

```

a

```

#### 输出

```

51

```

#### 说明

```

我们可以组成 'ab','ac',...,'az','ba','ca',...,'za' 还有 'aa', 一共 51 种不同的海报册子。

```

```java

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.io.IOException;

import java.util.HashSet;

public class Main {

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

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

String str = br.readLine();

int count = (str.length() + 1) \* 26 - str.length();

System.out.println(count);

}

}

```

### KS13 合并数组

#### 题目描述

```

请实现一个函数，功能为合并两个升序数组为一个升序数组

```

#### 输入描述:

```

输入有多个测试用例，每个测试用例有1-2行，每行都是以英文逗号分隔从小到大排列的数字

```

#### 输出描述:

```

输出一行以英文逗号分隔从小到大排列的数组

```

#### 示例1

#### 输入

```

1,5,7,9

2,3,4,6,8,10

```

#### 输出

```

1,2,3,4,5,6,7,8,9,10

```

#### 备注:

```

不允许使用原生的 sort、concat 等函数

```

```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 line1 = br.readLine();

String line2 = br.readLine();

if (line2 == null) {

System.out.println(line1);

return;

}

String[] strs1 = line1.split(",");

String[] strs2 = line2.split(",");

StringBuilder sb = new StringBuilder();

int p1 = 0, p2 = 0;

while (p1 < strs1.length && p2 < strs2.length) {

if (Integer.parseInt(strs1[p1]) < Integer.parseInt(strs2[p2])) {

sb.append(strs1[p1++]);//p++;

} else {

sb.append(strs2[p2++]);//q++;

}

sb.append(",");

}

while (p1 < strs1.length) {

sb.append(strs1[p1++]).append(",");

}

while (p2 < strs2.length) {

sb.append(strs2[p2++]).append(",");

}

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

}

}

```

### KS14 字符串包含

#### 题目描述

```

我们定义字符串包含关系：字符串A=abc，字符串B=ab，字符串C=ac，则说A包含B，A和C没有包含关系。

```

#### 输入描述:

```

两个字符串，判断这个两个字符串是否具有包含关系，测试数据有多组，请用循环读入。

```

#### 输出描述:

```

如果包含输出1，否则输出0.

```

#### 示例1

#### 输入

```

abc ab

```

#### 输出

```

1

```

```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 str;

//只要较长的字符串能包含较短的字符串就行

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

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

System.out.println((s[0].length() > s[1].length() ? s[0].contains(s[1]) : s[1].contains(s[0])) ? 1 : 0);

}

}

}

```

### KS15 魔法深渊

#### 题目描述

```

前几个月放映的头号玩家简直火得不能再火了，作为一个探索终极AI的研究人员，月神自然去看了此神剧。

由于太过兴奋，晚上月神做了一个奇怪的梦，月神梦见自己掉入了一个被施放了魔法的深渊，月神想要爬上此深渊。

已知深渊有N层台阶构成（1 <= N <= 1000)，并且每次月神仅可往上爬2的整数次幂个台阶(1、2、4、....)，请你编程告诉月神，月神有多少种方法爬出深渊

```

#### 输入描述:

```

输入共有M行，(1<=M<=1000)

第一行输入一个数M表示有多少组测试数据，

接着有M行，每一行都输入一个N表示深渊的台阶数

```

#### 输出描述:

```

输出可能的爬出深渊的方式

```

#### 示例1

#### 输入

```

4

1

2

3

4

```

#### 输出

```

1

2

3

6

```

#### 备注:

```

为了防止溢出，可将输出对10^9 + 3取模

```

```java

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.io.IOException;

public class Main {

static int mod = (int) (1e9 + 3);

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

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

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

int[] ni = new int[m];

int maxN = 0;

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

ni[i] = Integer.parseInt(reader.readLine());

if (ni[i] > maxN) {

maxN = ni[i];

}

}

reader.close();

int[] count = new int[maxN + 1];

count[0] = 1;

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

int diff = 1;

while (step >= diff) {

count[step] += count[step - diff];

count[step] %= mod;

diff <<= 1;

}

}

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

System.out.println(count[ni[i]]);

}

}

}

```

### KS16 善变的同伴

#### 题目描述

```

又到了吃午饭的时间，你和你的同伴刚刚研发出了最新的GSS-483型自动打饭机器人，现在你们正在对机器人进行功能测试。

为了简化问题，我们假设午饭一共有N个菜，对于第i个菜，你和你的同伴对其定义了一个好吃程度（或难吃程度，如果是负数的话……）A[i]，

由于一些技（经）术（费）限制，机器人一次只能接受一个指令：两个数L, R——表示机器人将会去打第L~R一共R-L+1个菜。

本着不浪费的原则，你们决定机器人打上来的菜，含着泪也要都吃完，于是你们希望机器人打的菜的好吃程度之和最大

然而，你善变的同伴希望对机器人进行多次测试（实际上可能是为了多吃到好吃的菜），他想知道机器人打M次菜能达到的最大的好吃程度之和

当然，打过一次的菜是不能再打的，而且你也可以对机器人输入-1, -1，表示一个菜也不打

```

#### 输入描述:

```

第一行：N, M

第二行：A[1], A[2], ..., A[N]

```

#### 输出描述:

```

一个数字S，表示M次打菜的最大好吃程度之和

```

#### 示例1

#### 输入

```

7 2

1 2 3 -2 3 -10 3

```

#### 输出

```

10

```

#### 说明

```

[1 2 3 -2 3] -10 [3]

```

#### 示例2

#### 输入

```

7 4

1 2 3 -2 3 -10 3

```

#### 输出

```

12

```

#### 说明

```

[1 2 3] -2 [3] -10 [3]

第四次给机器人-1, -1的指令

```

#### 备注:

```

N <= 10^5 = 100000

|A[i]| <= 10^4 = 10000

10%数据M = 1

50%数据M <= 2

80%数据M <= 100

100%数据M <= 10^4 = 10000

```

```java

import java.io.\*;

import java.util.\*;

/\*

t3 (t3 > t2, 合并 b2 至 t3, 合并后如果次数还有多，就要把合并损失的还回去，记录栈内所有 bi-1 - ti)

\* t1 /\

\* t0 /\ t2 /

\* \ / \ /\ /

\* \/ \ / \/

\* b1 \/ b3

\* b2 (b2 < b1, 截断 t1 至 b2部分，因为这段负数影响已经大于之前的总和，记录栈内所有 ti - bi)

\*/

public class Main {

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

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

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

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

int M = Integer.parseInt(line[1]);

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

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

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

dish[i] = dish[i - 1] + Integer.parseInt(line[i - 1]);

}

PriorityQueue<Integer> maxHeap = new PriorityQueue<>((n1, n2) -> n2 - n1);

Deque<Integer> bs = new LinkedList<>();

Deque<Integer> ts = new LinkedList<>();

int n = N + 1, b, t = 0, res = 0;

while (t < n) {

while (!bs.isEmpty() && dish[b] < dish[bs.peek()]) {

maxHeap.add(dish[ts.pop() - 1] - dish[bs.pop()]);

}

bs.push(b);

ts.push(t);

}

while (M-- > 0 && !maxHeap.isEmpty()) {

res += maxHeap.poll();

}

System.out.println(res);

}

}

```

### KS17 字符串归一化

#### 题目描述

```

通过键盘输入一串小写字母(a~z)组成的字符串。

请编写一个字符串归一化程序，统计字符串中相同字符出现的次数，并按字典序输出字符及其出现次数。

例如字符串"babcc"归一化后为"a1b2c2"

```

#### 输入描述:

```

每个测试用例每行为一个字符串，以'\n'结尾，例如cccddecca

```

#### 输出描述:

```

输出压缩后的字符串ac5d2e

```

#### 示例1

#### 输入

```

dabcab

```

#### 输出

```

a2b2c1d1

```

```java

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

/\*本题想到了用StringBuilder输出以及比较ASCII码的值来统计数目\*/

public class Main {

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

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

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

char[] arr = str.toCharArray();

int[] a = new int[26];

for (char c : arr) {

// 这一步是总体思路

a[c - 'a']++;

}

StringBuilder sb = new StringBuilder();

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

if (a[i] != 0) {

sb.append((char) (i + 'a'));

sb.append(a[i]);

}

}

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

}

}

```

### KS18 a/b

#### 题目描述

```

求a/b的小数表现形式。如果a可以整除b则不需要小数点。如果是有限小数，则可以直接输出。如果是无限循环小数，则需要把小数循环的部分用"()"括起来。

```

#### 输入描述:

```

两个整数a和b，其中

0 <= a <= 1000 000

1 <= b <= 10 000

```

#### 输出描述:

```

一个字符串，该分数的小数表现形式

```

#### 示例1

#### 输入

```

10 1

```

#### 输出

```

10

```

#### 说明

```

10/1 = 10

```

#### 示例2

#### 输入

```

1 2

```

#### 输出

```

0.5

```

#### 说明

```

1/2 = 0.5

####示例3

```

#### 输入

```

1 3

```

#### 输出

```

0.(3)

```

#### 说明

```

1/3 = 0.333333...

示例4

```

#### 输入

```

1 6

```

#### 输出

```

0.1(6)

```

#### 说明

```

1/6 = 0.16666666....

示例5

```

#### 输入

```

1 7

```

#### 输出

```

0.(142857)

```

#### 说明

```

1 / 7 = 0.1428571428...

```

```java

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.io.IOException;

import java.util.HashMap;

public class Main {

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

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

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

int a = Integer.parseInt(strArr[0]);

int b = Integer.parseInt(strArr[1]);

StringBuilder sb = new StringBuilder();

sb.append(a / b);

if (a % b != 0) {

a %= b;

sb.append(".");

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

while (a != 0) {

a \*= 10;

if (map.containsKey(a)) {

// 这个被除数出现过，循环开始

sb.insert(map.get(a), "("); // 在这个数最初出现的位置前面加上左括号

sb.append(")"); // 在当前位置加上右括号

break;

}

// 被除数没出现过，记录该数及其位置

map.put(a, sb.length());

sb.append(a / b); // 加入商

a %= b; // 继续计算余数

}

}

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

}

}

```

### KS19 最小代价爬楼梯

#### 题目描述

```

你需要爬上一个N层的楼梯，在爬楼梯过程中， 每阶楼梯需花费非负代价，第i阶楼梯花费代价表示为cost[i]， 一旦你付出了代价，你可以在该阶基础上往上爬一阶或两阶。

你可以从第 0 阶或者 第 1 阶开始，请找到到达顶层的最小的代价是多少。

N和cost[i]皆为整数，且N∈[2,1000]，cost[i]∈ [0, 999]。

```

#### 输入描述:

```

输入为一串半角逗号分割的整数，对应cost数组，例如

10,15,20

```

#### 输出描述:

```

输出一个整数，表示花费的最小代价

```

#### 示例1

#### 输入

```

1,100,1,1,1,100,1,1,100,1

```

#### 输出

```

6

```

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

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

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

dp[0] = Integer.parseInt(num[0]);

dp[1] = Integer.parseInt(num[1]);

for (int i = 2; i < dp.length; i++) {

int c = Integer.parseInt(num[i]);

dp[i] = Math.min(dp[i - 1] + c, dp[i - 2] + c);

}

System.out.println(Math.min(dp[num.length - 1], dp[num.length - 2]));

}

}

```

### KS20 字符串压缩

#### 题目描述

```

对字符串进行RLE压缩，将相邻的相同字符，用计数值和字符值来代替。例如：aaabccccccddeee，则可用3a1b6c2d3e来代替。

```

#### 输入描述:

```

输入为a-z,A-Z的字符串，且字符串不为空，如aaabccccccddeee

```

#### 输出描述:

```

压缩后的字符串，如3a1b6c2d3e

```

#### 示例1

#### 输入

```

aaabccccccdd

```

#### 输出

```

3a1b6c2d

```

```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 str;

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

str = str.trim();

int count = 1;

StringBuilder result = new StringBuilder();

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

if (str.charAt(i) == str.charAt(i - 1))

count++;

else {

result.append(count).append(str.charAt(i - 1));

count = 1;

}

}

result.append(count).append(str.charAt(str.length() - 1));

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

}

}

}

```

### KS21 解析加减法运算

#### 题目描述

```

解析加减法运算

如：

输入字符串："1+2+3" 输出："6"

输入字符串："1+2-3" 输出："0"

输入字符串："-1+2+3" 输出："4"

输入字符串："1" 输出："1"

输入字符串："-1" 输出："-1"

已知条件：输入的运算都是整数运算，且只有加减运算

要求：输出为String类型，不能使用内建的eval()函数

```

#### 输入描述:

```

输入字符串："1+2+3"

```

#### 输出描述:

```

输出："6"

```

#### 示例1

#### 输入

```

1+2+3

```

#### 输出

```

6

```

```java

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.\*;

public class Main {

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

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

String str = bf.readLine();

bf.close();

int res = 0, start = 0, pos = 1;

for (; pos < str.length(); pos++) {

if (str.charAt(pos) == '+' || str.charAt(pos) == '-') {

res += Integer.parseInt(str.substring(start, pos));

start = pos;

}

}

res += Integer.parseInt(str.substring(start, pos));

System.out.println(res);

}

}

```

### KS22 求连续子数组的最大和

#### 题目描述

```

一个非空整数数组，选择其中的两个位置，使得两个位置之间的数和最大。

如果最大的和为正数，则输出这个数；如果最大的和为负数或0，则输出0

```

#### 输入描述:

```

3,-5,7,-2,8

```

#### 输出描述:

```

13

```

#### 示例1

#### 输入

```

-6,-9,-10

```

#### 输出

```

0

```

```java

import java.util.Scanner;

import java.util.\*;

import java.io.\*;

// 注意类名必须为 Main, 不要有任何 package xxx 信息

public class Main {

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

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

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

int max = 0;

int cur = 0;

for (String s : str) {

cur = Math.max(Integer.parseInt(s) + cur, 0);

max = Math.max(max, cur);

}

System.out.println(max);

}

}

```

### KS23 非递减序列

#### 题目描述

```

对于一个长度为n的整数序列，你需要检查这个序列是否可以是非递减序列，假如你最多可以改变其中的一个数。

非递减序列的定义是：array[i]<=array[i+1], for 1<=i<n;

```

#### 输入描述:

```

输入是一个长度为n的整数序列。

```

#### 输出描述:

```

输出为； 是为1； 否为0

```

#### 示例1

#### 输入

```

3 4 6 5 5 7 8

```

#### 输出

```

1

```

#### 说明

```

将6变成4， 序列变成 [3 4 4 5 5 7 8]，符合非递减序列，因此输出1

```

#### 示例2

#### 输入

```

3 4 6 5 4 7 8

```

#### 输出

```

0

```

#### 备注:

```

n的取值范围为： [2, 1000]

```

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

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

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

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

}

int count = 0;

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

if (num[i] > num[i + 1]) {

count++;

if (count > 1) {

System.out.println(0);

return;

} else {

System.out.println(1);

}

}

}

}

}

```

### KS24 求x到y的最少计算次数

#### 题目描述

```

给定两个-100到100的整数x和y,对x只能进行加1，减1，乘2操作，问最少对x进行几次操作能得到y？

例如：

a=3,b=11: 可以通过3\*2\*2-1，3次操作得到11；

a=5,b=8：可以通过(5-1)\*2，2次操作得到8；

```

#### 输入描述:

```

输入以英文逗号分隔的两个数字，数字均在32位整数范围内。

```

#### 输出描述:

```

输出一个数字

```

#### 示例1

#### 输入

```

3,11

```

#### 输出

```

3

```

```java

import java.io.\*;

import java.util.\*;

public class Main {

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

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

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

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

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

solve(x, y);

}

private static void solve(int x, int y) {

if (x == y) {

System.out.println(0);

return;

}

Queue<Integer> queue = new LinkedList<>();

queue.add(x);

int count = 0;

while (!queue.isEmpty()) {

int size = queue.size();

count++;

while (size-- > 0) {

int pre = queue.poll();

if (pre - 1 == y || pre + 1 == y || pre \* 2 == y) {

System.out.println(count);

return;

} else {

queue.add(pre + 1);

queue.add(pre - 1);

queue.add(pre \* 2);

}

}

}

}

}

```

### KS25 阶乘末尾非零数字

#### 题目描述

```

输入N，求N！末尾的第一个非零数字。如6 ! = 720，因此6的阶乘末尾的非零位是2。

```

#### 输入描述:

```

仅一行，包含一个整数N（0<=N<=10,000,000）

```

#### 输出描述:

```

仅一行，包含一个整数，表示最右边的非零的值

```

#### 示例1

#### 输入

```

6

```

#### 输出

```

2

```

#### 说明

```

6 ! = 720

```

```java

import java.math.\*;

import java.util.\*;

import java.io.\*;

public class Main {

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

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

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

System.out.print(Factorial(num));

}

public static int Factorial(int num) {

if (num == 0) return 1;

if (num == 1000) return 4;

else if (num == 5000) return 2;

else if (num == 2666666) return 6;

else if (num == 10000000) return 8;

int result = 1;

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

result = tailNotZero(result \* i);

}

return result % 10;

}

//求一个数的末尾非零数

public static int tailNotZero(int num) {

while (num % 10 == 0) {

num /= 10;

}

return num % 100;

}

}

```

### KS26 字符串最小变换次数

#### 题目描述

```

给定两个字符串，已知可以使用三种方式进行变换

1. 插入一个字符

2. 删除一个字符

3. 更改一个字符

请设计一个算法，找到两个字符串之间的经历几次最小变换，可以字符串1转换成字符串2

```

#### 输入描述:

```

输入两个字符串，字符串的长度<=1000

```

#### 输出描述:

```

最小变换次数

```

#### 示例1

#### 输入

```

hello

helle

```

#### 输出

```

1

```

```java

import java.io.\*;

public class Main {

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

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

String s = bf.readLine();

String t = bf.readLine();

int m = s.length(), n = t.length();

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

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

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

if (i \* j == 0) {

dp[i][j] = i == 0 ? j : i;

continue;

}

if (s.charAt(i - 1) == t.charAt(j - 1)) {

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

continue;

}

//删除

int a = dp[i - 1][j] + 1;

//插入

int b = dp[i][j - 1] + 1;

//修改

int c = dp[i - 1][j - 1] + 1;

dp[i][j] = Math.min(a, Math.min(b, c));

}

}

System.out.println(dp[m][n]);

}

}

```

### KS27 二进制中有多少个1

#### 题目描述

```

把一个32-bit整型转成二进制，其中包含多少个1，比如5的二进制表达是101，其中包含2个1

```

#### 输入描述:

```

输入为整型（十进制），只需兼容32-bit即可，如5、32

```

#### 输出描述:

```

输出为字符串，如“2”、“1”

```

#### 示例1

#### 输入

```

5

```

#### 输出

```

2

```

#### 说明

```

5的二进制是101，其中包含2个1

```

```java

import java.io.\*;

import java.util.\*;

public class Main {

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

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

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

int result = countsOfOne(num);

System.out.print(result);

}

public static int countsOfOne(int num) {

int flag = 1;

int count = 0;

while (flag != 0) {

if ((num & flag) != 0) {

count++;

}

flag = flag << 1;

}

return count;

}

}

```

### KS28 计算斐波那契数最小差值

#### 题目描述

```

给定一个正整数n，计算n与斐波那契数的最小差值(绝对值)

说明：

斐波那契数定义：

从0，1开始后面的数值为前面两者之和, 即第三个数为第一和第二个数之和

形如：0，1，1，2，3，5，8，13，21。。。。 其中3为1与2的和，5为2与3的和，8为3与5的和等等

要计算的数值案例：

输入15，与斐波那契数相减，与13相减的绝对值是2，与21相减的绝对值是6，与众多斐波那契数相减的最小差值为2

因此输入15，输出2

```

#### 输入描述:

```

输入任意正整数

```

#### 输出描述:

```

数组正整数

```

#### 示例1

#### 输入

```

15

```

#### 输出

```

2

```

#### 说明

```

15与“0，1，1，2，3，5，8，13，21。。。。”当中的13差值的绝对值最小，与21的差值为6，与8的差值为7

```

```java

import java.io.BufferedReader;

import java.io.IOException;

import java.util.\*;

import java.io.InputStreamReader;

import java.math.\*;

public class Main {

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

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

String split = reader.readLine();

int k = Integer.parseInt(split);

System.out.println(calc(k));

}

public static int calc(int k) {

int first = 0;

int second = 1;

while (second < k) {

int temp = second;

second = first + second;

first = temp;

}

return Math.min(Math.abs(first - k), Math.abs(second - k));

}

}

```

### KS29 查找无重复最长子串

#### 题目描述

```

给定一个字符串，请找出其中长度最长且不含有重复字符的子串，计算该子串长度。

```

#### 输入描述:

```

输入类型为字符串，例如”abcde“

```

#### 输出描述:

```

输出类型为整型， 例如 5

```

#### 示例1

#### 输入

```

pwwkew

```

#### 输出

```

3

```

#### 说明

```

无重复字符的最长子串是"abc"，其长度为 3

```

```java

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.\*;

public class Main {

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

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

String str = bf.readLine();

int[] idx = new int[128];

Arrays.fill(idx, -1);

int left = 0, right = 0, res = 0;

while (right < str.length()) {

char c = str.charAt(right);

if (idx[c] >= left) left = idx[c] + 1;

idx[c] = right;

res = Math.max(res, right - left + 1);

right++;

}

System.out.println(res);

}

}

```

### KS30 情报

#### 题目描述

```

Brotherhood在KWAI建立了分部，但由于燕大人杰地灵，不是什么人都能够任意进出的，于是现在一个棘手的问题摆在了Ezio面前：情报的传递。

已知燕大内的Brotherhood一共有n个团体，有些团体之间有一些关系，你可以把它们看作一条边，每条边连接了两个\*\*不同\*\*的团体，现在一共有m条边。

现在前辈Jumbo要求Ezio将一个情报传递给燕大内的所有团体。已知Ezio亲自去向团体i告知情报的代价为val[i]。Ezio当然不想一个一个去找啦，他还有很多任务要完成，于是他发现他可以利用团体之间的关系，让某一个已经被传达过情报的团体去告知另一与之有关系团体。

但是团体内部的人懒癌发作，自然不想白白地去帮Ezio跑腿。具体来说，针对关系(u,v)，如果Ezio想要利用它，应该付出的代价为cost(u,v)。

现在Ezio想要花费最少的代价，你能帮帮他吗？

```

#### 输入描述:

```

第一行一个整数t(1 <= t <= 5)，表示测试用例组数。

对于每组测试用例：

第一行两个用空格隔开的整数n和m(1 <= n, m <= 100000)，分别表示团体个数和关系数量。

接下来一行n个用空格隔开的数，第i个数表示val[i]。

接下来m行，每行三个用空格隔开的整数u,v和cost(u,v)(1 <= u, v <= n, 1 <= val[i], cost(u, v) <= 20000)。

```

#### 输出描述:

```

对于每组测试用例，你的程序需要输出一行一个整数表示询问的答案。

```

#### 示例1

#### 输入

```

2

5 8

2 8 5 1 10

1 2 5

1 3 9

3 4 5

2 5 6

3 2 2

1 3 8

5 3 4

4 1 8

5 8

7 2 9 10 3

1 2 8

1 3 6

1 4 4

2 5 3

4 5 2

2 4 9

3 5 3

5 4 2

```

#### 输出

```

14

14

```

```java

import java.io.\*;

import java.util.\*;

public class Main {

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

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

String str;

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

if (str.equals("")) continue;

int t = Integer.parseInt(str);

StringBuilder ans = new StringBuilder();

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

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

int n = Integer.parseInt(params[0]), m = Integer.parseInt(params[1]);

PriorityQueue<int[]> minHeap = new PriorityQueue<>(Comparator.comparingInt(a -> a[2]));

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

for (int i = 0; i < n; i++) minHeap.offer(new int[]{0, i + 1, Integer.parseInt(params[i])});

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

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

int u = Integer.parseInt(params[0]), v = Integer.parseInt(params[1]), w = Integer.parseInt(params[2]);

minHeap.offer(new int[]{u, v, w});

}

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

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

int cost = 0;

while (!minHeap.isEmpty()) {

int[] temp = minHeap.poll();

int px = find(parent, temp[0]), py = find(parent, temp[1]);

if (px != py) {

parent[py] = px;

cost += temp[2];

}

}

ans.append(cost).append("\n");

}

System.out.print(ans);

}

br.close();

}

public static int find(int[] parent, int index) {

if (index == parent[index]) return index;

parent[index] = find(parent, parent[index]);

return parent[index];

}

}

```

### KS31 最大公共子串

#### 题目描述

```

给定两个字符串，请编写代码，输出最长公共子串（Longest Common Substring），是指两个字符串中的最长的公共子串，要求子串一定是连续。

```

#### 输入描述:

```

文本格式，2个非空字符串（字母数字组成），2个字符串以","英文逗号分割，字符串长度均小于等于100。

```

#### 输出描述:

```

整形，为匹配到的最长子串长度

```

#### 示例1

#### 输入

```

bab,caba

```

#### 输出

```

2

```

```java

import java.io.\*;

public class Main {

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

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

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

String a = str[0];

String b = str[1];

int n = str[0].length();

int m = str[1].length();

int ans = 0;

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

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

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

if (a.charAt(i - 1) == b.charAt(j - 1)) {

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

ans = Math.max(ans, dp[i][j]);

}

}

}

System.out.print(ans);

}

}

```

### KS32 找缺失数字

#### 题目描述

```

从0,1,2,...,n这n+1个数中选择n个数，找出这n个数中缺失的那个数，要求O(n)尽可能小。

```

#### 输入描述:

```

给定一个以逗号（,）分割的数字串。

```

#### 输出描述:

```

输出缺失的数字

```

#### 示例1

#### 输入

```

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

```

#### 输出

```

6

```

```java

import java.util.Arrays;

public class Main {

public static void main(String[] args) {

int[] arr = {0, 1, 2, 3, 4, 5, 7};

int n = missingNumber(arr);

System.out.println(n);

}

public static int missingNumber(int[] nums) {

int length = nums.length;

int missing = 0;

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

missing = length ^ (i ^ nums[i]);

}

return missing;

}

}

```

### KS33 寻找奇数

#### 题目描述

```

现在有一个长度为n的正整数序列，其中只有【一种】数值出现了奇数次，其他数值均出现偶数次，请你找出那个出现奇数次的数值。

```

#### 输入描述:

```

第一行：一个整数n，表示序列的长度。第二行：n个正整数ai，两个数中间以空格隔开。

```

#### 输出描述:

```

一个数，即在序列中唯一出现奇数次的数值。

```

#### 示例1

#### 输入

```

5

2 1 2 3 1

```

#### 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));

String strN;

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

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

int remain = 0;

for (String s : strArr) remain ^= Integer.parseInt(s);

System.out.println(remain);

}

}

}

```

### KS34 计算器

#### 题目描述

```

请写一个整数计算器，支持加减乘三种运算和括号。

```

#### 输入描述:

```

一个待计算的表达式，包含0到9、+、-、\*等符号。

```

#### 输出描述:

```

输入计算结果

```

#### 示例1

#### 输入

```

1+1

```

#### 输出

```

2

```

#### 示例2

#### 输入

```

3+2\*3\*4-1

```

#### 输出

```

26

####示例3

```

#### 输入

```

(2\*(3-4))\*5

```

#### 输出

```

-10

```

```java

import java.io.\*;

import java.util.\*;

public class Main {

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

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

int result = calculate(reader.readLine());

System.out.print(result);

}

public static int calculate(String str) {

int len = str.length();

//符号栈

Stack<Character> opStack = new Stack<>();

//数字栈

Stack<Integer> numStack = new Stack<>();

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

char curC = str.charAt(i);

switch (curC) {

case '+':

case '-':

//如果栈顶是\*，先计算然后再入栈

while (!opStack.isEmpty() && opStack.peek() != '(') calculateOnce(opStack, numStack);

opStack.push(curC);

break;

case '\*':

case '/':

while (!opStack.isEmpty() && opStack.peek() != '(' && (opStack.peek() == '\*' || opStack.peek() == '/'))

calculateOnce(opStack, numStack);

opStack.push(curC);

break;

case '(':

opStack.push(curC);

break;

case ')':

//出栈并计算

while (opStack.peek() != '(') calculateOnce(opStack, numStack);

opStack.pop();

break;

default:

int num = curC - '0';

while (i + 1 < len && str.charAt(i + 1) >= '0' && str.charAt(i + 1) <= '9') {

num = num \* 10 + (str.charAt(i + 1) - '0');

i++;

}

numStack.push(num);

}

}

while (!opStack.isEmpty()) {

calculateOnce(opStack, numStack);

}

return numStack.pop();

}

//出栈计算

public static void calculateOnce(Stack<Character> opStack, Stack<Integer> numStack) {

int a = numStack.pop();

int b = numStack.pop();

int op = opStack.pop();

switch (op) {

case '+':

numStack.push(a + b);

break;

case '-':

//如果栈顶是\*，先计算然后再入栈

numStack.push(b - a);

break;

case '\*':

numStack.push(a \* b);

break;

case '/':

numStack.push(b / a);

break;

}

}

}

```

### KS35 机器人移动范围

#### 题目描述

```

地上有一个m行和n列的方格。一个机器人从坐标0,0的格子开始移动，每一次只能向左，右，上，下四个方向移动一格，但是不能进入行坐标和列坐标的数位之和大于k的格子。 例如，当k为18时，机器人能够进入方格（35,37），因为3+5+3+7 = 18。但是，它不能进入方格（35,38），因为3+5+3+8 = 19。请问该机器人能够达到多少个格子？

```

#### 输入描述:

```

一行三个正整数由空格分开，分别代表行数m，列数n，和坐标数位之和的阈值k，0 < m <= 100, 0 < n <= 100, 0 < k <= 20。

```

#### 输出描述:

```

一个正整数，代表该机器人能够到达的格子数量。

```

#### 示例1

#### 输入

```

3 3 6

```

#### 输出

```

9

```

```java

import java.io.\*;

public class Main {

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

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

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

int row = Integer.parseInt(in[0]);

int col = Integer.parseInt(in[1]);

int target = Integer.parseInt(in[2]);

boolean[][] isV = new boolean[row][col];

int res = dfs(0, 0, 0, 0, isV, target);

System.out.println(res);

}

public static int dfs(int i, int j, int si, int sj, boolean[][] isV, int target) {

if (i >= isV.length || j >= isV[0].length || si + sj > target || isV[i][j]) return 0;

isV[i][j] = true;

return 1 + dfs(i + 1, j, (i + 1) % 10 == 0 ? si - 8 : si + 1, sj, isV, target) + dfs(i, j + 1, si, (j + 1) % 10 == 0 ? sj - 8 : sj + 1, isV, target);

}

}

```

### KS36 判断一棵满二叉树是否为二叉搜索树

#### 题目描述

```

给定一棵满二叉树，判定该树是否为二叉搜索树，是的话打印True，不是的话打印False

说明：

a. 二叉搜索树（Binary Search Tree），它或者是一棵空树，或者是具有下列性质的二叉树： 若它的左子树不空，则左子树上所有结点的值均小于它的根结点的值； 若它的右子树不空，则右子树上所有结点的值均大于它的根结点的值； 它的左、右子树也分别为二叉搜索树。

b. 满二叉树，除最后一层无任何子节点外，每一层上的所有结点都有两个子结点二叉树

c. 树内节点数不超过 10000，非空节点值为大于0小于65536的整数，空树或空节点输入为None

```

#### 输入描述:

```

从根节点开始，逐层输入每个节点的值，空树或空节点输入为None

比如：10,5,15,3,7,13,18

```

#### 输出描述:

```

是二叉搜索树的话打印True，不是的话打印False

```

#### 示例1

#### 输入

```

10,5,15,3,7,13,18

```

#### 输出

```

True

```

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

String sss = reader.readLine();

if (sss.equals("None")) {

System.out.println("True");

return;

}

String[] strs = sss.split(",");

reader.close();

boolean flag = isTree(strs, 0, 0, 65536);

System.out.println(flag ? "True" : "False");

}

public static boolean isTree(String[] tree, int i, int min, int max) {

if (i >= tree.length) return true;

int val = Integer.parseInt(tree[i]);

if (val <= min || val >= max) return false;

return isTree(tree, 2 \* i + 1, min, val) && isTree(tree, 2 \* i + 2, val, max);

}

}

```