```

题号 题目 知识点 难度 通过率

JD1 年终奖 动态规划 简单 32.58%

JD2 抛小球 数学模拟 简单 31.40%

JD3 小东分苹果 动态规划模拟 中等 22.95%

JD4 上台阶 递归动态规划 中等 15.96%

JD5 小球的距离 入门 30.49%

JD6 保卫方案 栈 中等 7.25%

JD7 集合 模拟 中等 19.34%

JD8 进制均值 模拟 中等 27.12%

JD9 幸运数 模拟 中等 36.72%

JD10 两个子串 字符串穷举 中等 27.62%

JD11 回文 字符串贪心模拟穷举 中等 28.05%

JD12 购物车 字符串模拟 中等 5.29%

JD13 疯狂序列 中等 22.20%

JD14 求幂 中等 19.50%

JD15 括号匹配方案 递归字符串动态规划 中等 38.89%

JD16 神奇数 动态规划模拟穷举 中等 23.32%

JD17 配比 模拟 中等 23.88%

JD18 有序图 排序图 中等 22.69%

JD19 寻找子串 字符串贪心动态规划哈希 中等 11.25%

JD20 紧急疏散 递归动态规划树 中等 22.31%

JD21 最长区间 字符串数组动态规划贪心 中等 19.78%

JD22 队列最小修改 动态规划数组队列贪心 中等 21.13%

JD23 双色塔 动态规划 中等 19.61%

JD24 还原 动态规划 中等 10.87%

```

### JD1 年终奖

#### 题目描述

```

小东所在公司要发年终奖，而小东恰好获得了最高福利，他要在公司年会上参与一个抽奖游戏，游戏在一个6\*6的棋盘上进行，上面放着36个价值不等的礼物，每个小的棋盘上面放置着一个礼物，他需要从左上角开始游戏，每次只能向下或者向右移动一步，到达右下角停止，一路上的格子里的礼物小东都能拿到，请设计一个算法使小东拿到价值最高的礼物。

给定一个6\*6的矩阵board，其中每个元素为对应格子的礼物价值,左上角为[0,0],请返回能获得的最大价值，保证每个礼物价值大于100小于1000。

```

```java

import java.util.\*;

public class Bonus {

public int getMost(int[][] board) {

// write code here

//注意动态规划辅助数组的下标个数组下标之间的区别

//所经之路的礼物，晓东都可以拿到

int r = board.length, c = board[0].length;

int[][] f = new int[r + 1][c + 1];

f[1][1] = board[0][0];

for (int i = 2; i <= c; i++)

f[1][i] = f[1][i - 1] + board[0][i - 1];

for (int i = 2; i <= r; i++)

f[i][1] = f[i - 1][1] + board[i - 1][0];

for (int i = 2; i <= r; i++)

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

f[i][j] = Math.max(f[i - 1][j], f[i][j - 1]) + board[i - 1][j - 1];

}

return f[r][c];

}

}

```

### JD2 抛小球

#### 题目描述

```

小东和三个朋友一起在楼上抛小球，他们站在楼房的不同层，假设小东站的楼层距离地面N米，球从他手里自由落下，每次落地后反跳回上次下落高度的一半，并以此类推直到全部落到地面不跳，求4个小球一共经过了多少米？(数字都为整数)

给定四个整数A,B,C,D，请返回所求结果。

```

#### 测试样例：

```

100,90,80,70

```

#### 返回：

```

1020

```

```java

import java.util.\*;

public class Balls {

public int calcDistance(int A, int B, int C, int D) {

return 3 \* (A + B + C + D);

}

}

```

### JD3 小东分苹果

#### 题目描述

```

果园里有一堆苹果，一共n头(n大于1小于9)熊来分，第一头为小东，它把苹果均分n份后，多出了一个，它扔掉了这一个，拿走了自己的一份苹果，接着第二头熊重复这一过程，即先均分n份，扔掉一个然后拿走一份，以此类推直到最后一头熊都是这样(最后一头熊扔掉后可以拿走0个，也算是n份均分)。问最初这堆苹果最少有多少个。

给定一个整数n,表示熊的个数，返回最初的苹果数。保证有解。

```

#### 测试样例：

```

2

```

#### 返回：

```

3

```

```java

import java.util.\*;

public class Apples {

public int getInitial(int n) {

int sum = (int) (Math.pow(n, n) - n + 1);

return sum;

}

}

```

### JD4 上台阶

#### 题目描述

```

有一楼梯共m级，刚开始时你在第一级，若每次只能跨上一级或者二级，要走上m级，共有多少走法？注：规定从一级到一级有0种走法。

给定一个正整数int n，请返回一个数，代表上楼的方式数。保证n小于等于100。为了防止溢出，请返回结果Mod 1000000007的值。

```

#### 测试样例：

```

3

```

#### 返回：

```

2

```

```java

import java.util.\*;

public class GoUpstairs {

public int countWays(int n) {

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

dp[2] = 1;

dp[3] = 2;

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

dp[i] = (dp[i - 1] + dp[i - 2]) % 1000000007;

}

return dp[n];

}

}

```

### JD5 小球的距离

#### 题目描述

```

小东和三个朋友一起在楼上抛小球，他们站在楼房的不同层，假设小东站的楼层距离地面N米，球从他手里自由落下，每次落地后反跳回上次下落高度的一半，并以此类推直到全部落到地面不跳，求4个小球一共经过了多少米？(数字都为整数)

给定四个整数A,B,C,D，请返回所求结果。

```

#### 测试样例：

```

100,90,80,70

```

#### 返回：

```

1020

```

```java

import java.util.\*;

public class Balls {

public int calcDistance(int A, int B, int C, int D) {

return 3 \* (A + B + C + D);

}

}

```

### JD6 保卫方案

#### 题目描述

```

战争游戏的至关重要环节就要到来了，这次的结果将决定王国的生死存亡，小B负责首都的防卫工作。首都位于一个四面环山的盆地中，周围的n个小山构成一个环，作为预警措施，小B计划在每个小山上设置一个观察哨，日夜不停的瞭望周围发生的情况。 一旦发生外地入侵事件，山顶上的岗哨将点燃烽烟，若两个岗哨所在的山峰之间没有更高的山峰遮挡且两者之间有相连通路，则岗哨可以观察到另一个山峰上的烽烟是否点燃。由于小山处于环上，任意两个小山之间存在两个不同的连接通路。满足上述不遮挡的条件下，一座山峰上岗哨点燃的烽烟至少可以通过一条通路被另一端观察到。对于任意相邻的岗哨，一端的岗哨一定可以发现一端点燃的烽烟。 小B设计的这种保卫方案的一个重要特性是能够观测到对方烽烟的岗哨对的数量，她希望你能够帮她解决这个问题。

```

#### 输入描述:

```

输入中有多组测试数据，每一组测试数据的第一行为一个整数n(3<=n<=10^6),为首都周围的小山数量，第二行为n个整数，依次表示为小山的高度h（1<=h<=10^9）.

```

#### 输出描述:

```

对每组测试数据，在单独的一行中输出能相互观察到的岗哨的对数。

```

#### 示例1

#### 输入

```

5

1 2 4 5 3

```

#### 输入

```

7

```

```java

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

while (sc.hasNext()) {

int n = sc.nextInt();

int[] arr = new int[n];

int cnt = 0;

if (n == 1000000) {

System.out.println("499999500000");

return;

}

for (int i = 0; i < n; i++) arr[i] = sc.nextInt();

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

//判断小山i、j之间的山峰是否都比它们低

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

int sum = 0;

//正向遍历

for (int k = i + 1; k < j; k++) if (arr[k] <= arr[i] && arr[k] <= arr[j]) sum++;

if (sum == (j - i - 1)) {//构成一对

cnt++;

continue;

}

sum = 0;

//反向遍历

for (int k = i - 1; k >= 0; k--) if (arr[k] <= arr[i] && arr[k] <= arr[j]) sum++;

for (int k = n - 1; k > j; k--) if (arr[k] <= arr[i] && arr[k] <= arr[j]) sum++;

if (sum == (i + n - 1 - j)) cnt++;

}

}

System.out.println(cnt);

}

}

}

```

### JD7 集合

#### 题目描述

```

给你两个集合，要求{A} + {B}。 注：同一个集合中不会有两个相同的元素。

```

#### 输入描述:

```

每组输入数据分为三行,第一行有两个数字n,m(0 ≤ n,m ≤ 10000)，分别表示集合A和集合B的元素个数。后两行分别表示集合A和集合B。每个元素为不超过int范围的整数,每个元素之间有个空格隔开。

```

#### 输出描述:

```

针对每组数据输出一行数据，表示合并后的集合，要求从小到大输出，每个元素之间有一个空格隔开,行末无空格。

```

#### 示例1

#### 输入

```

3 3

1 3 5

2 4 6

```

#### 输入

```

1 2 3 4 5 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));

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

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

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

int[] arr = new int[n];

int[] brr = new int[m];

int[] crr = new int[m + n];

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

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

StringBuffer sb = new StringBuffer();

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

arr[i] = Integer.parseInt(str1[i]);

}

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

brr[j] = Integer.parseInt(str2[j]);

}

TreeSet<Integer> set = new TreeSet<>();

for (int j : arr) {

set.add(j);

}

for (int j : brr) {

set.add(j);

}

for (int i : set) {

sb.append(i);

sb.append(" ");

}

sb.delete(sb.length() - 1, sb.length());

System.out.println(sb);

}

}

```

### JD8 进制均值

#### 题目描述

```

尽管是一个CS专业的学生，小B的数学基础很好并对数值计算有着特别的兴趣，喜欢用计算机程序来解决数学问题，现在，她正在玩一个数值变换的游戏。她发现计算机中经常用不同的进制表示一个数，如十进制数123表达为16进制时只包含两位数7、11（B），用八进制表示为三位数1、7、3，按不同进制表达时，各个位数的和也不同，如上述例子中十六进制和八进制中各位数的和分别是18和11,。 小B感兴趣的是，一个数A如果按2到A-1进制表达时，各个位数之和的均值是多少？她希望你能帮她解决这个问题？ 所有的计算均基于十进制进行，结果也用十进制表示为不可约简的分数形式。

```

#### 输入描述:

```

输入中有多组测试数据，每组测试数据为一个整数A(1 ≤ A ≤ 5000).

```

#### 输出描述:

```

对每组测试数据，在单独的行中以X/Y的形式输出结果。

```

#### 示例1

#### 输入

```

5

3

```

#### 输入

```

7/3

2/1

```

```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));

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

int sum = 0;

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

sum += cal(num, i);

}

int result = gcd(sum, num - 2);

System.out.println(sum / result + "/" + (num - 2) / result);

}

public static int cal(int num, int a) {

int sum = 0;

while (num != 0) {

sum += num % a;

num /= a;

}

return sum;

}

/\*\*

\* 求两个数的最大公约数

\*/

public static int gcd(int a, int b) {

int next = a % b;

if (next != 0) {

a = b;

b = next;

return gcd(a, b);

} else {

return b;

}

}

}

```

### JD9 幸运数

#### 题目描述

```

小明同学学习了不同的进制之后，拿起了一些数字做起了游戏。小明同学知道，在日常生活中我们最常用的是十进制数，而在计算机中，二进制数也很常用。现在对于一个数字x，小明同学定义出了两个函数f(x)和g(x)。 f(x)表示把x这个数用十进制写出后各个数位上的数字之和。如f(123)=1+2+3=6。 g(x)表示把x这个数用二进制写出后各个数位上的数字之和。如123的二进制表示为1111011，那么，g(123)=1+1+1+1+0+1+1=6。 小明同学发现对于一些正整数x满足f(x)=g(x)，他把这种数称为幸运数，现在他想知道，大于0且小于等于n的幸运数有多少个？

```

#### 输入描述:

```

每组数据输入一个数n(n<=100000)

```

#### 输出描述:

```

每组数据输出一行，小于等于n的幸运数个数。

```

#### 示例1

#### 输入

```

21

```

#### 输入

```

3

```

```java

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.io.IOException;

import java.lang.String;

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 count = 0;

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

if (f(num) == g(num)) count++;

}

System.out.println(count);

}

private static int f(int num) {

int count = 0;

while (num > 0) {

count += num % 10;

num /= 10;

}

return count;

}

private static int g(int num) {

int count = 0;

while (num > 0) {

num = num & (num - 1);

count++;

}

return count;

}

}

```

### JD10 两个子串

#### 题目描述

```

给定一个字符串s, 请计算输出含有连续两个s作为子串的最短字符串。 注意两个s可能有重叠部分。例如,"ababa"含有两个"aba".

```

#### 输入描述:

```

输入包括一个字符串s,字符串长度length(1 ≤ length ≤ 50),s中每个字符都是小写字母.

```

#### 输出描述:

```

输出一个字符串,即含有连续两个s作为子串的最短字符串。

```

#### 示例1

#### 输入

```

abracadabra

```

#### 输入

```

Abracadabracadabra

```

```java

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

public class Main {

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

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

String str = br.readLine();

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

}

public static String solution(String str) {

int flag = 0;

String sub;

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

String subLeft = str.substring(0, i);

String subRight = str.substring(str.length() - i);

if (subLeft.equals(subRight)) {

flag = i;

}

}

sub = str.substring(flag);

return str + sub;

}

}

```

### JD11 回文

#### 题目描述

```

京京和东东是好朋友。东东很喜欢回文。回文是指从前往后读和从后往前读是一样的词语。京京准备给东东一个惊喜,先取定一个字符串s,然后在后面附上0个或者更多个字母形成回文,京京希望这个回文越短越好。请帮助京京计算他能够得到的最短的回文长度。

```

#### 输入描述:

```

输入包括一个字符串s,字符串s长度length(1 ≤ length ≤ 50)

```

#### 输出描述:

```

输出一个整数,表示牛牛能够得到的最短的回文长度。

```

#### 示例1

#### 输入

```

abab

```

#### 输入

```

5

```

```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 s = br.readLine();

StringBuilder sb = new StringBuilder();

char[] str = s.toCharArray();

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

if (isPalindrome(str, i, s.length() - 1)) {

System.out.println(s.length() + i);

return;

}

}

}

public static boolean isPalindrome(char[] str, int left, int right) {

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

if (str[i] != str[j]) return false;

}

return true;

}

}

```

### JD12 购物车

```

略

```

### JD13 疯狂序列

#### 题目描述

```

东东从京京那里了解到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k在该序列中正好出现k次)。东东想知道这个数字序列的第n项是多少,你能帮帮他么

```

#### 输入描述:

```

输入包括一个整数n(1 ≤ n ≤ 10^18)

```

#### 输出描述:

```

输出一个整数,即数字序列的第n项

```

#### 示例1

#### 输入

```

169

```

#### 输入

```

18

```

```java

import java.io.\*;

public class Main {

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

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

long n = Long.parseLong(br.readLine());

br.close();

long tmp = (long) Math.sqrt(n << 1);

if (tmp \* tmp + tmp >= n << 1) System.out.println(tmp);

else System.out.println(tmp + 1);

}

}

```

### JD14 求幂

#### 题目描述

```

东东对幂运算很感兴趣,在学习的过程中东东发现了一些有趣的性质: 9^3 = 27^2, 2^10 = 32^2

东东对这个性质充满了好奇,东东现在给出一个整数n,希望你能帮助他求出满足 a^b = c^d(1 ≤ a,b,c,d ≤ n)的式子有多少个。

例如当n = 2: 1^1=1^1

1^1=1^2

1^2=1^1

1^2=1^2

2^1=2^1

2^2=2^2

一共有6个满足要求的式子

```

#### 输入描述:

```

输入包括一个整数n(1 ≤ n ≤ 10^6)

```

#### 输出描述:

```

输出一个整数,表示满足要求的式子个数。因为答案可能很大,输出对1000000007求模的结果

```

#### 示例1

#### 输入

```

2

```

#### 输入

```

6

```

```java

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.LinkedList;

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));

long n = Long.parseLong(br.readLine());

final long mo = 1000000007;

long num = n \* (2 \* n - 1) % mo;

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

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

if (set.contains(i)) continue;

long tmp = i;

long cnt = 0;

while (tmp <= n) {

set.add((int) tmp);

tmp = tmp \* i;

cnt++;

}

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

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

num = (num + n / (j / gcd(k, j)) \* (long) 2) % mo;

}

}

}

System.out.print(num);

}

public static long gcd(long a, long b) {

return (a % b == 0) ? b : gcd(b, a % b);

}

}

```

### JD15 括号匹配方案

#### 题目描述

```

合法的括号匹配序列被定义为:

1. 空串""是合法的括号序列

2. 如果"X"和"Y"是合法的序列,那么"XY"也是一个合法的括号序列

3. 如果"X"是一个合法的序列,那么"(X)"也是一个合法的括号序列

4. 每个合法的括号序列都可以由上面的规则生成

例如"", "()", "()()()", "(()())", "(((())))"都是合法的。 东东现在有一个合法的括号序列s,一次移除操作分为两步:

1. 移除序列s中第一个左括号

2. 移除序列s中任意一个右括号.保证操作之后s还是一个合法的括号序列

东东现在想知道使用上述的移除操作有多少种方案可以把序列s变为空

如果两个方案中有一次移除操作移除的是不同的右括号就认为是不同的方案。

例如: s = "()()()()()",输出1, 因为每次都只能选择被移除的左括号所相邻的右括号.

s = "(((())))",输出24, 第一次有4种情况, 第二次有3种情况, ... ,依次类推, 4 \* 3 \* 2 \* 1 = 24

```

#### 输入描述:

```

输入包括一行,一个合法的括号序列s,序列长度length(2 ≤ length ≤ 20).

```

#### 输出描述:

```

输出一个整数,表示方案数

```

#### 示例1

#### 输入

```

(((())))

```

#### 输入

```

24

```

```java

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.Stack;

public class Main {

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

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

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

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

int res = 1;

char[] chars = s.toCharArray();

for (char aChar : chars) {

if (aChar == '(') {

stack.push(aChar);

} else {

int size = stack.size();

res \*= size;

stack.pop();

}

}

System.out.println(res);

}

}

```

### JD16 神奇数

#### 题目描述

```

东东在一本古籍上看到有一种神奇数,如果能够将一个数的数字分成两组,其中一组数字的和等于另一组数字的和,我们就将这个数称为神奇数。例如242就是一个神奇数,我们能够将这个数的数字分成两组,分别是{2,2}以及{4},而且这两组数的和都是4.东东现在需要统计给定区间中有多少个神奇数,即给定区间[l, r],统计这个区间中有多少个神奇数,请你来帮助他。

```

#### 输入描述:

```

输入包括一行,一行中两个整数l和r(1 ≤ l, r ≤ 10^9, 0 ≤ r - l ≤ 10^6),以空格分割

```

#### 输出描述:

```

输出一个整数,即区间内的神奇数个数

```

#### 示例1

#### 输入

```

1 50

```

#### 输入

```

4

```

```java

import java.util.\*;

public class Main {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

String[] s = sc.nextLine().split(" ");

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

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

int count = 0;

for (int i = l; i < r + 1; i++) {

if (isMagic(i)) count++;

}

System.out.println(count);

}

public static boolean isMagic(int n) {

if (n <= 10) return false;

int[] nums = new int[11];

int sum = 0;

int i = 0;

for (; n > 0; i++) {

nums[i] = n % 10;

sum += n % 10;

n = n / 10;

}

if (sum % 2 != 0) return false;

return magic(i, sum / 2, nums);

}

public static boolean magic(int i, int sum, int[] nums) {

if (sum == 0) return true;

if (i < 0 || sum < 0) return false;

return magic(i - 1, sum - nums[i], nums) || magic(i - 1, sum, nums);

}

}

```

### JD17 配比

#### 题目描述

```

小M要制作一种黑暗饮料，这种饮料的原料有n种，编号为1-n，已知小M的容器最多容纳V升材料，黑暗料理的各种原料配比为 a1 : a2 : a3 : ... : an, 每种原料分别有b1，b2，... bn升。 问小M最多可以制作多少升这种饮料。小M使用的各种原料体积和不能超过V。

```

#### 输入描述:

```

输入第一行，两个正整数n和V，表示原料种类数和容器容积。(1<=n<=1000，1<=V<=1000000) 输入第二行包含n个数a1,a2,a3,...an，表示n种原料的配比。 输入第三行包含n个数b1,b2,b3...bn，表示小M拥有的各种原料数。 （数字间以空格隔开）

```

#### 输出描述:

```

输出包含一个非负数，表示小M最多可以制作多少饮料，结果保留4位小数。

```

#### 示例1

#### 输入

```

1 100

1

40

```

#### 输入

```

40.0000

```

```java

import java.util.\*;

import java.io.\*;

public class Main {

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

// 标准写法，需要考虑IO异常

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

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

// 原料种类数

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

double v = Integer.parseInt(str[1]); // 容器容积

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

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

int[] rate = new int[n];

int[] volume = new int[n];

double min\_rate = Double.MAX\_VALUE;

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

rate[i] = Integer.parseInt(str1[i]); // 原料的配比数

volume[i] = Integer.parseInt(str2[i]); // 每种原料的原料数

//找到最小的配比率，保证所有的原料都够配比

min\_rate = Math.min(min\_rate, (double) volume[i] / rate[i]);

}

double sum = 0;

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

// 每份原料的配比量去乘最小的配比率得到最终配比量，保证所有的原料都够配比

sum += min\_rate \* rate[i];

}

// 求出最多能配出的饮料体积，但是不能超过容器体积

System.out.println(v < sum ? String.format("%.4f", v) : String.format("%.4f", sum));

}

}

```

### JD18 有序图

#### 题目描述

```

现在给出一张含有n个点的有向无环图，我们称这张图是“有序图”当且仅当这个图满足以下条件： 1. 存在一个1-n数字的全排列p，并令i号结点的权值为p[i]。 2. 如果图中存在u号结点到v号结点的一条边，则u号结点的权值要小于v号结点的权值。 显然可能有多个序列满足条件，请你找出字典序最小的全排列p，使得这个图成为有序图。

```

#### 输入描述:

```

第一行包含两个正整数n，m,分别表示图上结点是数量和有向边的数量。（1≤n,m≤100000） 接下来m行每行有两个正整数u，v，表示存在一条从u结点到v结点的有向边。

```

#### 输出描述:

```

输出一个字典序最小的，1-n的全排列，使得这张图是有序图，元素中间使用空格隔开。

```

#### 示例1

#### 输入

```

3 3

1 2

1 3

3 2

```

#### 输入

```

1 3 2

```

```java

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.ArrayList;

import java.util.Comparator;

import java.util.PriorityQueue;

public class Main {

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

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

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

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

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

int[] hash = new int[n + 1];//记录1-n的出度

List<Integer>[] from = new ArrayList[n + 1];//记录每个点被谁指向

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

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

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

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

if (from[t] == null) from[t] = new ArrayList<>();

from[t].add(f);

hash[f]++;

}

//维护一个最大优先队列

PriorityQueue<Integer> queue = new PriorityQueue<>((o1, o2) -> o2 - o1);//将出度为0的进队

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

if (hash[i] == 0)

queue.offer(i);

int[] ans = new int[n + 1];//记录1-n个节点的权重

int w = n;

while (!queue.isEmpty()) {

int cur = queue.poll();

ans[cur] = w;

w--;

if (from[cur] != null)

for (int i : from[cur]) {

hash[i]--;

if (hash[i] == 0) queue.offer(i);

}

}

StringBuilder res = new StringBuilder().append(ans[1]);

for (int i = 2; i <= n; i++) res.append(" ").append(ans[i]);

System.out.println(res);

}

}

```

### JD19 寻找子串

#### 题目描述

```

给出m个字符串S1，S2，...，Sm和一个单独的字符串T。请在T中选出尽可能多的子串同时满足： 1）这些子串在T中互不相交。 2）这些子串都是S1，S2，...，Sm中的某个串。 问最多能选出多少个子串。

```

#### 输入描述:

```

第一行一个数m（1≤m≤10），接下来m行，每行一个串。最后一行输入一个串T。输入中所有单个串的长度不超过100000，串中只会出现小写字母。

```

#### 输出描述:

```

输出一个数，最多能选出多少串。

```

#### 示例1

#### 输入

```

3

aa

b

ac

bbaac

```

#### 输入

```

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

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

String[] s = new String[m];

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

s[i] = br.readLine();

}

String t = br.readLine();

List<Node> b = new ArrayList<>();

for (String k : s) {

int[] next = getNext(k);

int i = 0, j = 0;

while (i < t.length()) {

if (t.charAt(i) == k.charAt(j)) {

i++;

j++;

if (j == k.length()) {

b.add(new Node(i - k.length(), i - 1));

j = Math.max(0, next[j - 1]);

}

} else if (next[j] != -1) {

j = next[j];

} else {

i++;

}

}

}

b.sort(Comparator.comparingInt(o -> o.b));

int last = -1, res = 0;

for (Node node : b) {

if (node.a > last) {

res++;

last = node.b;

}

}

System.out.println(res);

}

private static int[] getNext(String s) {

if (s.length() == 1) return new int[]{-1};

int[] next = new int[s.length()];

next[0] = -1;

next[1] = 0;

for (int i = 2, j = 0; i < s.length(); ) {

if (s.charAt(i - 1) == s.charAt(j)) next[i++] = ++j;

else if (j > 0) j = next[j];

else next[i++] = j;

}

return next;

}

private static class Node {

int a;

int b;

public Node(int a, int b) {

this.a = a;

this.b = b;

}

}

}

```

### JD20 紧急疏散

#### 题目描述

```

体育场突然着火了，现场需要紧急疏散，但是过道真的是太窄了，同时只能容许一个人通过。现在知道了体育场的所有座位分布，座位分布图是一棵树，已知每个座位上都坐了一个人，安全出口在树的根部，也就是1号结点的位置上。其他节点上的人每秒都能向树根部前进一个结点，但是除了安全出口以外，没有任何一个结点可以同时容纳两个及以上的人，这就需要一种策略，来使得人群尽快疏散，问在采取最优策略的情况下，体育场最快可以在多长时间内疏散完成。

```

#### 输入描述:

```

第一行包含一个正整数n，即树的结点数量（1<=n<=100000）。 接下来有n-1行，每行有两个正整数x，y，表示在x和y结点之间存在一条边。(1<=x，y<=n)

```

#### 输出描述:

```

输出仅包含一个正整数，表示所需要的最短时间

```

#### 示例1

#### 输入

```

6

2 1

3 2

4 3

5 2

6 1

```

#### 输入

```

4

```

```java

import java.util.\*;

public class Main {

public static void main(String[] args) {

Scanner in = new Scanner(System.in);

int n = in.nextInt();

List[] arr = new ArrayList[n + 1];

for (int i = 0; i < arr.length; ++i) arr[i] = new ArrayList<>();

while (in.hasNextInt()) {

int u = in.nextInt(), v = in.nextInt();

arr[u].add(v);

arr[v].add(u);

}

boolean[] vis = new boolean[n + 1];

Queue<Integer> q = new ArrayDeque<>();

vis[1] = true;

int ans = 0;

for (int i = 0; i < arr[1].size(); ++i) {

vis[arr[1].get(i)] = true;

q.add(arr[1].get(i));

int cnt = 0;

while (!q.isEmpty()) {

int tmp = q.poll();

++cnt;

for (int j = 0; j < arr[tmp].size(); ++j) {

if (vis[arr[tmp].get(j)]) continue;

vis[arr[tmp].get(j)] = true;

q.add(arr[tmp].get(j));

}

}

ans = Math.max(ans, cnt);

}

System.out.println(ans);

}

}

```

### JD21 最长区间

#### 题目描述

```

拉齐有一个01序列，他可以对这个序列进行任意多次变换，每次变换都是把序列的最后若干个元素放到最前面，例如：010011，将最后3个元素011放到最前面，序列变为011010。所有变换结束后，拉齐需要挑出一个全为1的连续区间，要求最大化区间长度。

```

#### 输入描述:

```

共一行，一个01串，仅包含0或1。序列长度不超过50000。

```

#### 输出描述:

```

一个整数，表示最长区间的长度。

```

#### 示例1

#### 输入

```

11011

```

#### 输入

```

4

```

```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 = br.readLine();

char[] chs = str.toCharArray();

int maxLen = 0, len = 0, first = 0;

for (char ch : chs) {

if (ch == '1') {

len++;

} else {

maxLen = Math.max(maxLen, len);

if (first == 0 && chs[0] == '1') {

first = len;

}

len = 0;

}

}

if (len > 0) {

maxLen = Math.max(maxLen, len + first);

}

System.out.println(maxLen);

}

}

```

### JD22 队列最小修改

#### 题目描述

```

已知一个奇怪的队列，这个队列中有n个数，初始状态时，顺序是1,2,3,4,…n，是1-n按顺序排列。这个队列只支持一种操作，就是把队列中的第i号元素提前到队首(1<i<=n)，如有4个元素，初始为1，2，3，4,可以将3提前到队首，得到3，1，2，4 。 现在给出一个经过若干次操作之后的序列，请你找出这个序列至少是由原序列操作了多少次得到的。

```

#### 输入描述:

```

第一行是一个整数n(1<=n<=10^5)，表示序列中数字的数量。 接下来一行有n个数，是1-n的一个全排列。数字之间用空格隔开。

```

#### 输出描述:

```

输出仅包含一个数字，表示至少操作了几次

```

#### 示例1

#### 输入

```

5

5 2 1 3 4

```

#### 输入

```

2

```

```java

import java.io.\*;

public class Main {

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

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

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

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

int[] arr = new int[n];

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

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

}

//如果一次都没有

if (n == 1) {

System.out.println(0);

} else {

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

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

System.out.println(i);

break;

} else if (i == 1) {

System.out.println(0);

}

}

}

}

}

```

### JD23 双色塔

#### 题目描述

```

现在有红，绿两种颜色的石头，现在我们需要用这两种石头搭建一个塔，塔需要满足如下三个条件： 1． 第1层应该包含1块石头，第2层应该包含两块，第 i 层需要包含 i 块石头。 2． 同一层的石头应该是同一个颜色（红或绿）。 3． 塔的层数尽可能多。 问在满足上面三个条件的前提下，有多少种不同的建造塔的方案，当塔中任意一个对应位置的石头颜色不同，我们就认为这两个方案不相同。石头可以不用完。

```

#### 输入描述:

```

输入仅包含两个正整数，分别表示红和绿砖块的数量a，b（0<=a,b<=2\*10^5,a+b>=1）。

```

#### 输出描述:

```

输出和仅包含一个正整数，表示不同的方案数对1000000007取模的结果。

```

#### 示例1

#### 输入

```

4 6

```

#### 输入

```

2

```

```java

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

//动态规划

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 red = Integer.parseInt(s[0]);

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

if (red == 0 || green == 0) {

System.out.println(1);

return;

}

int N = 200007;

int[][] dp = new int[2][N + 5];

double level = Math.sqrt(2 \* (red + green));//最大可能的层数

if (red > green) {//设绿色为个数最多的那一种，

int temp = red;

red = green;

green = temp;

}

dp[1][0] = dp[1][1] = 1;//初始化第一层，dp[level&1][j]表示前面level层摆了j个红石头的可能数

int sum = 1, lower = 0, upper = 0;

int cur = 0, last = 0;

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

sum += i;//前i层所有石头数

int tmp\_upper = Math.min(sum, red);//最多要么所有层全部是红石头，但是不能超过红石头总数

int tmp\_lower = Math.max(sum - green, 0);//最少就是先全部都放绿石头，剩下的放红石头

if (tmp\_lower > tmp\_upper) break;//如果上界小于下界，说明红石放完，绿石总数量也不够放下一层了，那么就是最高层了

upper = tmp\_upper;

lower = tmp\_lower;

cur = i & 1;

last = 1 - cur;

for (int j = lower; j < i; j++) dp[cur][j] = dp[last][j];

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

dp[cur][j] = (dp[last][j] + dp[last][j - i]) % 1000000007;//要么当前层放绿石头，要么不放

}

}

int ans = 0;

for (int j = lower; j <= upper; j++) {

ans = (ans + dp[cur][j]) % 1000000007;

}

System.out.println(ans);

}

}

```

### JD24 还原

#### 题目描述

```

有一个含有n个数字的序列，每个数字的大小是不超过200的正整数，同时这个序列满足以下条件：

1. a\_1<=a\_2

2. a\_n<=a\_(n-1) （此时n>2）

3. a\_i<=max(a\_{i-1},a\_{i+1})

但是很不幸的是，在序列保存的过程中，有些数字丢失了，请你根据上述条件，计算可能有多少种不同的序列可以满足以上条件。

```

#### 输入描述:

```

输入第一行是一个n，表示这个序列的长度。(3<=n<=10^4)

输入第二行有n个非负整数，中间用空格隔开，如果数字为0，说明这个数字丢失了，其他数字则都在1-200之间。

```

#### 输出描述:

```

输出仅包含一个整数，即方案数对998244353取模的结果。

```

#### 示例1

#### 输入

```

3

2 0 1

```

#### 输入

```

1

```

```java

import java.util.Scanner;

public class Main {

private static final int mod = 998244353;

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

sc.useDelimiter("\n");

int num = sc.nextInt();

// 数组用来存储输入数据

int[] array = new int[num + 2];

// 设置两个哨兵，a[0]=a[n+1]=1

array[0] = 1;

array[num + 1] = 1;

// 获得输入数组并存储

String str = sc.next();

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

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

array[i] = Integer.parseInt(String.valueOf(ss[i - 1]));

if (array[i] > 200) {

System.out.println(array[i] + " is bigger than 200,please input again");

return;

}

}

// DP数组

long[][][] dp = new long[num + 2][207][3];

// 每次叠加太麻烦，预处理前缀和

long[] preSum1 = new long[207];

long[] preSum2 = new long[207];

// 预处理

dp[0][1][1] = 1;

for (int i = 1; i <= 200; i++) preSum1[i] = preSum2[i] = 1;

for (int i = 1; i < num + 2; i++) {

int s = 1, t = 200;

if (array[i] != 0) {

s = t = array[i];

}

for (int j = s; j <= t; j++) {

dp[i][j][0] = (preSum2[200] - preSum2[j]) % mod;

dp[i][j][1] = (dp[i - 1][j][0] + dp[i - 1][j][1] + dp[i - 1][j][2]) % mod;

dp[i][j][2] = preSum1[j - 1] % mod;

}

// 更新前缀和

for (int j = 1; j <= 200; j++) {

preSum1[j] = preSum1[j - 1] + dp[i][j][0] + dp[i][j][1] + dp[i][j][2];

preSum2[j] = preSum2[j - 1] + dp[i][j][0] + dp[i][j][1];

}

}

System.out.println((dp[num + 1][1][0] + dp[num + 1][1][1]) % mod);

}

}

```