

# 《算法竞赛中的初等数论》

数论全家桶（信奥 / 数竞 / ACM）

繁凡さん（孟繁宇）

2021/1/29

<https://fanfansann.blog.csdn.net/>

# 前言

数论是纯粹数学的分支之一，主要研究整数的性质。整数可以是方程式的解（丢番图方程）。有些解析函数（像黎曼  $\zeta$  函数）中包括了一些整数、质数的性质，透过这些函数也可以了解一些数论的问题。透过数论也可以建立实数和有理数之间的关系，并且用有理数来逼近实数（丢番图逼近）。

按研究方法来看，数论大致可分为初等数论和高等数论。初等数论是用初等方法研究的数论，它的研究方法本质上说，就是利用整数环的整除性质，主要包括整除理论、同余理论、连分数理论。高等数论则包括了更为深刻的数学研究工具。它大致包括代数数论、解析数论、计算数论等等。

简而言之,数论就是 **研究整数的理论**，在ACM / OI 竞赛中,经常用到数论的相关知识，纯粹考察数论的题目并不会很多，大部分是和其他类型的问题结合起来的，常考知识：约数，倍数，模线性方程，欧拉定理，素数，反演等等。

本文着重于 **编程竞赛**，即 **国际大学生程序设计竞赛** ACM / ICPC、**信息学奥林匹克竞赛** NOIP，NOI，IOI等编程竞赛中相关的**初等数论问题**，包含几乎所有数论相关内容的**定理，概念，代码实现，部分证明以及例题选讲**，更多**定理证明内容**参见《初等数论及其应用》和《具体数学》。

由于涵盖了初等数论的绝大多数概念，定理，证明，推论，所以本文同样适合**数学竞赛**的同学阅读使用。

其中：

**0x00整除**，讲解了一些整除的基础知识，理解这些内容是我们学习接下来的内容的基础。

**0x10整除相关**，为大家系统地讲解了初等数论研究的最重要的内容之一：素数与约数，包括素数、反素数、唯一分解定理、最大公约数、最小公倍数、互质与欧拉函数等整除相关的基本概念，并稍作拓展，部分小节为大家提供了一些经典例题，供大家学习巩固使用，还讲解了容斥原理的相关概念和在数论里的两个经典运用，是非常重要的知识点。最后拓展了一下素数在实际生活中的应用：加密与解密，感兴趣的读者可以深入学习一下。

**0x20同余**，同余及同余方程，初等数论研究的另一大版块，也是竞赛要求的基础知识，首先为大家带来的模运算这个重要概念，在了解了同余的概念以后又为大家简单讲解了竞赛常用定理，欧几里得算法，拓展欧几里德算法，类欧几里得算法，求乘法逆元，解线性同余方程、高次同余方程等基本算法，需要大家理解并掌握。前三章是数论的基础，请一定掌握。

**0x30积性函数**，从第四章开始，就进入了进阶版的数论知识，我为大家讲解了数学中会涉及到的一些函数，以及最重要的积性函数的相关概念，常见积性函数，并简要证明了莫比乌斯函数是积性函数，其余积性函数的证明方法大同小异，留给读者自己尝试。介绍了较为重要的狄利克雷卷积概念，即两个积性函数间的一种基本运算，给出了一些性质以及推导结论，希望读者可以理解并自己推导，理解这些内容是大家迈向高阶数论的基石。

**0x40反演**，从最基础的欧拉反演、莫比乌斯反演出发，为大家列举了包括二项式反演、斯特林反演、单位根反演、子集反演、最值反演、拉格朗日反演等“**简单**”反演的基本性质定理以及部分推导出的“**常用**”结论，希望读者可以阅读以后自行证明。除此之外，还为大家列举了一些常用技巧，以及在求解积性函数时有奇效的 Dirichlet 前缀和的相关概念和实现代码。

**0x50筛法**，即积性函数的筛法，从基础的线性筛法拓展开来，介绍了复杂度较之线性筛更为优秀的杜教筛的相关证明及应用，并提供了更加优秀的 Min\_25 筛、sieve 筛的基本模板和使用用法。

**0x60原根**，研究了正整数  $n$  的模  $n$  整数集中的乘法结构，介绍了模  $n$  整数的阶的基本性质，引出原根的概念。提供了原根的性质、求法、离散对数这一重要概念以及竞赛相关的一些简单应用，如原根在快速数论变换的应用以及证明。并讲解了前面遗留的第二种高次同余方程（ $N$  次剩余）的解法。

**0x70二次剩余**，就是一个二次项  $\%p$  后的剩余。为大家讲解二次同余方程的相关性质解法，在竞赛中常有考察。

**0x80某些非线性丢番图方程**，前面介绍了丢番图方程的概念，并讲解了使用拓展欧几里得算法求解线性丢番图方程的方法以及代码实现。那么对于非线性的丢番图方程呢？一个广为人知的定理表明，并不存在一个适用于所有非线性丢番图方程的通用解法，人们在数百年的研究中得到了一些特殊的非线性丢番图方程的解法，例如著名的毕达哥斯拉三元组，费马大定理，佩尔方程，此部分竞赛涉及较少，但是不排除哪位出题人的兴致盎然，建议简要阅读了解相关定理解法。

**0x90高斯整数**，前面的章节，我们研究的都是整数集合的一些性质，有趣的是其他的一些数集中也存在着一些类似整数的性质。我们将数论拓展到复数集，也就是高斯整数，即形如  $a + bi$  的数。本章为大家讲解了一些数论概念在复数集上的拓展，例如高斯素数，高斯整数上的最大公约数、唯一分解定理，高斯整数环等相关概念，竞赛中略有涉及，建议了解相关概念即可。

**0x100杂项**，收录了一些不知道怎么分类的内容，说不定会有什么用... 例如幂级数的展开式大全，哥德巴赫猜想及其拓展，特征值法求递推式通项公式，预处理  $x$  的次幂  $O(1)$  等等。

为了能使大家更好地运用这些知识，我基本上为每个算法都增加了一个竞赛例题选讲板块，优先选择**算法经典例题、ICPC / CCPC 真题**，并附上了我的详细题解以及部分题目代码，每道题目都给出了相应的题目编号，来源各大OJ，作为大家的课后练习。当然，仅凭这些例题是远远不够的，剩下的就需要大家自己上网检索高质量习题自行练习。

希望大家可以认真阅读本篇文章，真正学会数论，学懂数论，爱上数论，希望本文能在大家的算法竞赛道路上提供帮助，助力大家实现自己的梦想。若真如此，本人荣幸之极。

孟繁宇

2021年1月29日于郑州家中书房

注：本文涵盖了编程竞赛相关的几乎所有数论知识，0x00 ~ 0x30 为基础数论知识，建议一定掌握，0x0x40 ~ 0x72 均为高阶数论知识，稍有难度，建议进阶的同学理解掌握，0x73 ~ 0x100为拓展内容，部分取自《初等数论及其应用》（机械工业出版社，原书第5版），考察频率较低，但还是建议阅读了解，属于选学内容。

## 目录

《算法竞赛中的初等数论》 .....	1
数论全家桶（信奥 / 数竞 / ACM） .....	1
繁凡さん（孟繁宇） .....	1
2021/1/29 .....	1
<a href="https://fanfansann.blog.csdn.net/">https://fanfansann.blog.csdn.net/</a> .....	1
前言 .....	2
0x00 整除 .....	6
0x10 整除相关 .....	6
0x11 素数（质数） .....	6
0x11.1 素数的判定 .....	7
0x11.2 素数的筛法 .....	8
0x11.3 反素数 .....	11
0x12 $\mathbb{Z}^*$ 与 $(\mathbb{Z}^*_p, ^\wedge)$ 结构 .....	12
0x12.1 唯一分解定理（算数基本定理） .....	13
0x12.2 $\mathbb{Z}^*$ 结构中的一些定理 .....	19
0x12.3 $(\mathbb{Z}^*_p, ^\wedge)$ 结构 .....	20
0x13 最大公因数与最小公倍数 .....	20
0x13.1 约数 .....	20
0x13.2 最大公约数 .....	22
0x13.3 最小公倍数 .....	24
0x13.4 GCD 与 LCM 的一些性质与定理 .....	25
0x13.5 补充知识：Fibonacci 数列及其推论 .....	25
0x14 互质与欧拉函数 .....	26
0x14.1 欧拉函数 .....	26
0x14.2 欧拉函数的性质 .....	26
0x15 容斥原理初探 .....	29
0x16 RSA 原理 .....	34
0x20 同余 .....	34
0x21 整数的取余运算 .....	34
0x21.1 整数的取余运算（模运算） .....	34
0x21.2 整数模意义下的加减乘乘方运算 .....	35
0x22 同余 .....	36
0x21.1 同余的性质 .....	37
0x21.2 费马小定理 .....	38
0x21.3 欧拉定理 .....	38
0x21.4 威尔逊定理 .....	40
0x22 拓展欧几里德 .....	43
0x22.1 裴蜀（Bézout）定理 .....	43
0x22.2 扩展欧几里德算法 .....	45
0x22.3 解二元模线性方程 .....	45
0x22.4 类欧几里德算法（一个求和技巧） .....	49
0x22.5 整式方程 .....	50
0x23 乘法逆元 .....	51
0x23.1 乘法逆元定义与性质 .....	51
0x23.2 费马小定理求乘法逆元 .....	51
0x23.3 扩展欧几里德求乘法逆元 .....	51
0x23.4 线性递推求乘法逆元 .....	51
0x23.5 求阶乘的逆元 .....	52
0x24 线性同余方程 .....	52
0x24.1 同余方程 .....	52
0x24.2 中国剩余定理 .....	54
0x24.3 拓展中国剩余定理 .....	55
0x25 高次同余方程（一） .....	62
0x25.1 BSGS 算法 .....	63
0x25.2 拓展BSGS算法 .....	64
0x26 【题型探究】公约数之和 .....	68
0x26.1 母题：UVA11417 GCD .....	68

0x26.2 拓展一、UVA11426 GCD - Extreme (II)	69
0x26.3 扩展二、luogu P2398 GCD SUM	70
0x26.4 扩展三、luogu P2568 GCD	72
<b>0x30 积性函数</b>	<b>73</b>
0x31 常见积性函数	74
0x32 莫比乌斯函数	74
0x33 狄利克雷卷积	76
0x33.1 常见积性函数的卷积	77
0x33.2 $O(n\log n)$ 的卷积预处理	78
<b>0x40 反演</b>	<b>81</b>
0x41 整除分块	81
0x41.1 前置知识	81
0x41.2 整除分块	82
0x42 莫比乌斯反演	87
0x42.1 莫比乌斯反演公式	87
0x42.2 莫比乌斯反演证明	87
0x42.3 推导结论	88
0x42.4 竞赛例题选讲	89
0x42.5 莫比乌斯反演扩展	106
0x43 欧拉反演	107
0x44 二项式反演	111
0x45 斯特林反演	114
0x46 单位根反演	115
0x47 子集反演	116
0x48 最值反演 (Min - Max 容斥)	119
0x49 拉格朗日反演	122
0x4A 反演常用技巧	123
0x4B. Dirichlet 前缀和	124
0x4B.1 Dirichlet 前缀和	124
0x4B.2 Dirichlet 后缀和	125
0x4B.2 倒推 Dirichlet 前缀和	125
0x4B.3 倒推 Dirichlet 后缀和	125
<b>0x50 筛法</b>	<b>125</b>
0x51 线性筛法	125
0x51.1 线性筛法求欧拉函数	126
0x51.2 线性筛求莫比乌斯函数	126
0x51.3 线性筛求约数个数函数	127
0x51.4 线性筛求约数和函数	127
0x52 杜教筛	128
0x52.1 杜教筛	128
0x52.2 求欧拉函数前缀和	130
0x52.3 求莫比乌斯函数前缀和	131
0x53 Min_25筛	143
0x54 洲阁筛	146
<b>0x60 原根</b>	<b>148</b>
0x61 整数的阶、原根与指标	148
0x61.1 整数的阶	148
0x61.2 整数的原根	148
0x61.3 整数的指标 (也称指数、离散对数)	151
0x62 素数的原根	151
0x62.1 多项式同余	151
0x63 原根的存在性	151
0x64 原根的应用	152
0x64.1 乘法换加法 (取模意义下)	152
0x64.2 快速数论变换	154
0x65 高次同余方程 (二) ( $N$ 次剩余)	154
<b>0x70 二次剩余</b>	<b>159</b>
0x71 二次剩余与二次非剩余	159
0x72 Cipolla 算法解二次同余方程	159



**0x80 某些非线性丢番图方程 .....160**

    0x81 毕达哥斯拉三元组（勾股数） .....160

    0x82 费马大定理.....161

    0x83 平方和.....161

    0x84 佩尔方程与连分数.....162

        0x84.1 佩尔方程与连分数.....162

        0x84.2 解佩尔方程.....162

        0x84.3 竞赛例题选讲.....164

**0x90 高斯整数.....164**

    0x90.0 复数 .....164

    0x91 高斯整数 .....165

    0x92 高斯素数 .....166

    0x93 唯一分解 .....166

    0x94 最大公约数.....166

    0x95 同余和剩余系 .....166

    0x96 费马二平方和定理.....167

    0x97 分解 $4k+1$ 型素数.....167

    0x98 构造  $a^2+b^2=n$ 的方案 .....167

    0x99 竞赛例题选讲 .....167

**0x100 杂项.....169**

    0x101 哥德巴赫猜想及其拓展.....169

    0x102 幂级数展开式常用公式.....170

    0x103 各进制整数的位数 .....170

    0x104 判断组合数的奇偶性 .....170

    0x105 特征根法求数列通项公式 .....170

**0x110 后记.....173**

## 0x00 整除

数论的旅程，从整除开始。整除这个基本概念将贯穿数论始末，理解并掌握它是基础中的基础。

- 整除

**定义：**若整数  $n$  除以整数  $d$  的余数为 0，即  $d$  能整除  $n$ ，则称  $d$  是  $n$  的约数， $n$  是  $d$  的倍数，记为  $d \mid n$ 。

- 整除的性质

**性质00.1.1：**  $a \mid b, b \mid c \Rightarrow a \mid c$

**性质00.1.2：**  $a \mid b \Rightarrow a \mid bc$ ， $c$  为任意的整数。

**性质00.1.3：**  $a \mid b, a \mid c \Rightarrow a \mid kb \pm lc$ ， $k$  与  $l$  均为任意的整数。（都有公因子  $a$ ，正确性显然）

**拓展：**  $k_1, k_2$  互质，则  $k_1 + k_2$  与  $k_1 \times k_2$  互质。（仅有  $a = 1$  能整除  $k_1, k_2$ ，故仅有  $a = 1$  能同时整除  $k_1 + k_2$  与  $k_1 \times k_2$ ）

**性质00.1.4：**  $a \mid b, b \mid a \Rightarrow a = \pm b$

**性质00.1.5：**  $a = kb \pm c \Rightarrow a, b$  的公因数与  $b, c$  的公因数完全相同

**性质00.1.6：** 若  $a \mid bc$ ，且  $a$  与  $c$  互质，则  $a \mid b$

性质 00.1.1 ~ 00.1.4 的正确性可由定义得到，正确性显然，这里仅给出性质00.1.5的证明。**性质 00.1.6** 涉及到互质的概念，具体性质与概念详见本文 **0x14 互质与欧拉函数**。

**性质00.1.5的证明：**

利用**性质00.1.3**， $(a \mid b, a \mid c \Rightarrow a \mid kb \pm lc)$ ，对于任意的  $a, b$  的公因数  $d$ ： $a = kb \pm c \Rightarrow c = \pm(a - kb) \Rightarrow d \mid c$

这里给出一个1987年的初中数学联赛真题帮助大家理解整除的相关概念与性质。

- 1987年全国初中数学联赛

$x, y, z$  均为整数，若  $11 \mid (7x + 2y - 5z)$ ，求证： $11 \mid (3x - 7y + 12z)$ 。

**Solution**

非常经典的一个问题，令  $a = (7x + 2y - 5z)$ ， $b = (3x - 7y + 12z)$ ，通过构造可以得到一个等式： $3a + 4b = 11(3x - 2y + 3z)$ ，则  $4b = 11(3x - 2y + 3z) - 3a$ 。

**性质00.1.2** + **性质00.1.3**，得出  $11 \mid (11(3x - 2y + 3z) - 4a) = 11 \mid 3b$ 。

**性质00.1.6：**，由于 11 和 3 互素，得出  $11 \mid b$ ，证明完毕。

## 0x10 整除相关

在了解完整除的相关性质以后，我们将进入数论的第一章，最重要的章节之一，也是数论千百年研究突破的重点，整除相关：素数，约数。

### 0x11 素数（质数）

**素数定义**

**素数（又称质数）是只有 1 和它本身两个因数的数。** 规定 1 既非素数也非合数。特殊的，2 是唯一的偶素数。

**素数定理**

我们设  $\pi(x)$  为 1 到  $x$  中素数的个数。

其中：

$$\pi(n) = -1 + \sum_{k=1}^x \lfloor \cos^2 \left[ \pi \frac{(n-1)! + 1}{n} \right] \rfloor$$

上述公式是由 黎曼 给出的精确公式，详细证明参见：<https://mathworld.wolfram.com/RiemannPrimeCountingFunction.html>

尝试求极限发现（其中  $\ln(x)$  表示  $x$  的自然对数）：

$$\lim_{x \rightarrow \infty} \frac{\pi(x)}{\left(\frac{x}{\ln(x)}\right)} = 1$$

即可得到定理：

**定理11.0.1：** 在自然数集中，小于  $n$  的质数约有  $\frac{n}{\ln(n)}$  个。

由该定理可知，是 `int` 范围内的素数的个数并不会很多，其中 `int` 范围内的素数间距大概是  $10^2$  的数量级。

**定理11.0.2：**（伯特兰 — 切比雪夫定理）若整数  $n > 3$ ，则至少存在一个质数  $p$ ，符合  $n < p < 2n - 2$ 。另一个稍弱说法是：对于所有大于 1 的整数  $n$ ，至少存在一个质数  $p$ ，符合  $n < p < 2n$ 。

### 0x11.1 素数的判定

既然知道了什么是素数，我们肯定要考虑如何判定一个数是素数，我们往往直接从定义角度出发来判定是否为素数。

接下来介绍的四种算法均为单个素数的判定方法。

- **试除法**

试除法是最常用的判断素数的方法。

一个数如果不是素数，则一定能被一个小于它自己的数整除。假设一个数能整除  $n$ ，即  $a \mid n$ ，那么  $\frac{n}{a}$  也必定能整除  $n$ ，不妨设  $a \leq \frac{n}{a}$ ，则有  $a^2 \leq n$ ，即  $a \leq \sqrt{n}$ 。

时间复杂度： $O(\sqrt{n})$

```
1 inline bool is_prime(int x){
2     if(x < 2)
3         return false;
4     for(register int i = 2; i * i <= x; ++ i)
5         if(x % i == 0)
6             return false;
7     return true;
8 }
```

如果想要追求更高的效率，可以考虑使用  $kn + i$  法

- $kn + i$  法

一个大于 1 的整数如果不是素数，那么一定有素因子，因此在枚举因子时只需要考虑可能为素数的因子即可。 $kn + i$  法即枚举形如  $kn + i$  的数，例如取  $k = 6$ ，那么  $6n + 2, 6n + 3, 6n + 4, 6n + 6$  都不可能为素数（显然它们分别有因子 2, 3, 2, 6 一定不是素数），因此我们只需要枚举形如  $6n + 1, 6n + 5$  的数即可，这样整体的时间复杂度就会降低了  $\frac{2}{3}$ ，也就是  $O(n^{\frac{1}{3}})$ 。

下面是  $kn + i$  法  $k = 30$  版本的模板：

```
1 bool isPrime(ll n){
2     if(n == 2 || n == 3 || n == 5) return 1;
3     if(n % 2 == 0 || n % 3 == 0 || n % 5 == 0 || n == 1) return 0;
4     ll c = 7, a[8] = {4,2,4,2,4,6,2,6};
5     while(c * c <= n) for(auto i : a){if(n % c == 0) return 0; c += i;}
6     return 1;
7 }
```

- **预处理法**

对于多组数据，如果  $n$  是合数，那么它必然有一个小于等于  $\sqrt{n}$  的素因子，只需要对  $\sqrt{n}$  内的素数进行测试即可，也就是预处理求出  $\sqrt{n}$  中的素数，假设该范围内素数的个数为  $s$ （ $s = \frac{n}{\ln n}$ ），那么时间复杂度为  $O(\frac{n}{\ln n})$ 。

- **Miller — Rabin 判定法**

对于一个很大的数  $n$ （例如十进制表示有 100 位），如果还是采用试除法进行判定，时间复杂度必定难以承受，目前比较稳定的大素数测试算法是米勒-拉宾 (Miller — Rabin) 素数测试算法，该素数测试算法可以通过控制迭代次数来间接控制正确率。

Miller — Rabin 判定法是基于费马小定理的，即如果一个数  $p$  为素数的条件是对于所有和  $p$  互素的正整数  $a$  满足以下等式： $a^{p-1} \equiv 1 \pmod{p}$ 。（费马小定理的相关概念定理性质详见 [本文 0x21.2 费马小定理](#)）

然而我们不可能试遍所有和  $p$  互素的正整数，这样的话复杂度反而更高，事实上我们只需要取比  $p$  小的几个素数进行测试就行了。

具体判断  $n$  是否为素数的算法如下：

1. 如果 `n==2` , 返回 `true` ; 如果 `n<2 || !(n & 1)` , 返回 `false` ; 否则跳到 ii) 。
2. 令 `n = m *(2 ^ k) + 1` , 其中  $m$  为奇数, 则 `n - 1 = m * (2 ^ k)` 。
3. 枚举小于  $n$  的素数  $p$  (至多枚举 10 个) , 对每个素数执行费马测试, 费马测试如下: 计算 `pre = p ^ m % n` , 如果  $pre$  等于 1, 则该测试失效, 继续回到 iii) 测试下一个素数; 否则进行  $k$  次计算 `next = pre ^ 2 % n` , 如果 `next == 1 && pre != 1 && pre != n-1` 则  $n$  必定是合数, 直接返回;  $k$  次计算结束判断  $pre$  的值, 如果不等于 1, 必定是合数。
4. 10次判定完毕, 如果  $n$  都没有检测出是合数, 那么  $n$  为素数。

时间复杂度为  $O(k \log n)$

Code

```
1  ll Rand() { //决定了程序的性能
2      static ll x = (srand((int)time(0)), rand());
3      x += 1000003;
4      if (x > 1000000007)
5          x -= 1000000007;
6
7      return x;
8  }
9  bool Witness(ll a, ll n) {
10     ll t = 0, u = n - 1;
11     while (!(u & 1))
12         u >>= 1, t++;
13     ll x = qpow(a, u, n), y; //qpow为快速幂
14
15     while (t--) {
16         y = x * x % n;
17         if (y == 1 && x != 1 && x != n - 1)
18             return true;
19         x = y;
20     }
21     return x != 1;
22 }
23 bool MillerRabin(ll n, ll s) {
24     if (n == 2 || n == 3 || n == 5)
25         return 1;
26
27     if (n % 2 == 0 || n % 3 == 0 || n % 5 == 0 || n == 1)
28         return 0;
29
30     while (s--) {
31         if (Witness(Rand() % (n - 1) + 1, n))
32             return false;
33     }
34     return true;
35 }
```

有了大素数测试算法, 那么就会有大数据的质因子分解法, 详见本文 **0x12.1唯一分解定理**

0x11.2 素数的筛法

通过上文的学习, 我们了解了判断单个数是否为素数的四种常用方法, 那么对于一个很大范围内的所有数的素数判定, 若我们直接对于每一个数都使用一次素数判定法, 如此庞大的时间复杂度我们是无法承担的, 因此引入了对于大规模数的素数判定方法: **质数筛法**。

质数筛法一般分为**埃氏筛**和**线性筛**。

埃氏筛没有线性筛时间复杂度好, 不常用, 但是他的**时间复杂度分析方法**却比较常用。

首先我们来证明一下  $O(\sum_{i=1}^n \frac{1}{i}) \approx O(\log n)$

显然有:

$$\begin{aligned} \sum_{i=1}^n \frac{1}{i} &= 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \frac{1}{6} + \frac{1}{7} + \cdots \\ &= 1 + (\frac{1}{2} + \frac{1}{3}) + (\frac{1}{4} + \frac{1}{5} + \frac{1}{6} + \frac{1}{7}) + \cdots \end{aligned}$$

显然有

$$\sum_{j=i}^{i+k-1} \frac{1}{j} \leq k \times \frac{1}{i}$$

即:  $\frac{1}{2} + \frac{1}{3} \leq \frac{1}{2} \times 2 \leq 1, \frac{1}{4} + \frac{1}{5} + \frac{1}{6} + \frac{1}{7} \leq \frac{1}{4} \times 4 \leq 1$

即：

$$\sum_{j=2^{i-1}}^{2^i} \frac{1}{j} \leq 2^{i-1} \times \frac{1}{j} = 1$$

所以  $1 \sim n$  就被分为了  $\log n$  组，每组的和小于等于 1，故  $\sum_{i=1}^n \frac{1}{i} \approx \log n$

● Eratosthenes 筛法 （埃拉托色尼筛法）

显然如果  $x$  是合数，那么  $x$  的倍数也一定是合数。利用这个结论，我们可以避免很多次不必要的检测。

我们可以从小到大枚举分析每一个数，然后同时把当前这个数的所有（比自己大的）倍数记为合数，那么运行结束的时候没有被标记的数就是素数了。

```
1 int v[N];
2 void primes(int n{
3     memset(v, 0, sizeof v);
4     for(int i = 2;i <= n; ++ i){
5         if(v[i])continue;
6         cout << i << endl;
7         for(int j = i;j <= n / i; ++ j)
8             v[i * j] = 1;
9     }
10 }
```

埃氏筛的时间复杂度为  $O(n \log \log n) \approx O(n)$ （这里的log以10为底），因为我们这里外层循环  $O(n)$ ，内层循环上界为  $\frac{n}{i}$ ，随着  $i$  的增加， $\frac{n}{i} \in \{n, \frac{n}{2}, \frac{n}{3}, \frac{n}{4}, \frac{n}{5} \dots, \frac{n}{n}\}$ ，而调和级数  $f(n) = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} \dots + \frac{1}{n}$  当  $n$  趋近于  $\infty$  时，调和级数极限为  $\ln n + C$  其中  $C$  为欧拉常数  $C \approx 0.577218$  当我们使用优化筛掉合数以后，这个调和级数就变成  $f(n) = \frac{1}{2} + \frac{1}{3} + \frac{1}{5} + \frac{1}{7} \dots + \frac{1}{n} \approx \log \log n$ ，所以整体的算法时间复杂度为  $O(n \log \log n)$ 。

这个**利用调和级数**计算时间复杂度的方法很常用。

更严谨的证明：

如果每一次对数组的操作花费 1 个单位时间，则时间复杂度为：

$$O\left(n \sum_{k=1}^{\pi(n)} \frac{1}{p_k}\right)$$

其中  $p_k$  表示第  $k$  小的素数。根据 Mertens 第二定理，存在常数  $B_1$  使得：

$$\sum_{k=1}^{\pi(n)} \frac{1}{p_k} = \log \log n + B_1 + O\left(\frac{1}{\log n}\right)$$

所以 **Eratosthenes 筛法** 的时间复杂度为  $O(n \log \log n)$ 。接下来我们证明 Mertens 第二定理的弱化版本  $\sum_{k \leq \pi(n)} 1/p_k = O(\log \log n)$ ：

根据  $\pi(n) = \Theta(n/\log n)$ ，可知第  $n$  个素数的大小为  $\Theta(n \log n)$ 。于是就有

$$\begin{aligned} \sum_{k=1}^{\pi(n)} \frac{1}{p_k} &= O\left(\sum_{k=2}^{\pi(n)} \frac{1}{k \log k}\right) \\ &= O\left(\int_2^{\pi(n)} \frac{dx}{x \log x}\right) \\ &= O(\log \log \pi(n)) = O(\log \log n) \end{aligned}$$

● 线性筛（欧拉筛）

在欧拉筛我们可以保证每个数一定只会被它的最小质因子筛掉一次。

由于 `primes` 数组中的质数是递增的。

我们从小到大枚举 `primes` 数组，当第一次枚举到一个质数 `primes[j]` 满足 `primes[j] | i` 时，`primes[j]` 一定是  $i$  的最小质因子，`primes[j]` 也一定是  $i \times \text{primes[j]}$  的最小质因子，而接下来的  $i \times \text{primes[j + 1]}$  的最小质因子应该是 `primes[j]` 而不是 `primes[j + 1]`，故此时直接 `break` 即可。

那么对于任意一个合数  $x$ ，假设  $x$  的最小质因子为  $y$ ，那么当枚举到  $\frac{x}{y} < x$  的时候一定会把  $x$  筛掉，即在枚举到  $x$  之前一定能把合数  $x$  筛掉，所以一定能把所有的合数都筛掉。

由于 **保证每个合数只都被自己的最小质因子筛掉一遍**，所以时间复杂度是  $O(n)$  的。（注意到筛法求素数的同时也得到了每个数的最小质因子，这是后面筛法求欧拉函数的关键）

## Code

```
1  const int N = 10005;
2  int n, primes[N], cnt;
3  bool vis[N];
4  inline void get_prime()
5  {
6      for(register int i = 2; i <= n; i++) {
7          if(!vis[i]) primes[ ++ cnt] = i;
8          for(register int j = 1; j <= cnt && i * primes[j] <= n; ++ j) {
9              vis[i * primes[j]] = 1;
10             if(i % primes[j] == 0) break;
11         }
12     }
13 }
```

## ● 竞赛例题选讲

### Problem A 夏洛克和他的女朋友 (AcWing 1293)

有  $n$  件珠宝，第  $i$  件的价值是  $i + 1$ ，也就是说，珠宝的价值分别为  $2, 3, \dots, n + 1$ 。请你来为这些珠宝染色，使得一件珠宝的价格是另一件珠宝的价格的质因子时，两件珠宝的颜色不同。求使用的颜色数的最小值以及染色方案。

## Solution

因为一个数不能与自己的**质因子**同色，所以所有的质数可以同色。

又因为所有质因数已被染色，所以就不需要对其他数染色了。即所有质数染色为 1，所有合数染色为 2 即可。所以我们只需要找出  $[2, n + 1]$  的所有质数输出即可。注意特殊情况，不存在合数的时候，仅输出 1 即可。

### Problem B 质数距离 (AcWing 196)

给定两个整数  $L$  和  $U$ ，你需要在闭区间  $[L, U]$  内找到距离最接近的两个相邻质数  $C_1$  和  $C_2$ （即  $C_2 - C_1$  是最小的），如果存在相同距离的其他相邻质数对，则输出第一对。

同时，你还需要找到距离最远的两个相邻质数  $D_1$  和  $D_2$ （即  $D_1 - D_2$  是最大的），如果存在相同距离的其他相邻质数对，则输出第一对。

$$1 \leq L \leq U \leq 2^{31} - 1, R - L \leq 10^6$$

## Solution

看数据范围做题。如果小的话我们可以直接枚举  $O(n)$  即可。但是这里数据范围达到了  $2^{31}$ ，就算用线性筛也存不下这么大的数。

尽管数据范围大到不可做，我们发现区间  $[L, R]$  的差值不会超过  $10^6$ ，所以我们需要改进线性筛法，得到一个可以求得尽管数据很大但是实际区间长度不大的质数筛法，即**二次筛法**。

首先显然任何一个合数  $n$  必定包含一个不超过  $\sqrt{n}$  的**因子**  $x$ ，该因子  $x$  一定有一个小于  $x$  的质因子或者说  $x$  就是一个质数。

这样我们只需要求出  $2 \sim \sqrt{n}$  的质数  $p$ ，也就是 50000 左右，对于每一个  $L \sim R$  中的数，能被  $p$  整除的一定是合数，剩下的就是质数的，再对剩下的质数大概只有两两相比较就行了。

## 步骤

- 找出  $1 \sim \sqrt{2^{31} - 1}$  ( $< 50000$ ) 中的所有质因子
- 对于  $1 \sim 50000$  中每个质数  $P$ ，将  $[L, R]$  中所有  $P$  的倍数筛掉(至少2倍)
- 找到大于等于  $L$  的最小的  $P$  的倍数  $P_0$ ，显然  $P_0 = \max\{2 \times P, \left\lceil \frac{L}{P} \right\rceil \times P\}$ ，找下一个倍数时只需要  $+= P$  即可。

## 引理(分数的上取整转换下取整)

$$\left\lceil \frac{L}{P} \right\rceil = \left\lfloor \frac{L + P - 1}{P} \right\rfloor$$

## Code

```
1  typedef long long LL;
2  const int N = 1000010;
3  int primes[N], cnt;
4  bool st[N];
5  void get_primes(int n); // 线性筛代码略去
6  int main()
7  {
8      int l, r;
9      while (cin >> l >> r) {
10         get_primes(50000);
11         memset(st, 0, sizeof st);
12         for (int i = 0; i < cnt; i++) {
13             LL p = primes[i];
14             // j初始化为KP, 使得KP>L, 即找到最少P需要多少倍可以大于等于左边界L, 并且至少需要是二倍。
15             for (LL j = max(p * 2, (l + p - 1) / p * p); j <= r; j += p)
```



```
16         st[j - 1] = true;
17     }
18     cnt = 0;
19     for (int i = 0; i <= r - 1; i ++ )
20         if (!st[i] && i + 1 >= 2)//特判，因为P可能为1
21             primes[cnt ++ ] = i + 1;
22
23     if (cnt < 2) puts("There are no adjacent primes.");
24     else {
25         int minp = 0, maxp = 0;
26         for (int i = 0; i + 1 < cnt; i ++ ) { //筛完之后暴力跑
27             int d = primes[i + 1] - primes[i];
28             if (d < primes[minp + 1] - primes[minp]) minp = i;
29             if (d > primes[maxp + 1] - primes[maxp]) maxp = i;
30         }
31
32         printf("%d,%d are closest, %d,%d are most distant.\n",
33             primes[minp], primes[minp + 1],
34             primes[maxp], primes[maxp + 1]);
35     }
36 }
37 return 0;
38 }
```

11.3 反素数

学完素数以后，最后我们再来介绍一下反素数。（建议在学完约数的相关概念以后再来学习本小节）

反素数定义

如果某个正整数  $n$  满足如下条件，则称为是反素数：任何小于  $n$  的数的正约数个数都小于  $n$  的正约数个数，即  $n$  是  $1 \dots n$  中正约数个数最多的数。

素数就是因子只有两个的数，那么反素数，就是**因子最多的数**（并且**因子个数相同的时候值最小**），所以反素数是相对于一个集合来说的，也就是在一个集合中，因素最多并且值最小的数，就是反素数。

既然要求因子数，我们首先想到的就是素因子分解。把  $n$  分解成  $n = p_1^{k_1} p_2^{k_2} \cdots p_n^{k_n}$  的形式，其中  $p$  是素数， $k$  是他的指数。这样的话总因子个数就是  $(k_1 + 1) \times (k_2 + 1) \times (k_3 + 1) \cdots \times (k_n + 1)$ 。

但是对于每一个数都质因数分解的代价太大，总时间复杂度  $O(n\sqrt{n})$ ，并且每个数都相对独立，比较浪费。

我们考虑一些反素数的性质：

若  $N \leq 2^{31}$

**引理11.3.1**：  $1 \sim N$  中的反素数，就是  $1 \sim N$  中约数个数最多的数中**最小**的一个。

**引理11.3.2**：  $1 \sim N$  中任何数的不同质因子都不会超过 10 个且所有质因子的质数都不会超过 30。

**引理11.3.3**：  $\forall x \in [1, N]$ ， $x$  为反素数的必要条件是： $x$  分解质因数后可以写成  $2^{c_1} \times 3^{c_2} \times 5^{c_3} \times 7^{c_4} \times 11^{c_5} \times 13^{c_6} \times 17^{c_7} \times 19^{c_8} \times 23^{c_9} \times 29^{c_{10}}$ ，且  $c_1 \geq c_2 \geq c_3 \geq \cdots \geq c_{10} \geq 0$ ，换句话说， $x$  的质因子是连续的若干个最小的质数，并指数单调递减。

这三个引理非常显然，应该不需要证明。

根据上面的三个引理，我们可以直接DFS，一次确认前 10 个质数的指数，并满足指数单调递减，总成绩不超过  $N$ ，同时记录约数的个数即可。最后利用 **引理 11.3.1** 找到约数个数最多的数里最小的那个数即可。

我们可以把当前走到每一个素数前面的时候列举成一棵树的根节点，然后一层层的去找。找到什么时候停止呢？

- 1. 当前走到的数字已经大于我们想要的数字了
- 2. 当前枚举的因子已经用不到了
- 3. 当前因子大于我们想要的因子了
- 4. 当前因子正好是我们想要的因子（此时判断是否需要更新最小  $ans$ ）

然后 dfs 里面不断一层一层枚举次数继续往下迭代

竞赛例题选讲

Problem A Number With The Given Amount Of Divisors (CF27E)

给定一个正整数  $n$ ，输出最小的整数，满足这个整数有  $n$  个因子，即求因子数一定的最小反素数。

Solution

对于这种题，我们只要以因子数为 dfs 的返回条件基准，不断更新找到的最小值就可以了

Code

```
1 #include <stdio.h>
```

```
2 #define ULL unsigned long long
3 #define INF ~0ULL
4 ULL p[16] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53};
5 ULL ans;
6 ULL n;
7 // depth: 当前在枚举第几个素数。num: 当前因子数。
8 // temp: 当前因子数量为 num
9 // 的时候的数值。up: 上一个素数的幂，这次应该小于等于这个幂次嘛
10 void dfs(ULL depth, ULL temp, ULL num, ULL up) {
11     if (num > n || depth >= 16) return;
12     if (num == n && ans > temp) {
13         ans = temp;
14         return;
15     }
16     for (int i = 1; i <= up; i++) {
17         if (temp / p[depth] > ans) break;
18         dfs(depth + 1, temp = temp * p[depth], num * (i + 1), i);
19     }
20 }
21 int main() {
22     while (scanf("%llu", &n) != EOF) {
23         ans = INF;
24         dfs(0, 1, 1, 64);
25         printf("%llu\n", ans);
26     }
27     return 0;
28 }
```

**Problem B More Divisors (ZOJ 1562)**

求  $n$  以内因子数最多的数

**Solution**

思就是求  $1 \sim n$  内的反素数嘛，改改 `dfs` 的返回条件就行了。注意题目的数据范围，用 `int` 会溢出，在循环里可能就出不来了就超时了。

**Code**

```
1 #include <cstdio>
2 #include <iostream>
3 #define ULL unsigned long long
4 int p[16] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53};
5 ULL n;
6 ULL ans, ans_num; // ans 为 n 以内的最大反素数（会持续更新），ans_sum 为 ans
7 // 的因子数。
8 void dfs(int depth, ULL temp, ULL num, int up) {
9     if (depth >= 16 || temp > n) return;
10    if (num > ans_num) {
11        ans = temp;
12        ans_num = num;
13    }
14    if (num == ans_num && ans > temp) ans = temp;
15    for (int i = 1; i <= up; i++) {
16        if (temp * p[depth] > n) break;
17        dfs(depth + 1, temp *= p[depth], num * (i + 1), i);
18    }
19    return;
20 }
21 int main() {
22     while (scanf("%llu", &n) != EOF) {
23         ans_num = 0;
24         dfs(0, 1, 1, 60);
25         printf("%llu\n", ans);
26     }
27     return 0;
28 }
```

**0x12  $Z^*$  与  $(Z_p^*, \cdot)$  结构**

$Z^*$ ，即正整数集。

$(Z_p^*, \cdot)$ ， $Z_p$  的剩余类群，即  $(1, 2, \dots, p - 1)$ ，与  $p$  互素的数，即  $U(Z_p)$

0x12.1 唯一分解定理（算数基本定理）

任何一个大于 1 的数都可以被分解成有限个质数乘积的形式

$$n = \prod_{i=1}^m p_i^{C_i} = p_1^{C_1} \times p_2^{C_2} \times \cdots \times p_n^{C_n}$$

其中  $p_1 < p_2 < \cdots < p_m$  为质数,  $C_i$  为正整数。

- 唯一分解定理证明：
  - (1) 大于1的自然数必可写成质数之积

使用反证法：假设存在大于1的自然数不能写成质数的乘积，把最小的那个称为  $n$ 。

自然数可以根据其可除性（是否能表示成两个不是自身的自然数的乘积）分成3类：质数、合数和1。首先，按照定义， $n$  大于1。其次， $n$  不是质数，因为质数  $p$  可以写成质数乘积： $p = p$ ，这与假设不相符合。因此 $n$ 只能是合数，但每个合数都可以分解成两个严格小于自身而大于1的自然数的积。设  $n = a \times b$ ，其中  $a$  和  $b$  都是介于 1 和  $n$  之间的自然数，因此，按照  $n$  的定义， $a$  和  $b$  都都可以写成质数的乘积。从而  $n = a \times b$  也可以写成质数的乘积。由此产生矛盾。因此大于 1 的自然数必可写成质数的乘积。

- (2) 证明质数分解唯一性

同样使用反证法：设存在一个能分解为两种根本不同的质数乘积的正整数，则其中必有一最小的（最小数原理），设  $m = p_1 p_2 \cdots p_r = q_1 q_2 \cdots q_s$ ，从小到大排序。 $p_1$  不等于  $q_1$ ，否则可消去，不满足最小假设。设  $p_1 < q_1$ 。构造一整数

$$m' = m - (p_1 q_2 \cdots q_s) = (p_1 p_2 \cdots p_r) - (p_1 q_2 \cdots q_s) = p_1 (p_2 \cdots p_r - q_2 \cdots q_s) = (q_1 q_2 \cdots q_s) - (p_1 q_2 \cdots q_s) = (q_1 - p_1) q_2 \cdots q_s$$

由于  $p_1 < q_1$ ， $m'$ 是一个正整数，且 $< m$ 。因此  $m'$  的质数分解，除了因子次序外，必须是唯一的（ $m$  是最小数假设）。从上面表达式可看出， $p_1$ 是  $m'$  的因子，再根据  $m'$  质数分解的唯一性得出（推论、用反证法很容易得到）， $p_1$ 必须是  $(q_1 - p_1)$  或者  $q_2 \cdots q_s$  的因子，由于  $q$ 都比  $p_1$  大，这后一情形是不可能的，这样就有某个整数  $h$ ，使  $q_1 - p_1 = p_1 * h$  或  $q_1 = p_1(h + 1)$ 。这表明  $p_1$  是  $q_1$  的一个因子，与  $q_1$  是质数相矛盾。故得证。

显然  $n$  最多仅有一个大于  $\sqrt{n}$  的质因子（若有两个的话，他们的乘积就大于  $n$  了）。

- 试除法

类似埃式筛，我们直接枚举因子然后把当前因子全部除尽即可，时间复杂度  $O(\sqrt{n})$ 。

```
1 int c[N],p[N];
2 void divide(int n) {
3     cnt = 0;
4     for(int i = 2; i * i <= n; ++ i) {
5         if(n % i == 0) {
6             p[ ++ cnt] = i,c[cnt] = 0;
7             while(n % i == 0) n /= i,c[cnt] ++ ;
8         }
9     }
10    if(n > 1)//如果n是质数
11        p[ ++ cnt] = n,c[cnt] = 1;
12    for(int i = 1;i <= cnt; ++ i)
13        cout << p[i] << "^" << c[i] << endl;
14 }
```

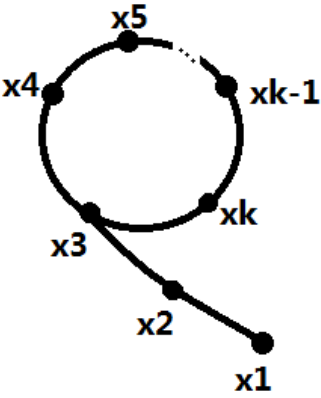
- Pollard Rho 算法

对于数据较大的情况，如  $n \geq 10^{18}$ ，有用来分解其因数的 Pollard Rho 算法。

Pollard-rho 算法是一个大数分解的随机算法，能够在  $O(n^{\frac{1}{4}})$  的时间内找到  $n$  的一个素因子  $p$ ，然后再递归计算  $n' = \frac{n}{p}$ ，直到  $n$  为素数为止，通过这样的方法将  $n$  进行素因子分解。

Pollard-rho 的策略为：从  $[2, n)$  中随机选取  $k$  个数 $x_1、x_2、x_3、\dots、x_k$ ，求任意两个数 $x_i、x_j$ 的差和  $n$  的最大公约数，即  $d = \gcd(x_i - x_j, n)$ ，如果  $1 < d < n$ ，则  $d$  为  $n$  的一个因子，直接返回  $d$  即可。

然后来看如何选取这  $k$  个数，我们采用生成函数法，令  $x_1 = rand() \% (n - 1) + 1$ ， $x_i = (x_i - 1^2 + 1) \mod n$ ，很明显，这个序列是有循环节的，如图所示：



我们需要做的就是它在进入循环的时候及时跳出循环，因为  $x_1$  是随机选的， $x_1$  选的不好可能使得这个算法永远都找不到  $n$  的一个范围在  $(1, n)$  的因子，这里采用步进法，保证在进入环的时候直接跳出循环。

【模板】Pollard-Rho算法（Luogu P4718）

T组数据，每组数据输入一个数  $n$ ，检验  $n$  是否是质数，是质数就输出 Prime。如果不是质数，输出它的最大质因子。

$$1 \leq T \leq 350, 1 \leq n \leq 10^{18}$$

## Solution

模板题，大数质因数分解后，比较每一个分解出来的质因子，找到最大质因子即可。

```

1  typedef long long ll;
2  const int N = 1e5 + 7;
3  ll x, y, a[N];
4  ll max_factor;
5  struct BigIntegerFactor {
6      const static int N = 1e6 + 7;
7      ll prime[N], p[N], fac[N], sz, cnt; //多组输入注意初始化cnt = 0
8      inline ll mul(ll a, ll b, ll mod) {          //WA了尝试改为__int128或慢速乘
9          if (mod <= 1000000000)
10             return a * b % mod;
11             return (a * b - (ll)((long double)a / mod * b + 1e-8) * mod + mod) % mod;
12     }
13     void init(int maxn) {
14         int tot = 0;
15         sz = maxn - 1;
16         for (int i = 1; i <= sz; ++i)
17             p[i] = i;
18         for (int i = 2; i <= sz; ++i) {
19             if (p[i] == i)
20                 prime[tot++] = i;
21             for (int j = 0; j < tot && 1ll * i * prime[j] <= sz; ++j) {
22                 p[i * prime[j]] = prime[j];
23                 if (i % prime[j] == 0)
24                     break;
25             }
26         }
27     }
28     ll qpow(ll a, ll x, ll mod) {
29         ll res = 1ll;
30         while (x) {
31             if (x & 1)
32                 res = mul(res, a, mod);
33             a = mul(a, a, mod);
34             x >>= 1;
35         }
36         return res;
37     }
38     bool check(ll a, ll n) {          //二次探测原理检验n
39         ll t = 0, u = n - 1;
40         while (!(u & 1))
41             t++, u >>= 1;
42         ll x = qpow(a, u, n), xx = 0;
43         while (t--) {
44             xx = mul(x, x, n);
45             if (xx == 1 && x != 1 && x != n - 1)
46                 return false;
47             x = xx;
48         }
49         return xx == 1;
50     }
51     bool miller(ll n, int k) {
52         if (n == 2)
53             return true;
54         if (n < 2 || !(n & 1))
55             return false;
56         if (n <= sz)
57             return p[n] == n;
58         for (int i = 0; i <= k; ++i) {          //测试k次
59             if (!check(rand() % (n - 1) + 1, n))
60                 return false;
61         }
62         return true;
63     }
64     inline ll gcd(ll a, ll b) {
65         return b == 0 ? a : gcd(b, a % b);
66     }
67     inline ll Abs(ll x) {
68         return x < 0 ? -x : x;

```

```

69     }
70     ll Pollard_rho(ll n) {                                     //基于路径倍增的Pollard_Rho算法
71         ll s = 0, t = 0, c = rand() % (n - 1) + 1, v = 1, ed = 1;
72         while (1) {
73             for (int i = 1; i <= ed; ++i) {
74                 t = (mul(t, t, n) + c) % n;
75                 v = mul(v, Abs(t - s), n);
76                 if (i % 127 == 0) {
77                     ll d = gcd(v, n);
78                     if (d > 1)
79                         return d;
80                 }
81             }
82             ll d = gcd(v, n);
83
84             if (d > 1)
85                 return d;
86             s = t;
87             v = 1;
88             ed <<= 1;
89         }
90     }
91     void getfactor(ll n) {                                     //得到所有的质因子(可能有重复的)
92         if (n <= sz) {
93             while (n != 1)
94                 fac[cnt ++ ] = p[n], n /= p[n];
95             max_factor = max_factor > p[n] ? max_factor : p[n];
96             return;
97         }
98         if (miller(n, 6)) {
99             fac[cnt ++ ] = n;
100             max_factor = max_factor > n ? max_factor : n;
101         }
102         else {
103             ll d = n;
104             while (d >= n)
105                 d = Pollard_rho(n);
106             getfactor(d);
107             getfactor(n / d);
108         }
109         return ;
110     }
111 } Q;
112 int main() {
113     //Q.init(N - 1);//如果代码超时且仅需要分解大数的质因数可以用这句话，否则不要用
114     ll T, n;
115     scanf("%lld", &T);
116     while (T--) {
117         max_factor = -1;
118         scanf("%lld", &n);
119         Q.getfactor(n);
120         if(max_factor == n)
121             puts("Prime");
122         else printf("%lld\n", max_factor);
123     }
124     return 0;
125 }

```

- 竞赛例题选讲

### Problem A 阶乘分解 (AcWing 197)

给定整数  $N$ ，试把阶乘  $N!$  分解质因数，按照算术基本定理的形式输出分解结果中的  $p_i$  和  $c_i$  即可。如：  $5! = 120 = 2^3 \times 3 \times 5$

### Solution

我们发现  $N!$  中质数因子  $p$  的个数,就是  $1 \sim N$  中每个数含有的质因数  $p$  个数之和。既然如此的话,那么我们发现，  $1 \sim N$  中，  $p$  的倍数，即至少有一个质因子  $p$  的数显然有  $\lfloor \frac{N}{p} \rfloor$  个，而  $p^2$  的倍数，即至少有两个质因子  $p$  数的显然是有  $\lfloor \frac{N}{p^2} \rfloor$ ，不过由于这两个质因子  $p$  中有一个已经在  $\lfloor \frac{N}{p} \rfloor$  统计过了，所以只需要再统计第二个质因子，也就是直接累加  $\lfloor \frac{N}{p^2} \rfloor$ ，而不是累乘。

即：

$$\left\lfloor \frac{N}{p} \right\rfloor + \left\lfloor \frac{N}{p^2} \right\rfloor + \cdots + \left\lfloor \frac{N}{p^{\log_p N}} \right\rfloor = \sum_{p^k \leq N} \left\lfloor \frac{N}{p^k} \right\rfloor$$

时间复杂度  $O(N \log N)$

```
1 int main()
2 {
3     int n;
4     cin >> n;
5     init(n); //线性筛代码略去
6     for (int i = 0; i < cnt; i++) {
7         int p = primes[i];
8         int s = 0;
9         for (int j = n; j; j /= p) s += j / p;
10        printf("%d %d\n", p, s);
11    }
12    return 0;
13 }
```

由 **Problem A 阶乘分解** 就可以引申出一道趣味题：

**Problem B 0的个数**

$n$  的阶乘  $n!$  中的末尾有多少个 0 ？

**Solution**

通过了Problem A 阶乘分解我们已经知道了如何求得  $n!$  中的质因子个数，所以这道题我们考虑如何应用一些。我们可以从 “**哪些质因子相乘可以得到10**” 的角度出发，发现只有 2 和 5 相乘的时候才会得到 10 。假设  $n! = 2^a \times 3^b \times 5^c \cdots$  。我们发现每一对 2 和 5 相乘就可以得到一个 10，所以 10 的个数  $num = \min\{a, c\}$  。由于  $a$  一定小于  $c$  ，所以  $num = c$  。

所以我们直接求出来  $n!$  质因数分解后 5 的次数即可。

**Code**

```
1 int number_0(int n)
2 {
3     int num = 0;
4
5     while(n) {
6         num += n / 5;
7         n /= 5;
8     }
9     return num;
10 }
```

这样又可以引申出一道题目：求  $n!$  的二进制表示中最低位 1 中的位置。

**Solution**

求  $n!$  的二进制表示中最低位 1 中的位置，我们发现实际上就是求二进制下最低位 1 后面 0 的个数。在所有小于  $n$  的数中，2 的倍数贡献一个 0，4 的倍数再贡献一个 0，所以答案就是  $n!$  质因数分解后 2 的次数。

**Problem C Divisors of the Divisors of an Integer** (2018-2019 ACM-ICPC, Asia Dhaka Regional ContestC)

给出  $n$ ，问  $n!$  的因子的因子的个数和。

**Solution**

学会上面的阶乘分解之后，我们能一眼看出来这道题也一定跟它有关系，所以我们按照惯例先对  $n!$  进行质因数分解。

$n! = p_1^{\alpha_1} \times p_2^{\alpha_2} \times \cdots \times p_k^{\alpha_k}$  。

我们单独考虑每一中质数，我们假设一个素数为  $p$ ，它的幂次  $\alpha$ ，因为我们要求的是因子的因子个数，因为它一共有幂次  $\alpha$ ，那么该因子的因子就有  $0, 1, 2, \cdots, \alpha$  的  $\alpha + 1$  种选择。

即：  $p^0, p^1, p^2, p^3, \cdots, p^\alpha$ ，对于  $p^0$  而言，因子个数为 1，对于  $p^1$  而言，因子个数为 2，对于  $p^\alpha$  而言，因子个数为  $\alpha + 1$ ，总的因子个数为：  
 $1 + 2 + 3 + \cdots + \alpha + 1 = \frac{(\alpha + 1)(\alpha + 2)}{2}$ 。（等差数列求和公式）

对于一个素数的贡献如上，那么总的贡献为： $\prod_{i=1}^k \frac{(\alpha_i + 1)(\alpha_i + 2)}{2}$ 。

我们由 **Problem A 阶乘分解** 知道了求解  $\alpha$  的方法，所以使用公式计算答案即可。

**Code**

```
1 const ll mod = 1e7 + 7;
2 int n, m;
3 int primes[N], cnt;
```



```

4  bool vis[N];
5  void init(int n)
6  {
7      for(int i = 2; i <= n; ++ i) {
8          if(vis[i] == 0) primes[ ++ cnt] = i;
9          for(int j = 1; j <= cnt && primes[j] * i <= n; ++ j) {
10             vis[i * primes[j]] = true;
11             if(i % primes[j] == 0) break;
12         }
13     }
14 }
15 ll f(ll x)
16 {
17     ll res = 0;
18     ll tmp = n;
19     while(tmp) {
20         res += tmp / x;
21         tmp /= x;
22     }
23     return res;
24 }
25 int main()
26 {
27     init(N - 1);
28     while(scanf("%d", &n) != EOF && n) {
29         ll res = 1;
30         for(int i = 1; i <= cnt && primes[i] <= n; ++ i) {
31             ll alpha = f(primes[i]);
32             res = (res * ((alpha + 1) * (alpha + 2)) / 2) % mod;
33         }
34         printf("%lld\n", res);
35     }
36     return 0;
37 }

```

### Problem D Multiply (2019 ACM- ICPC Asia Xuzhou Regional Contest E)

给  $n$  个数的序列  $a$ ，定义  $Z = \prod_{i=1}^n (a_i!)$ 。给出  $X, Y$ ，若  $b_i = Z \times X^i$ ，求出最大的  $i$  使得  $b_i \mid Y!$ 。

#### Solution

首先按照套路先将  $X$  质因数分解得  $X = p_1^{\alpha_1} p_2^{\alpha_2} p_3^{\alpha_3} \cdots p_n^{\alpha_n}$ 。对于某质因子  $p_i$ ，设  $Z$  中有  $a$  个， $X$  中有  $b$  个， $Y!$  中有  $c$  个。因为  $b_i = Z \times X^i$ ， $Z$  是一个常数，所以我们很自然地想到先去掉这个常数。

$$b_i \mid Y! \Rightarrow Z \times X^i \mid Y! \Rightarrow X^i \mid \frac{Y!}{Z}$$

即该质因子  $p_i$  最多对应  $\frac{c-a}{b}$  个，我们只需要贪心地求出所有质因子中最小的那个  $i$ ，即为答案。

#### Code

```

1  typedef long long ll;
2  const int N = 1e5 + 7;
3  ll x, y, a[N];
4  ll max_factor;
5  struct BigIntegerFactor {
6      const static int N = 1e6 + 7;
7      ll prime[N], p[N], fac[N], sz, cnt; //多组输入注意初始化cnt = 0
8      inline ll mul(ll a, ll b, ll mod) { //WA了尝试改为__int128或慢速乘
9          if (mod <= 1000000000)
10             return a * b % mod;
11         return (a * b - (ll)((long double)a / mod * b + 1e-8) * mod + mod) % mod;
12     }
13     void init(int maxn) {
14         int tot = 0;
15         sz = maxn - 1;
16         for (int i = 1; i <= sz; ++i)
17             p[i] = i;
18         for (int i = 2; i <= sz; ++i) {
19             if (p[i] == i)
20                 prime[tot++] = i;
21             for (int j = 0; j < tot && 1ll * i * prime[j] <= sz; ++j) {
22                 p[i * prime[j]] = prime[j];
23                 if (i % prime[j] == 0)

```

```

24         break;
25     }
26 }
27 }
28 ll qpow(ll a, ll x, ll mod) {
29     ll res = 1ll;
30     while (x) {
31         if (x & 1)
32             res = mul(res, a, mod);
33         a = mul(a, a, mod);
34         x >>= 1;
35     }
36     return res;
37 }
38 bool check(ll a, ll n) { //二次探测原理检验n
39     ll t = 0, u = n - 1;
40     while (!(u & 1))
41         t++, u >>= 1;
42     ll x = qpow(a, u, n), xx = 0;
43     while (t--) {
44         xx = mul(x, x, n);
45         if (xx == 1 && x != 1 && x != n - 1)
46             return false;
47         x = xx;
48     }
49     return xx == 1;
50 }
51 bool miller(ll n, int k) {
52     if (n == 2)
53         return true;
54     if (n < 2 || !(n & 1))
55         return false;
56     if (n <= sz)
57         return p[n] == n;
58     for (int i = 0; i <= k; ++i) { //测试k次
59         if (!check(rand() % (n - 1) + 1, n))
60             return false;
61     }
62     return true;
63 }
64 inline ll gcd(ll a, ll b) {
65     return b == 0 ? a : gcd(b, a % b);
66 }
67 inline ll Abs(ll x) {
68     return x < 0 ? -x : x;
69 }
70 ll Pollard_rho(ll n) { //基于路径倍增的Pollard_Rho算法
71     ll s = 0, t = 0, c = rand() % (n - 1) + 1, v = 1, ed = 1;
72     while (1) {
73         for (int i = 1; i <= ed; ++i) {
74             t = (mul(t, t, n) + c) % n;
75             v = mul(v, Abs(t - s), n);
76             if (i % 127 == 0) {
77                 ll d = gcd(v, n);
78                 if (d > 1)
79                     return d;
80             }
81         }
82         ll d = gcd(v, n);
83
84         if (d > 1)
85             return d;
86         s = t;
87         v = 1;
88         ed <<= 1;
89     }
90 }
91 void getfactor(ll n) { //得到所有的质因子(可能有重复的)
92     if (n <= sz) {
93         while (n != 1)
94             fac[cnt ++ ] = p[n], n /= p[n];

```

```

95     max_factor = max_factor > p[n] ? max_factor : p[n];
96     return;
97 }
98 if (miller(n, 6)) {
99     fac[cnt ++ ] = n;
100    max_factor = max_factor > n ? max_factor : n;
101 }
102 else {
103     ll d = n;
104     while (d >= n)
105         d = Pollard_rho(n);
106     getfactor(d);
107     getfactor(n / d);
108 }
109 return ;
110 }
111 ll cal(ll n, ll x) { //计算 n! 中质因子 x 的数量
112     ll num = 0;
113     while (n) {
114         num += n / x;
115         n = n / x;
116     }
117     return num;
118 }
119 ll solve(int n, ll x, ll y) {
120     map<ll, ll> mp;
121     ll ans = 4e18;
122     cnt = 0;
123     getfactor(x);
124     for (int i = 0; i < cnt; ++i)
125         mp[fac[i]]++;
126     map<ll, ll>::iterator it = mp.begin();
127     while (it != mp.end()) {
128         ll num = 0;
129         for (int i = 1; i <= n; ++i) {
130             num += cal(a[i], it->first);
131         }
132         ans = min(ans, (cal(y, it->first) - num) / it->second);
133         it ++ ;
134     }
135     return ans;
136 }
137 } Q;
138 int main() {
139     //Q.init(N - 1); //如果代码超时且仅需要分解大数的质因数可以用这句话，否则不要用
140     int T, n;
141     scanf("%d", &T);
142     while (T--) {
143         scanf("%d %lld %lld", &n, &x, &y);
144         for (int i = 1; i <= n; ++i)
145             scanf("%lld", a + i);
146         printf("%lld\n", Q.solve(n, x, y));
147     }
148     return 0;
149 }

```

## 0x12.2 $Z^*$ 结构中的一些定理

### 推论12.2.1:

若

$$n = p_1^{\alpha_1} \times p_2^{\alpha_2} \times p_3^{\alpha_3} \times \cdots \times p_k^{\alpha_k}$$

$$m = p_1^{\beta_1} \times p_2^{\beta_2} \times p_3^{\beta_3} \times \cdots \times p_k^{\beta_k}$$

则

$$n \times m = p_1^{\alpha_1+\beta_1} \times p_2^{\alpha_2+\beta_2} \times p_3^{\alpha_3+\beta_3} \times \cdots \times p_k^{\alpha_k+\beta_k}$$

$$\gcd(n, m) = p_1^{\min\{\alpha_1, \beta_1\}} \times p_2^{\min\{\alpha_2, \beta_2\}} \times \cdots \times p_k^{\min\{\alpha_k, \beta_k\}}$$

$$\text{lcm}(n, m) = p_1^{\max\{\alpha_1, \beta_1\}} \times p_2^{\max\{\alpha_2, \beta_2\}} \times \dots \times p_k^{\max\{\alpha_k, \beta_k\}}$$

**定理12.2.2:**  $(p-1)! + 1 \equiv 0 \pmod p$

**定理12.2.3:**  $((n+1)(n+2)\dots(n+k))\%k \neq 0$

0x12.3  $(Z_p^*, \cdot)$  结构

**定理12.3.1:**  $(Z_p^*, \cdot)$  是循环群, 即存在  $a \in Z_p^*$ , 使得  $Z_p^* = \{a^n | n = 1, 2, \dots, p-1\}$

这样的  $a$  称为  $p$  的原根。

素数一定有原根, 原根不唯一, 部分合数也有**原根**

- 1000000007 的原根为 5
- 998244353 的原根为 3

原根详见**0x60原根**。

0x13 最大公因数与最小公倍数

0x13.1 约数

**约数**, 又称因数。整数  $a$  除以整数  $b(b \neq 0)$  除得的商正好是整数而没有余数, 我们就说  $a$  能被  $b$  整除, 或  $b$  能整除  $a$ 。 $a$  称为  $b$  的倍数,  $b$  称为  $a$  的约数。

**唯一分解定理**, 任何一个大于 1 的数都可以被分解成有限个质数乘积的形式

$$N = \prod_{i=1}^m p_i^{C_i}$$

其中  $p_1 < p_2 < \dots < p_m$  为质数,  $C_i$  为正整数

$N$  的正约数个数为:

$$(c_1 + 1) \times (c_2 + 1) \times \dots \times (c_m + 1) = \prod_{i=1}^m (c_i + 1)$$

$$N^M \text{ 的正约数个数} = (M \times c_1 + 1) \times (M \times c_2 + 1) \times \dots \times (M \times c_m + 1) = \prod_{i=1}^m (M \times c_i + 1)$$

$N$  的所有正约数和为:

$$(1 + p_1 + p_1^2 + \dots + p_1^{c_1}) \times \dots \times (1 + p_m + p_m^2 + \dots + p_m^{c_m}) = \prod_{i=1}^m (\sum_{j=0}^{c_i} (p_i)^j)$$

随机数据下, 约数个数的期望是  $O(\ln n)$

- **试除法 - 求  $n$  的正约数集合**

显然约数总是成对出现 (除了完全平方数, 只有一个  $\sqrt{n}$ ), 所以只需要枚举到  $\sqrt{n}$  即可。

```
1 vector<int>factor;
2 int m;
3 int main()
4 {
5     scanf("%d", &n);
6     for(int i = 1; i * i <= n; ++ i){
7         if(n % i == 0){
8             factor.push_back(i);
9             if(i != n / i)
10                 factor.push_back(n / i);
11         }
12     }
13     for(int i = 0; i < factor.size(); ++ i){
14         printf("%d\n", factor[i]);
15     }
16     return 0;
17 }
18
```

推论: 一个整数  $n$  的约数个数上界为  $2\sqrt{n}$ 。

- **倍数法 - 求  $1 \sim n$  中每个数的正约数集合**

按照埃氏筛的形式枚举倍数, 时间复杂度为  $O(n + \frac{n}{2} + \frac{n}{3} + \dots + \frac{n}{n}) \approx n \times \log n = O(n \log n)$

时间复杂度为  $O(n \log n)$

可以求出  $1 \sim n$  中每个数的正约数集合, 但不能求出具体某个数的因子是谁, 常用于一些于因子有关的计算, 如计算  $\sum_{i=1}^n \sum_{d|n} d$  或是  $\sum_{i=1}^n \sum_{d|n} f(d)$ , 可以使用倍数法在  $O(n \log n)$  的复杂度下计算。

```
1 int main()
2 {
3     scanf("%d", &n);
```

```

4     for(int i = 1; i <= n; ++ i){
5         for(int j = 1 ;j * i <= n; ++ j){
6             factor[i * j].push_back(j);
7         }
8     }
9     for(int i = 1; i <= n; ++ i){
10        cout << i << ": ";
11        for(int j = 0; j < factor[i].size(); ++ j)
12            printf("%d ", factor[i][j]);
13        puts("");
14    }
15    return 0;
16 }
17

```

推论：  $1 \sim n$  中每个数的约数的总和大概为  $n \log n$ 。

● 竞赛例题选讲

**Problem A 约数之和 (AcWing 97)**

给定正整数  $A, B$ , 求  $A^B$  的所有约数之和  $\bmod 9901 (1 \leq A, B, \leq 5 \times 10^7)$ 。

**Solution**

根据唯一分解定理将A进行因式分解可得: $A = p_1^{a_1} \times p_2^{a_2} \times \cdots \times p_n^{a_n}$ .

$$A^B = p_1^{a_1 \times B} \times p_2^{a_2 \times B} \times \cdots \times p_n^{a_n \times B}$$

$$A^B \text{的所有约数之和} sum = (1 + p_1 + p_1^2 + \cdots + p_1^{a_1 \times B}) \times (1 + p_2 + p_2^2 + \cdots + p_2^{a_2 \times B}) \times \cdots \times (1 + p_n + p_n^2 + \cdots + p_n^{a_n \times B})$$

上式中每一项都是一个等比数列

根据等比数列求和公式：

$$S_n = n \times a_1 \quad (q = 1)$$

$$S_N = a_1 \times \frac{1 - q^n}{1 - q} = \frac{a_1 - a_n \times q}{1 - q} \quad (q \neq 1)$$

$$(1 + p_1 + p_1^2 + \cdots + p_1^{a_1 \times B}) = \frac{1 - p_1^{a_1 \times B + 1}}{1 - p_1} = \frac{p_1^{a_1 \times B + 1} - 1}{p_1 - 1}$$

我们可以用快速幂计算分子  $p_1^{a_1 \times B + 1} - 1 \pmod{9901}$ ，以及分母  $p_1 - 1 \pmod{9901}$ 。

因为 9901 是质数，所以只要  $p_1 - 1$  不是 9901 的倍数，我们就可以求分母的逆元用乘逆元的形式来代替除法，这样我们直接暴力计算整个式子即可。

特别地，若  $p_1 - 1$  是 9901 的倍数，则  $p_1 - 1 \equiv 0 \pmod{9901}$ ，此时的乘法逆元不存在，但是我们可以得到  $p_1 \equiv 1 \pmod{9901}$ ，所以  $(1 + p_1 + p_1^2 + \cdots + p_1^{a_1 \times B}) \equiv 1 + 1 + 1^2 + \cdots + 1^{a_1 \times B} \equiv a_1 \times B + 1 \pmod{9901}$ 。

**Code**

```

1  const int N = 507, M = 500007, INF = 0x3f3f3f3f;
2  const double eps = 1e-6;
3  ll a, b, m, mod = 9901;
4  ll ans = 1;
5  int p[N], c[N];
6  int cnt, n;
7  void divide(int n)//质因子分解
8  {
9      cnt = 0;
10     for(int i = 2; i * i <= n; ++ i) {
11         if(n % i == 0){
12             p[ ++ cnt] = i, c[cnt] = 0;
13             while(n % i == 0)n /= i, c[cnt] ++ ;
14         }
15     }
16     if(n > 1)
17         p[ ++ cnt] = n, c[cnt] = 1;
18 }
19 ll qpow(ll a, ll b)
20 {
21     ll res = 1;
22     while(b) {
23         if(b & 1)res = (res * a) % mod;
24         a = (a * a) % mod;
25         b >>= 1;
26     }
27     return res;
28 }

```

```

29 int main()
30 {
31     scanf("%lld%lld", &a, &b);
32     if(a == 0){puts("0");return 0;}//数据有毒啊
33     divide(a);
34     for(int i = 1; i <= cnt; ++ i) {
35         //cout << ans << endl;
36         if((p[i] - 1) % mod == 0){//没有逆元，需要特判
37             ans = (b * c[i] + 1) % mod * ans % mod;
38             continue;
39         }
40         ll x = qpow(p[i], (ll)b * c[i] + 1);//分子
41         x = (x - 1 + mod) % mod;
42         ll y = p[i] - 1;//分母
43         y = qpow(y, mod - 2);//费马小定理求乘法逆元
44         ans = ans * x % mod * y % mod;
45     }
46     printf("%lld\n", ans);
47 }

```

### Problem B. A New Function (LightOJ-1098)

定义约数和  $\sigma(n)$  为对于每个数求除 1 和它本身的约数和，求  $S(n) = \sum_{i=1}^n \sigma(i)$ 。

$$n \leq 2 \times 10^9$$

#### Solution

显然有：

$$S(n) = \sum_{i=2}^n i \times \left( \left\lfloor \frac{n}{i} \right\rfloor - 1 \right)$$

即对于  $1 \sim n$  内的数  $i$ ，含有因子  $i$  的数的个数为  $\left\lfloor \frac{n}{i} \right\rfloor$ ，除去它本身，显然有  $\left\lfloor \frac{n}{i} \right\rfloor - 1$  个，因为要除去 1，从 2 开始枚举，整除分块即可。

$$O(\sqrt{n})$$

#### Code

```

1 #define int long long
2 int n, m, s, t, k, ans, a[N], kcase;
3 void solve()
4 {
5     ans = 0;
6     scanf("%lld", &n);
7     for (int l = 2, r; l <= n; l = r + 1) {
8         r = n / (n / l);
9         ans += (l + r) * (r - l + 1) / 2 * (n / l - 1);
10    }
11    cout << ans << endl;
12 }
13 signed main()
14 {
15     scanf("%lld", &t);
16     while(t -- ) {
17         printf("Case %lld: ", ++ kcase);
18         solve();
19     }
20     return 0;
21 }

```

## Ox13.2 最大公约数

两个数  $a$  和  $b$  的**最大公约数** (GreatestCommonDivisor) 是指同时整除  $a$  和  $b$  的最大因数，记为  $\gcd(a, b)$ 。

**一个约定俗成的定理**：任何非零整数和零的最大公约数为它本身。

有如下基本性质：

**性质13.2.1**： $\gcd(a, b) = \gcd(b, a)$

**性质13.2.2**： $\gcd(a, b) = \gcd(a - b, b) (a \geq b)$

**性质13.2.3**： $\gcd(a, b) = \gcd(a \bmod b, b)$

**性质13.2.4**： $\gcd(a, b, c) = \gcd(\gcd(a, b), c)$



**性质13.2.5:**  $\gcd(ka, kb) = k \gcd(a, b)$

**性质13.2.6:**  $\gcd(k, ab) = 1 \iff \gcd(k, a) = 1 \ \&\& \ \gcd(k, b) = 1$

特别地, 如果  $a, b$  的  $\gcd(a, b) = 1$ , 则称这两个数互质 (互素)。

- 辗转相除法 (又称欧几里德算法)

$$\forall a, b \in \mathbb{N}, b \neq 0, \gcd(a, b) = \gcd(b, a \bmod b)$$

**欧几里德算法证明**

$a = kb + r = kb + a \% b$ , 则  $a \% b = a - kb$ 。令  $d$  为  $a$  和  $b$  的公约数, 则  $d \mid a$  且  $d \mid b$  根据整除的组合性原则, 有  $d \mid (a - kb)$ , 即  $d \mid (a \% b)$ 。

这就说明如果  $d$  是  $a$  和  $b$  的公约数, 那么  $d$  也一定是  $b$  和  $a \% b$  的公约数, 即两者的公约数是一样的, 所以最大公约数也必定相等。

时间复杂度  $O(\log n)$

最坏情况: 斐波那契数列相邻的两项, 因为斐波那契数列相邻的两项一定互质。

欧几里德算法由于存在大量的取模运算, 对于大整数耗时较大。

**Code**

```
1 int gcd(int a, int b){
2     return b == 0 ? a : gcd(b, a % b);
3 }
```

- 更相减损术:**  $\gcd(n, m) = \gcd(n, n - m)$

**证明1:**

设  $\gcd(n, n - m) = d$  令  $n = k_1 \times d, n - m = k_2 \times d, \gcd(k_1, k_2) = 1 \Rightarrow m = (k_1 - k_2) \times d$   
 $\therefore \gcd(n, m) = \gcd(k_1 \times d, (k_1 - k_2) \times d) = d = \gcd(n, n - m)$   
性质得证  $\square$

**证明2:**

由 **性质00.1.5** 可知,  $n, m$  的公约数集合与  $n, n - m$  的公约数集合完全一样, 故最大公约数自然也相等。  
性质得证  $\square$

- Stein 算法 (可看作更相减损术的应用)**

渐近时间, 空间复杂度均与欧几里德算法相同。

原理:  $\gcd(ka, kb) = k \times \gcd(a, b)$

最大特点: 只有移位和加减法计算, 避免了大整数的取模运算。

- 递归版**

```
1 int stein(int a, int b) {
2     if (a < b)
3         a ^= b, b ^= a, a ^= b; //交换, 使a为较大数;
4     if (b == 0)
5         return a; //当相减为零, 即两数相等时, gcd=a;
6     if ((!(a & 1)) && !(b & 1))
7         return stein(a >> 1, b >> 1) << 1; //s1,注意最后的左移, 在递归返回过程中将2因子乘上;
8     else if ((a & 1) && !(b & 1))
9         return stein(a, b >> 1); //s2;
10    else if (!(a & 1) && (b & 1))
11        return stein(a >> 1, b);
12    else
13        return stein(a - b, b); //s3;
14 }
```

- 迭代版**

```
1 int stein(int a, int b) {
2     int k = 1;
3     while ((!(a & 1)) && !(b & 1)) { //s1;
4         k <<= 1; //用k记录全部公因子2的乘积 ;
5         a >>= 1;
6         b >>= 1;
7     }
8     while (!(a & 1)) a >>= 1; //s2;
9     while (!(b & 1)) b >>= 1;
10    if (a < b) a ^= b, b ^= a, a ^= b; //交换, 使a为较大数;
11    while (a != b) { //s3;
12        a -= b;
13        if (a < b)
14            a ^= b, b ^= a, a ^= b;
15    }
16    return k * a;
```

• 竞赛例题选讲

Problem A 永远永远

$f[0] = 0$ , 当  $n > 1$  时,  $f[n] = (f[n - 1] + a) \% b$ , 给定  $a$  和  $b$ , 问是否存在一个自然数  $k$  ( $0 \leq k < b$ ), 是  $f[n]$  永远都取不到的。

Solution

我们发现这里的  $f[\dots]$  一定是有循环节的, 如果在某个循环节内都无法找到那个自然数  $k$ , 那么必定是永远都找不到了。

求出  $f[n]$  的通项公式, 为  $f[n] = an \% b$ , 令  $an = kb + r$ , 那么这里的  $r = f[n]$ , 如果  $t = \gcd(a, b)$ ,  $r = an - kb = t((\frac{a}{t})n - (\frac{b}{t})k)$ , 则有  $t \mid r$ , 要满足所有的  $r$  使得  $t \mid r$ , 只有当  $t = 1$  的时候, 于是这个问题的解也就出来了, 只要求  $a$  和  $b$  的  $\gcd$ , 如果  $\gcd(a, b) > 1$ , 则存在一个  $k$  使得  $f[n]$  永远都取不到, 直观的理解是当  $\gcd(a, b) > 1$ , 那么  $f[n]$  不可能是素数。

一个简单的应用就是本文 **0x22.1** 的 **Problem A Fox And Jumping**。

0x13.3 最小公倍数

两个数  $a$  和  $b$  的**最小公倍数** (LeatestCommonMultiple) 是指同时被  $a$  和  $b$  整除的最小倍数, 记为  $\text{lcm}(a, b)$ 。特殊的, 当  $a$  和  $b$  互素时,  $\text{lcm}(a, b) = ab$ 。

**性质13.3.1**:  $\forall a, b \in N, \gcd(a, b) \times \text{lcm}(a, b) = a \times b$

可以使用  $\gcd$ ,  $\text{lcm}$  的定义证明 **性质13.3.1**, 证明略。

```
1 int lcm(int a,int b){
2     return a / gcd(a,b) * b;//先除后乘，以免溢出64位整数
3 }
```

• 重要性质:  $\gcd$  与  $\text{lcm}$  的指数最值表示法

由唯一分解定理得, 若

$$n = p_1^{\alpha_1} \times p_2^{\alpha_2} \times p_3^{\alpha_3} \times \dots \times p_k^{\alpha_k}$$

$$m = p_1^{\beta_1} \times p_2^{\beta_2} \times p_3^{\beta_3} \times \dots \times p_k^{\beta_k}$$

则

$$n \times m = p_1^{\alpha_1+\beta_1} \times p_2^{\alpha_2+\beta_2} \times p_3^{\alpha_3+\beta_3} \times \dots \times p_k^{\alpha_k+\beta_k}$$

$$\gcd(n, m) = p_1^{\min\{\alpha_1, \beta_1\}} \times p_2^{\min\{\alpha_2, \beta_2\}} \times \dots \times p_k^{\min\{\alpha_k, \beta_k\}}$$

$$\text{lcm}(n, m) = p_1^{\max\{\alpha_1, \beta_1\}} \times p_2^{\max\{\alpha_2, \beta_2\}} \times \dots \times p_k^{\max\{\alpha_k, \beta_k\}}$$

• 竞赛例题选讲

Problem A GCD and LCM (hdu 4497)

三个未知数  $x, y, z$ , 它们的  $\gcd$  为  $G$ ,  $\text{lcm}$  为  $L$ ,  $G$  和  $L$  已知, 求  $(x, y, z)$  三元组的个数。

Solution

三个数的  $\gcd$  可以参照两个数  $\gcd$  的指数最值表示法, 只不过每个素因子的指数上是三个数的最值 (即  $\min\{x_1, y_1, z_1\}$ ), 那么这个问题首先要做的就是将  $G$  和  $L$  分别进行素因子分解, 然后轮询  $L$  的每个素因子, 对于每个素因子单独处理。

假设素因子为  $p$ ,  $L$  分解式中  $p$  的指数为  $l$ ,  $G$  分解式中  $p$  的指数为  $g$ , 那么显然  $l < g$  时不可能存在满足条件的三元组, 所以只需要讨论  $l \geq g$  的情况, 对于单个  $p$  因子, 问题转化成了求三个数  $x_1, y_1, z_1$ , 满足  $\min\{x_1, y_1, z_1\} = g$  且  $\max\{x_1, y_1, z_1\} = l$ , 更加通俗的意思就是三个数中最小的数是  $g$ , 最大的数是  $l$ , 另一个数在  $[g, l]$  范围内, 这是一个排列组合问题, 三元组  $x_1, y_1, z_1$  的种类数当  $l == g$  时只有 1 种, 否则答案就是  $6(l - g)$ 。

最后根据乘法原理将每个素因子对应的种类数相乘就是最后的答案了。

Code

```
1 int m, n, ans;
2 int t, num[1010], cnt;
3 int main()
4 {
5     scanf("%d", &t);
6     while(t -- ) {
7         scanf("%d%d", &n, &m);
8         if(m % n != 0) {
9             printf("0\n");
10            continue;
11        }
12        m = m / n;
13        cnt = 0;
14        for(int i = 2; i * i <= m; i ++ ) {
```

```
15         if(m % i == 0) {
16             num[cnt] = 0;
17             while(m % i == 0) {
18                 m = m / i;
19                 num[cnt] ++ ;
20             }
21             cnt ++ ;
22         }
23     }
24     if(m != 1) num[cnt ++ ] = 1;
25     ans = 1;
26     for(int i = 0; i < cnt; i ++ ) ans = ans * num[i] * 6;
27     printf("%d\n", ans);
28 }
29 return 0;
30 }
```

ox13.4 GCD 与 LCM 的一些性质与定理

- 性质13.4.1:**  $\gcd(F(n), F(m)) = F(\gcd(n, m))$
- 性质13.4.2:**  $\gcd(a^m - 1, a^n - 1) = a^{\gcd(n, m)} - 1$  ( $a > 1, n > 0, m > 0$ ) (证明待更...)
- 性质13.4.3:**  $\gcd(a^m - b^m, a^n - b^n) = a^{\gcd(m, n)} - b^{\gcd(m, n)}$  ( $\gcd(a, b) = 1$ )
- 性质13.4.4:**  $\gcd(a, b) = 1, \gcd(a^m, b^n) = 1$
- 性质13.4.5:**  $(a + b) \mid ab \implies \gcd(a, b) \neq 1$
- $a, b$ 不互质, 因为互质就提不出来公因子了。 [例题](#)
- 性质13.4.6:** 设 $G = \gcd(C_n^1, C_n^2, \dots, C_n^{n-1})$
- $n$  为素数,  $G = n$
  - $n$  非素且有一个素因子  $p$ ,  $G = p$
  - $n$  有多个素因子,  $G = 1$
- 性质13.4.7:**  $(n + 1)\text{lcm}(C_n^0, C_n^1, \dots, C_n^n) = \text{lcm}(1, 2, \dots, n + 1)$
- 性质13.4.8:**  $\sum_{i=1}^n \gcd(i, n) = \sum_{d \mid n} d \varphi(\frac{n}{d})$
- 性质13.4.8:** 在 `Fibonacci` 数列中求相邻两项的 `gcd` 时, **辗转相减**次数等于**辗转相除**次数。
- 性质13.4.8:**  $\gcd(fib_n, fib_m) = fib_{\gcd(n, m)}$  ([证明](#))
- 详见推论12.2.1

ox13.5 补充知识: *Fibonacci* 数列及其推论

- 基本性质定理:**
$$fib_n = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ fib_{n-1} + fib_{n-2} & n > 1 \end{cases}$$
  - 推导结论:**
- 性质13.5.1:**  $\sum_{i=1}^n f_i = f_{n+2} - 1$
- 性质13.5.2:**  $\sum_{i=1}^n f_{2i-1} = f_{2n}$
- 性质13.5.3:**  $\sum_{i=1}^n f_{2i} = f_{2n+1} - 1$
- 性质13.5.4:**  $\sum_{i=1}^n (f_n)^2 = f_n f_{n+1}$
- 性质13.5.5:**  $f_{n+m} = f_{n-1} f_{m-1} + f_n f_m$
- 性质13.5.6:**  $(f_n)^2 = (-1)^{(n-1)} + f_{n-1} f_{n+1}$
- 性质13.5.7:**  $f_{2n-1} = (f_n)^2 - (f_{n-2})^2$
- 性质13.5.8:**  $f_n = \frac{f_{n+2} + f_{n-2}}{3}$
- 性质13.5.9:**  $\frac{f_i}{f_{i-1}} \approx \frac{\sqrt{5}-1}{2} \approx 0.618$
- 性质13.5.10:**  $f_n = \frac{\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n}{\sqrt{5}}$  ([证明](#))

## Ox14 互质与欧拉函数

### Ox14.1 欧拉函数

#### 定义

$\forall a, b \in N$  若  $\gcd(a, b) = 1$ , 则称  $a, b$  **互质**。

对于三个数或更多的数, 把  $\gcd(a, b, c) = 1$  称之为  $a, b, c$  互质。

把  $\gcd(a, b) = \gcd(a, c) = \gcd(b, c) = 1$  称之为  $a, b, c$  两两互质。显然  $a, b, c$  两两互质是优于  $a, b, c$  互质的。

**性质14.1.1:**  $int$  范围内的数  $n$  中,  $1 \sim n$  中与  $n$  互质的个数最多只有1600 ( $\max\{\varphi(1 \sim 2147483647)\} = 1600$ )。

#### 欧拉函数

$1 \cdots N$  中与  $N$  互质的数的**个数**, 被称为欧拉函数, 记作  $\varphi(N)$ , **phi**。

如果  $n$  是一个素数, 那么  $\varphi(n) = n - 1$  (所有小于  $n$  的都互素)

如果  $n$  是素数的  $k$  次幂, 即  $n = p^k$ , 那么  $\varphi(p^k) = p^k - p^{k-1}$  (除了  $p$  的倍数以外, 与  $1 \sim n$  中的任意数都互素)

故我们可以得到下列结论:

由算数基本定理(唯一分解定理)得

$$N = p_1^{k_1} \times p_2^{k_2} \times p_3^{k_3} \times \cdots p_m^{k_m}$$

则有:

$$\varphi(N) = N \times \prod_{p|N} \left(1 - \frac{1}{p}\right)$$

证明见: **0x15 Problem A Co-prime**

由于欧拉函数是积性函数, 由 **性质14.2.3** 得:

$$\begin{aligned} \varphi(N) &= \varphi(p_1^{k_1} \times p_2^{k_2} \times p_3^{k_3} \times \cdots p_m^{k_m}) \\ &= \varphi(p_1^{k_1}) \times \varphi(p_2^{k_2}) \times \varphi(p_3^{k_3}) \times \cdots \times \varphi(p_m^{k_m}) \\ &= (p_1^{k_1} - p_1^{k_1-1}) \times (p_2^{k_2} - p_2^{k_2-1}) \times (p_3^{k_3} - p_3^{k_3-1}) \times \cdots (p_m^{k_m} - p_m^{k_m-1}) \\ &= (p_1^{k_1} \times p_2^{k_2} \times p_3^{k_3} \times \cdots p_m^{k_m}) \times \left(1 - \frac{1}{p_1}\right) \times \left(1 - \frac{1}{p_2}\right) \times \cdots \times \left(1 - \frac{1}{p_m}\right) \\ &= N \times \left(1 - \frac{1}{p_1}\right) \times \left(1 - \frac{1}{p_2}\right) \times \cdots \times \left(1 - \frac{1}{p_m}\right) \\ &= N \times \prod_{p|N} \left(1 - \frac{1}{p}\right) \end{aligned}$$

其中, 如果  $p$  是素数则  $\varphi(p) = p \times \left(1 - \frac{1}{p}\right) = p - 1$ 。

我们可以利用这个性质在分解质因数的同时  $O(\sqrt{n})$  使用公式求解欧拉函数

```
1 inline int euler_one(int n)
2 {
3     int ans = n;
4     for(int i = 2; i * i <= n; ++ i){
5         if(n % i == 0){
6             ans = ans / i * (i - 1);
7             while(n % i == 0) n /= i;
8         }
9     }
10    if(n > 1) ans = ans / n * (n - 1);
11    return ans;
12 }
```

显然我们也可以筛出质数之后用小于  $\sqrt{n}$  的质数再去分解质因数求欧拉函数时间复杂度为  $\frac{\sqrt{n}}{\log n}$ 。

我们也可以利用容斥原理得到欧拉函数的计算公式, 这里不再展开, 详见 **0x15 容斥原理初探**

### Ox14.2 欧拉函数的性质

#### • 积性函数

如果当  $a, b$  互质时, 满足  $f(ab) = f(a) \times f(b)$  的函数  $f$  称为积性函数。积性函数实际上是由欧拉函数推广到一般的函数上得到的概念, 我们将在本文 **0x30积性函数** 中详细探讨这类问题。

#### • 欧拉函数性质

**性质14.2.0**: 当  $n > 2$  时,  $\varphi(n)$  是偶数。

证明

由于更相减损术,  $\gcd(n, m) = \gcd(n, n - m)$ .

若  $n, m$  互质, 则有:  $\gcd(n, m) = 1$  ,  $\gcd(n, n - m) = 1(n > m)$

所以每一个与  $n$  互质的数  $m$  都对应一个  $n - m$  与之互质, 所以  $\varphi(n)$  是偶数。

**性质14.2.1**:  $\forall n > 1, 1 \cdots n$  中与  $n$  互质的数的和为 $n \times \frac{\varphi(n)}{2}$

证明

因为 $\gcd(n, x) = \gcd(n, n - x)$ , 所以与  $n$  不互质的数  $x, n - x$  一定成对出现, 平均值为  $\frac{n}{2}$  , 因此与  $n$  互质的数的平均值也是  $\frac{n}{2}$  , 进而得到性质14.2.1。

**性质14.2.2**: 若  $a, b$  互质, 则  $\varphi(ab) = \varphi(a) \times \varphi(b)$ 。

证明

根据欧拉函数的计算式, 对  $a, b$  分解质因数, 直接可得性质14.2.2。

**性质14.2.3**: 若  $f$  是积性函数, 且在算数基本定理中  $n = \prod_{i=1}^m p_i^{c_i}$ , 则 $f(n) = \prod_{i=1}^m f(p_i^{c_i})$ 。

**性质14.2.4**: 设  $p$  为质数, 若  $p \mid n$  且  $p^2 \nmid n$ , 则  $\varphi(n) = \varphi\left(\frac{n}{p}\right) \times p$ 。 ( $p \mid n$ , 即  $p$  是  $n$  的因数)

**性质14.2.5**: 设  $p$  为质数, 若  $p \mid n$  且 $p^2 \mid n$ ,则  $\varphi(n) = \varphi\left(\frac{n}{p}\right) \times (p - 1)$ 。

**性质14.2.6**:  $\sum_{d \mid n} \varphi(d) = n$ 。

证明

如果  $\gcd(k, n) = d$ , 那么  $\gcd(\frac{k}{d}, \frac{n}{d}) = 1$ , ( $k < n$ ) 。

如果我们设  $f(x)$  表示  $\gcd(k, n) = x$  的数的个数, 那么  $n = \sum_{i=1}^n f(i)$ 。(显然包含  $1 \sim n$  中所有的数)

根据上面的证明, 我们发现,  $f(x) = \varphi(\frac{n}{x})$ , 从而  $n = \sum_{d \mid n} \varphi(\frac{n}{d})$ 。注意到约数  $d$  和  $\frac{n}{d}$  具有对称性, 所以上式化为  $n = \sum_{d \mid n} \varphi(d)$ 。

**推论14.2.7**:  $\sum_{i=1}^n i[\gcd(i, n) = 1] = \frac{n\varphi(n) + [n = 1]}{2}$  (例题)

**推论14.2.8**:  $f(n) = \sum_{i=1}^n [\gcd(i, k) = 1] = \frac{n}{k} \varphi(k) + f(n \bmod k)$

**推论14.2.9**: 若  $i, j$  不互质, 则  $\varphi(i \times j) = \frac{\varphi(i)\varphi(j) \gcd(i, j)}{\varphi(\gcd(i, j))}$

**推论14.2.9证明**:

由欧拉函数计算式:  $\varphi(n) = n \times \prod_{p \mid n} (1 - \frac{1}{p})$

$$\begin{aligned} \varphi(i \times j) &= i \times j \times \prod_{p \mid ij} (1 - \frac{1}{p}) \\ &= \frac{i \times \prod_{p \mid i} (1 - \frac{1}{p}) \times j \times \prod_{p \mid j} (1 - \frac{1}{p})}{\prod_{p \mid i \text{ 且 } p \mid j} (1 - \frac{1}{p})} \\ &= \frac{\gcd(i, j) \times i \times \prod_{p \mid i} (1 - \frac{1}{p}) \times j \times \prod_{p \mid j} (1 - \frac{1}{p})}{\gcd(i, j) \times \prod_{p \mid i \text{ 且 } p \mid j} (1 - \frac{1}{p})} \\ &= \frac{\gcd(i, j) \times i \times \prod_{p \mid i} (1 - \frac{1}{p}) \times j \times \prod_{p \mid j} (1 - \frac{1}{p})}{\gcd(i, j) \times \prod_{p \mid \gcd(i, j)} (1 - \frac{1}{p})} \\ &= \frac{\varphi(i)\varphi(j) \gcd(i, j)}{\varphi(\gcd(i, j))} \end{aligned}$$

**推论14.2.10**:  设  $n = i \times d^k$ , 若  $k > 2$ , 则  $\varphi(n) = d \times \varphi(\frac{n}{d})$ 。

 若  $i, j$  互质,  $\varphi(ijd^{k+2}) = d^k \varphi(ijd^2) = d^k \times \frac{\varphi(id) \times \varphi(jd) \times d}{\varphi(d)}$

**推论14.2.10证明**:

设  $n = i \times d^k$ , 若  $k > 2$ , 显然  $n$  与  $\frac{n}{d}$  具有相同的质因子, 设  $n = \prod_{i=1}^m p_i^{c_i}$

$$\text{故 } \frac{\varphi(n)}{\varphi(\frac{n}{d})} = \frac{n \times \prod_{i=1}^m (1 - \frac{1}{p_i})}{\frac{n}{d} \times \prod_{i=1}^m (1 - \frac{1}{p_i})} = \frac{n}{\frac{n}{d}} = d$$

$$\text{即: } \varphi(n) = d \times \varphi(\frac{n}{d})$$

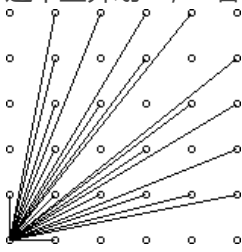
欧拉函数的筛法详见本文 **0x51.1 线性筛求欧拉函数**。

• 竞赛例题宣讲

**Problem A. 仪仗队** ([Luogu P2158 [SDOI2008]](<https://www.luogu.com.cn/problem/P2158>))

作为体育委员, C君负责这次运动会仪仗队的训练。仪仗队是由学生组成的  $N \times N$  的方阵, 为了保证队伍在行进中整齐划一, C君会跟在仪仗队的左后方,

根据其视线所及的学生人数来判断队伍是否整齐(如下图)。 现在, C君希望你告诉他队伍整齐时能看到的学生人数。



**Solution**

首先我们将原图从  $(1, 1)$  到  $(n, n)$  重新标号  $(0, 0)$  到  $(n - 1, n - 1)$

分析题目可知, 当  $n = 1$  时, 显然答案是 0, 特判断即可。

我们考虑  $n \neq 1$  的情况, 首先  $(0, 1), (1, 0), (1, 1)$  这三个点是一定能看到的

考虑剩余的点

对于任意的点  $(x, y)$ ,  $2 \leq x, y \leq n - 1$ , 若不被其它的点挡住必定满足  $\gcd(x, y) = 1$ 。

**证明:**

若点  $(x, y)$  不满足  $\gcd(x, y) = 1$ , 令  $d = \gcd(x, y)$   
则  $(x, y)$  与  $(0, 0)$  连线的斜率  $k = \frac{y}{x} = \frac{y/d}{x/d}$   
所以  $(x, y)$  会被  $(x/d, y/d)$  挡住

然后我们把这个图按照对角线分成两个等腰直角三角形

若点  $(x, y)$  在一个三角形中, 并满足  $\gcd(x, y) = 1$

则必有一个点  $(y, x)$  在另一个三角形中, 并满足  $\gcd(y, x) = 1$

所以只统计其中一个三角形即可

现在我们在  $y < x$  的三角形中考虑

对于任意一个  $x$ , 满足条件的  $y$  的个数就是  $\phi(x)$

所以答案就是

$$3 + 2 \times \sum_{i=2}^{n-1} \phi(i)$$

[https://blog.csdn.net/weixin\\_45697774](https://blog.csdn.net/weixin_45697774)

我们使用线性筛  $O(n)$  筛出欧拉函数然后计算答案即可。

**Time**

$O(n)$

**Code**

```
1 #include<cstdio>
2 #include<cmath>
3 #include<algorithm>
4 #include<iostream>
5 #include<cstring>
6
7 using namespace std;
8 typedef long long ll;
9 const int N = 50007, M = 50007, INF = 0x3f3f3f3f;
10 const double eps = 1e-6;
```



```
11
12 int n, m;
13
14 int vis[N];
15 int primes[N], phi[N], cnt;
16 int sum;
17
18 void get_euler(int n)
19 {
20     phi[1] = 1;
21     for(int i = 2; i <= n; ++ i) {
22         if(!vis[i]) {
23             primes[ ++ cnt] = i;
24             phi[i] = i - 1;
25         }
26         for(int j = 1; j <= cnt && i * primes[j] <= n; ++ j) {
27             vis[i * primes[j]] = true;
28             if(i % primes[j] == 0) { //最小的质因子
29                 phi[i * primes[j]] = phi[i] * primes[j];
30                 break;
31             }
32             else phi[i * primes[j]] = phi[i] * (primes[j] - 1);
33         }
34     }
35 }
36
37 int main()
38 {
39     scanf("%d", &n);
40     if(n == 1) puts("0") , exit(0);
41     n --;
42     get_euler(n);
43     sum = 0;
44     for(int i = 2; i <= n; ++ i)
45         sum += phi[i];
46     int ans = 3 + 2 * sum;
47     printf("%d\n", ans);
48     return 0;
49 }
```

## Ox15 容斥原理初探

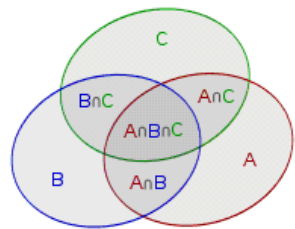
容斥原理虽然是组合数学的内容，但是与数论配合较为密切，因此在这里简单介绍一下容斥原理以及它与数论的一些简单结合。

容斥原理是一种应用在集合上的较常用的计数方法，其基本思想是：先不考虑重叠的情况，把包含于某内容中的所有对象的数目先计算出来（**容**），然后再把计数时重复计算的数目排斥出去（**斥**），使得计算的结果既无遗漏又无重复。

容斥原理核心的计数规则可以归为一句话：**奇加偶减**。

假设被计数的有  $A$ 、 $B$ 、 $C$  三类，那么， $A$ 、 $B$ 、 $C$  类元素个数总和 =  $A$  类元素个数 +  $B$  类元素个数 +  $C$  类元素个数 - 既是  $A$  又是  $B$  的元素个数 - 既是  $A$  又是  $C$  的元素个数 - 既是  $B$  又是  $C$  的元素个数 + 既是  $A$  又是  $B$  且是  $C$  的元素个数。

即： $A \cup B \cup C = A + B + C - AB - BC - AC + ABC$



当被计数的种类被推到  $n$  类时，其统计规则遵循**奇加偶减**。

### • 竞赛例题选讲

#### Problem A Co-prime (HDU 4135)

求区间  $[a, b]$  中与  $n$  互质的数的个数，其中  $1 \leq a, b, n \leq 2^{31}$ 。

#### Solution

容斥定理最常用于求解  $[a, b]$  区间与  $n$  互质的数的个数问题，该问题可以视为求  $[1, b]$  区间与  $n$  互质的个数减去  $[1, a - 1]$  区间内与  $n$  互质的个数。

那么我们这里只需要考虑它的一个子问题：求小于等于  $m$  且  $n$  互素的数的个数。

我们知道当  $m$  等于  $n$ ，就是一个简单的欧拉函数问题，但是一般  $m$  都不等于  $n$ ，我们考虑将  $n$  质因数分解。

我们首先分析最简单的情况：当  $n$  为素数的幂，即  $n = p^k$  时，那么显然答案就等于  $m - \left\lfloor \frac{m}{p} \right\rfloor$ （其中  $\left\lfloor \frac{m}{p} \right\rfloor$  表示的是  $p$  的倍数， $1 \sim m$  中去掉  $p$  的倍数，剩下的就都是与  $n$  互素的数了）

然后再来讨论  $n$  是两个素数的幂的乘积，即  $n = p_1^{k_1} \times p_2^{k_2}$ ，那么我们需要做的就是找到  $p_1$  的倍数和  $p_2$  的倍数，并且要减去  $p_1$  和  $p_2$  的公倍数，我们发现这实际上就是容斥原理，所以这种情况下答案为： $m - \left( \frac{m}{p_1} + \frac{m}{p_2} - \frac{m}{p_1 \times p_2} \right)$ 。

（拓展到  $n$  的所有质因子即可得到欧拉函数的计算式）

这里的  $+$  就是 **容**， $-$  就是 **斥**，并且 **容** 和 **斥** 总是交替进行的（一个的加上，两个的减去，三个的加上，四个的减去），而且可以推广到  $n$  个元素的情况，如果  $n$  分解成  $s$  个素因子，也同样可以用容斥原理求解。

容斥原理其实是枚举子集的过程，常见的枚举方法为  $dfs$ ，也可以采用二进制法（0 表示取，1 表示不取）。

例如我们求  $[1, 9]$  中和 6 互素的数的个数，这时 6 分解的素因子为 2 和 3。

$ans = 9 - \left( \frac{9}{2} + \frac{9}{3} \right) + \frac{9}{6} = 3$ ，其中， $ans$  分为三部分，0 个数的组合，1 个数的组合，2 个数的组合。

答案 = （ $1 \sim b$  的元素个数） - （ $1 \sim a - 1$  的元素个数） - （ $1 \sim b$  中与  $n$  不互质的数的个数） + （ $1 \sim a - 1$  中与  $n$  不互质的数的个数）

## Code

```
1  const int N = 10005;
2  ll n, primes[N], cnt, factor[N], num;
3  bool vis[N];
4  inline void get_primes(int n)
5  {
6      for(register int i = 2; i <= n; i++) {
7          if(!vis[i]) primes[ ++ cnt] = i;
8          for(register int j = 1; j <= cnt && i * primes[j] <= n; ++ j) {
9              vis[i * primes[j]] = 1;
10             if(i % primes[j] == 0) break;
11         }
12     }
13 }
14 void get_factor(int n) {
15     num = 0;
16     for (ll i = 1; primes[i] * primes[i] <= n && i <= cnt; i++) {
17         if (n % primes[i] == 0) { //记录n的因子
18             factor[num ++ ] = primes[i];
19             while (n % primes[i] == 0)
20                 n /= primes[i];
21         }
22     }
23     if (n != 1) //1既不是素数也不是合数
24         factor[num ++ ] = n;
25 }
26 ll solve(ll m, ll num) {
27     ll res = 0;
28     for (ll i = 1; i < (1 << num); i++) {
29         ll sum = 0;
30         ll temp = 1;
31         for (ll j = 0; j < num; j++) {
32             if (i & (1 << j)) {
33                 sum ++ ;
34                 temp *= factor[j];
35             }
36         }
37         if (sum % 2) res += m / temp;
38         else res -= m / temp;
39     }
40     return res;
41 }
42 int kcase, t;
43 int main() {
44     get_primes(N - 1);
45     scanf("%d", &t);
46     while(t -- ) {
47         ll a, b, n;
48         scanf("%lld%lld%lld", &a, &b, &n);
49         get_factor(n);
50         //容斥定理，奇加偶减，
51         ll res = (b - (a - 1) - solve(b, num)) + solve(a - 1, num);
52         printf("Case #d: %lld\n", ++ kcase, res);
53     }
```

```
53     }
54     return 0;
55 }
```

### Problem B Helping Cicada (LightOJ - 1117)

给定  $m$  个数。求  $[1, n]$  中不能被这  $m$  个数整除的数的个数。

#### Solution

我们知道对于任意一个数  $k$ ，在  $[1, n]$  中有  $\left\lfloor \frac{n}{k} \right\rfloor$  个数是  $k$  的倍数，也就是能被  $k$  整除，故  $ans = n - \sum_{i=1}^m \left\lfloor \frac{n}{a[i]} \right\rfloor$ 。

我们再来考虑两个数  $a, b$  的情况，因为  $a, b$  的公倍数  $\text{lcm}(a, b)$ ，即被  $a$  整除，又被  $b$  整除，所以减了两次。所以最后的答案要加上  $\left\lfloor \frac{n}{\text{lcm}(a, b)} \right\rfloor$ 。对于三个数  $a, b, c$  来说，答案就要减去  $\left\lfloor \frac{n}{\text{lcm}(a, b, c)} \right\rfloor$ ，四个数就要加上，五个就要减去，以此类推。

最后拓展到  $m$  个数的情况，我们发现需要使用容斥定理。

根据容斥定理的**奇加偶减**，对于  $m$  个数来说，其中的任意  $2, 4, \dots, 2k$  个数就要**减去**他们最小公倍数能组成的数， $1, 3, \dots, 2k + 1$  个数就要**加上**他们的最小公倍数，对于每一个数来说，我们都有选或不选两种情况，因此  $m$  个数就有  $2^m$  种情况，也就是从 0（啥也不选）到  $2^m - 1$ ，对应二进制就是  $m$  个 1，也就意味着我们选择了  $m$  个数。这种方法叫做状态压缩，我们接用状态压缩来枚举所有的可能状态，依次使用位运算判断当前的状态选择了多少个数，然后再根据容斥原理进行奇加偶减即可。

$sum =$  （从  $m$  中选 1 个数得到的倍数的个数） $-$  （从  $m$  中选 2 个数得到的倍数的个数） $+$  （从  $m$  中选 3 个数得到的倍数的个数） $-$  （从  $m$  中选 4 个数得到的倍数的个数） $\dots$

那么能被整除的数的个数就是  $sum$ ，不能被整除的数的个数就是  $n - sum$ 。

#### Code

```
1  const int N = 10005;
2  ll LCM(ll a, ll b) {
3      return a / __gcd(a, b) * b;
4  }
5  int m, kcase;
6  ll n, a[N];
7  int main()
8  {
9      int t;
10     scanf("%d", &t);
11     while(t -- ) {
12         scanf("%lld%d", &n, &m);
13         for(int i = 0; i < m; ++ i)
14             scanf("%lld", &a[i]);
15         ll sum = 0;
16         for(int i = 0; i < (1 << m); ++ i) {
17             ll lcm = 1;
18             ll cnt = 0;
19             for(int j = 0; j < m; ++ j) {
20                 if(i >> j & 1) { // 如当前的状态 i 选择了第 j 个数
21                     lcm = LCM(lcm, a[j]); // 那就选第 j 个数
22                     cnt ++ ;
23                 }
24             }
25             if(cnt != 0) {
26                 if(cnt & 1) sum += n / lcm; // 奇加
27                 else sum -= n / lcm; // 偶减
28             }
29         }
30         printf("Case %d: %lld\n", ++ kcase, n - sum);
31     }
32     return 0;
33 }
```

### Problem C 硬币购物 ([P1450 [HAOI2008]](<https://www.luogu.com.cn/problem/P1450>))

共有 4 种硬币。面值分别为  $c_1, c_2, c_3, c_4$ 。

某人去商店买东西，去了  $n$  次，对于每次购买，他带了  $d_i$  枚  $i$  种硬币，想购买  $s$  的价值的东西。请问每次有多少种付款方法。

#### Solution

看起来像是一个多重背包求方案数的模板题，但是由于有  $n$  组，所以每次都求一次多重背包绝对会超时。

显然多重背包即有限制的方案数，直接计算有限制的方案数比较难，考虑**正难则反**：

**有限制的方案数 = 无限的方案数 - 超过限制方案数**

首先无限制的方案数显然就是完全背包，我们可以先用完全背包预处理出所有的方案数，即  $f[i]$  表示购买价值为  $i$  的方案数。

考虑减去超过显示的即不合法的部分，考虑使用容斥原理。

首先考虑一种硬币超额使用的方案数怎么计算。若第  $j$  种硬币超额使用，即超过了原定的  $d_j$  个硬币的限制，我们可以先强制选了  $d_j + 1$  个第  $j$  种硬币，这样剩下的价值里，4 种硬币随便选，这样就能保证第  $j$  种硬币一定超额使用，第  $j$  种硬币超额的不合法的方案数就等于  $f[s - (d_j + 1) \times c_j]$ 。

然后我们只需要使用容斥原理奇加偶减即可。加上一种硬币不合法的方案数，减去两种硬币不合法的方案数，加上三种硬币不合法的方案数  $\cdots$  二进制枚举子集即可。

## Code

```
1 // Problem: P1450 [HAOI2008]硬币购物
2 // Contest: Luogu
3 // URL: https://www.luogu.com.cn/problem/P1450
4 // Memory Limit: 125 MB
5 // Time Limit: 1000 ms
6 //
7 // Powered by CP Editor (https://cpeditor.org)
8
9 #include <bits/stdc++.h>
10 #define int long long
11 using namespace std;
12 const int N = 1e5 + 7;
13 int n, m, s, t, k, ans;
14 int f[N];
15 int c[50], d[50];
16
17 void pre_work(int n)
18 {
19     f[0] = 1;
20     for (int i = 1; i <= 4; ++ i)
21         for (int j = c[i]; j < n; ++ j)
22             f[j] += f[j - c[i]];
23 }
24
25 void solve()
26 {
27     scanf("%lld%lld%lld%lld", &d[1], &d[2], &d[3], &d[4], &s);
28
29     int ans = f[s], now;
30     for (int i = 1; i <= (1 << 4) - 1; ++ i) {
31         now = s;
32
33         int tmp, j, k;
34         for (tmp = i, j = 1, k = 0; tmp; tmp >>= 1, ++ j) {
35             if(tmp & 1) {
36                 k ^= 1;
37                 now -= (d[j] + 1) * c[j];
38             }
39         }
40         if(now >= 0)
41             k ? ans -= f[now] : ans += f[now];
42     }
43     printf("%lld\n", ans);
44 }
45
46 signed main()
47 {
48     scanf("%lld%lld%lld%lld", &c[1], &c[2], &c[3], &c[4], &n);
49     pre_work(N - 7);
50     while (n -- ){
51         solve();
52     }
53     return 0;
54 }
```

## Problem D. Coprime Subsequences (CF803F)

给定一个  $n$  个数的序列，问你有多少个子序列的  $\gcd = 1$ 。

$$n \leq 10^5$$

## Solution

序列一共有  $n$  个数，显然一共有  $2^n - 1$  个子序列（每个数选或不选减去空集）

考虑容斥。显然答案就是  $2^n - 1$  减去  $\gcd > 1$  的子序列个数，设所有含有大于 1 的因子的序列中的个数为  $x$ ，显然  $\gcd > 1$  的子序列的个数为  $2^x - 1$ 。显然只与点的权值有关，而  $a[i] \leq 10^5$ ，考虑维护权值。设序列中的数的最大值为  $m$ 。

- 设  $cnt_i$  表示权值为  $i$  的序列中的数的个数，可以在输入的时候处理一下。
- 设  $sum_i$  表示含有因子  $i$  的数的个数，显然  $sum_i = \sum_{i|j} cnt_j$ ，即序列中  $i$  的倍数的个数。我们可以通过枚举倍数在  $O(m \log m)$  的复杂度下计算。
- 设  $f_i$  表示含有因子  $i$  的子序列的个数，显然  $f_i = 2^{sum_i} - 1 = 2^{\sum_{i|j} cnt_j} - 1$ ，显然  $sum < m \leq 10^5$ ，我们可以  $O(m)$  预处理一下 2 的次幂。

对于  $\gcd > 1$  的子序列个数，根据奇加偶减的容斥原理，显然为：含有因子 2 的子序列的个数 ( $f_2$ ) + 含有因子 3 的子序列的个数 ( $f_3$ ) + 含有因子 5 的子序列的个数 ( $f_5$ ) +  $\dots$  - 含有因子 2, 3 的子序列的个数 ( $f_6$ ) - 含有因子 2, 5 的子序列的个数 ( $f_{10}$ ) -  $\dots$  + 含有因子 2, 3, 5 ( $f_{30}$ ) 的子序列的个数 +  $\dots$

最终的答案为  $2^n - 1$  减去  $\gcd > 1$  的子序列个数，即变为奇减偶加的形式。

但是如果我们继续使用二进制枚举选取状态，复杂度为  $O(2^n), n \leq 10^5$ ，显然不可做。

因此我们引入莫比乌斯函数：

$$\mu(n) = \begin{cases} 0 & \exists i \in [1, m], C_i > 1 \\ (-1)^m & \forall i \in [1, m], C_i = 1 \end{cases}$$

然后我们可以发现前面  $f_x$  的系数实际上就是  $\mu(x)$ （莫比乌斯函数本身就是一个容斥的映射）。

即答案为

$$2^n - 1 + \sum_{i=2}^m \mu(i) \times f_i$$

### Time

$$O(m \log m), m = \max\{a[i]\}$$

### Code

```

1 // Problem: CF803F Coprime Subsequences
2 // Contest: Luogu
3 // URL: https://www.luogu.com.cn/problem/CF803F
4 // Memory Limit: 250 MB
5 // Time Limit: 2000 ms
6 // Powered by CP Editor (https://cpeditor.org)
7
8 #include <bits/stdc++.h>
9 using namespace std;
10 const int N = 500007, mod = 1e9 + 7;
11
12 typedef long long ll;
13 int n, m, t;
14 int a[N], mu[N], cnt[N];
15 bool vis[N];
16 int primes[N], tot;
17 int pow2[N];
18 ll ans;
19
20 int add(int a, int b)
21 {
22     return 1ll * a + b >= mod ? 1ll * a + b - mod : 1ll * a + b;
23 }
24
25 int sub(int a, int b)
26 {
27     return a - b < 0 ? a - b + mod : a - b;
28 }
29
30 void init(int n)
31 {
32     pow2[0] = 1, pow2[1] = 2, mu[1] = 1;
33     for(int i = 2; i <= n; ++ i) {
34         pow2[i] = add(pow2[i - 1], pow2[i - 1]);
35         if(vis[i] == 0) {
36             primes[ ++ tot] = i;
37             mu[i] = -1;
38         }
39         for(int j = 1; j <= tot && i * primes[j] <= n; ++ j) {
40             vis[i * primes[j]] = true;
41             if(i % primes[j] == 0) {

```

```
42         mu[i * primes[j]] = 0;
43         break;
44     }
45     mu[i * primes[j]] -= mu[i];
46 }
47 }
48 }
49
50 int main()
51 {
52     scanf("%d", &n);
53     for(int i = 1; i <= n; ++ i) {
54         scanf("%d", &a[i]);
55         m = max(m, a[i]);
56         cnt[a[i]] ++ ;
57     }
58
59     init(N - 7);
60     ans = pow2[n] - 1;
61     for(int i = 2; i <= m; ++ i) {
62         int sum = 0;
63         for(int j = i; j <= m; j += i)
64             sum = (1ll * sum + cnt[j]) % mod;
65         ans = (ans + 1ll * mu[i] * (pow2[sum] - 1) % mod + mod) % mod;
66     }
67     printf("%lld\n", ans);
68     return 0;
69 }
```

## 0x16 RSA原理

本小节的内容涉及到后面章节的内容，且竞赛不会考察（谁知道呢），建议学本书之后再了解

如果要问我哪一种算法是最重要的，那么答案一定会包含公钥加密算法，因为它是计算机通信安全的基石，保证了加密数据不会被破解，想想你的**加密资源**被别人破解，那将是一个多么可怕的事情😓。

RSA加密算法是非对称加密算法中的一种，在1977年由罗纳德·李维斯特（Ron Rivest）、阿迪·萨莫尔（Adi Shamir）和伦纳德·阿德曼（Leonard Adleman）一起提出的，并取三人名字的首字母命名该算法。

RSA加密算法因其可靠的安全性（目前看来是十分安全的），得到了广泛的认可和使用，ISO（国际标准化组织）、ITU（国际电信联盟）及SWIFT（环球同业银行金融电讯协会）等国际标准组织均采用RSA作为加密标准，PGP等协议也采用RSA算法来传输会话密钥和数字签名等等。

RSA加密算法用到的数学知识有：素数，互质，欧拉函数，费马小定理，斐蜀定理，欧拉定理。（有没有发现全是数论的知识嘻嘻嘻）

### RSA原理

*RSA* 算法有三个参数， $n$ ,  $\text{pub}$ ,  $\text{pri}$ ，其中  $n$  等于两个大素数  $p$  和  $q$  的乘积 ( $n = p \times q$ )， $\text{pub}$  可以任意取，但是要求与  $(p - 1) * (q - 1)$  互素， $\text{pub} \times \text{pri} \% () = 1$  (可以理解为  $\text{pri}$  是  $\text{pub}$  的逆元)，那么这里的  $(n, \text{pub})$  称为公钥， $(n, \text{pri})$  称为私钥。 $(p - 1) * (q - 1)$

*RSA*算法的加密和解密是一致的，令  $x$  为明文， $y$  为密文，则：

- 加密： $y = x^{\text{pub}} \% n$  （利用公钥加密， $y = \text{encode}(x)$ ）
- 解密： $x = y^{\text{pri}} \% n$  （利用私钥解密， $x = \text{decode}(y)$ ）

那么我们来看看这个算法是如何运作的。

假设你得到了一个密文  $y$ ，并且手上只有公钥，如何得到明文  $x$ ，从  $\text{decode}$  的情况来看，只要知道私钥貌似就可以了，而私钥的获取方式只有一个，就是求公钥对  $(p - 1) * (q - 1)$  的逆元，如果  $(p - 1) * (q - 1)$  已知，那么可以利用扩展欧几里德定理进行求解，问题是  $(p - 1) * (q - 1)$  是未知的，但是我们有  $n = p * q$ ，于是问题归根结底其实是难在了对  $n$  进行素因子分解上了，Pollard - rho 的分解算法时间复杂度只能达到  $O(n^{\frac{1}{4}})$ ，对 `int_64` 范围内的整数可以在几十毫秒内出解，而当  $n$  是几百位、几千位的大数的时候，计算时间就是一个天文数字了，以此达到加密的作用。

## 0x20 同余

### 0x21 整数的取余运算

#### 0x21.1 整数的取余运算（模运算）

**定义：**带余除法，设  $a, b$  是整数，且  $b > 0$ ，使得  $a = bq + r$ ，且  $0 \leq r < b$ ，称  $q$  为商， $r$  为余数。

显然带余除法中的商和余数都是唯一的，在下文中将商记为  $a/b$ ，将余数记为  $a \% b$ ，`/` 与 `%` 的运算优先级与乘除法相同。



定义以下运算：

- **取模运算**： $a \% p$  (或  $a \bmod p$ )，表示  $a$  除以  $p$  的余数。
- **模  $p$  加法**： $(a + b) \% p$ ，其结果是  $a + b$  算术和除以  $p$  的余数，也就是说， $(a + b) = kp + r$ ，则  $(a + b) \% p = r$ 。
- **模  $p$  减法**： $(a - b) \% p$ ，其结果是  $a - b$  算术差除以  $p$  的余数。
- **模  $p$  乘法**： $(a * b) \% p$ ，其结果是  $a * b$  算术乘法除以  $p$  的余数。

模运算有如下简单性质：

- $a \% b = a - b \times \lfloor \frac{a}{b} \rfloor$ ，即若  $a \% b = c$ ，则  $a = bx + c$ ， $x = \lfloor \frac{a}{b} \rfloor$ 。
- $n \% p$  得到结果的正负由被除数  $n$  决定，与  $p$  无关。

例如： $7 \% 4 = 3$ ， $-7 \% 4 = -3$ ， $7 \% -4 = 3$ ， $-7 \% -4 = -3$

- 若  $p \mid (a - b)$ ，则  $a \equiv b \pmod{p}$ 。例如  $11 \equiv 4 \pmod{7}$ ， $18 \equiv 4 \pmod{7}$  (符号 `int` 指整除)
- **结合率**： $((a + b) \% p + c) \% p = (a + (b + c) \% p) \% p$ ， $((a \times b) \% p \times c) \% p = (a \times (b \times c) \% p) \% p$
- **交换率**： $(a + b) \% p = (b + a) \% p$ ， $(a \times b) \% p = (b \times a) \% p$
- **分配率**： $((a + b) \% p \times c) \% p = ((a \times c) \% p + (b \times c) \% p) \% p$
- 若  $a \% p = x$ ， $a \% q = x$ ， $\gcd(p, q) = 1$ ，则  $a \% (p \times q) = x$

更多关于模运算性质以及同余详见本文 **0x22同余**。

0x21.2 整数模意义下的加减乘乘方运算

- $(a + b) \% c = (a \% c + b \% c) \% c$
- $(a - b) \% c = (a \% c - b \% c + c) \% c$
- $(a \times b) \% c = (a \% c) \times (b \% c) \% c$
- $(a^b) \% p = ((a \% p)^b) \% p$

计算减法的时候，通常需要加上模数  $c$ ，防止出现负数。

- **经典快速幂**

给定整数  $a$ ，正整数  $n$ ，以及非零整数  $p$ ，求  $a^n \% p$ 。利用模  $p$  乘法，这个问题可以递归求解，即令  $f(n) = a^n \% p$ ，那么  $f(n - 1) = a^{n-1} \% p$ ， $f(n) = a \times f(n - 1) \% p$ ，这样就转化成了递归式。但是递归求解的时间复杂度为  $O(n)$ ，往往当  $n$  很大的时候就很难在规定时间内得到解了。

当  $n$  为偶数时，我们可以将  $a^n \% p$  拆成两部分，令  $b = a^{\frac{n}{2}} \% p$ ，则  $a^n \% p = b \times b \% p$ 。

当  $n$  为奇数时，可以拆成三部分，令  $b = a^{\frac{n}{2}} \% p$ ，则  $a^n \% p = a \times b \times b \% p$ ；

上述两个等式中的  $b$  可以通过递归计算，由于每次都是除 2，所以时间复杂度是  $O(\log n)$ 。

```
1 ll qpow(ll a, ll b, ll q)
2 {
3     ll res = 1; // 因为是用乘法模拟乘方，所以res要是1
4     while(b) {
5         if(b & 1) res = (res * a) % q;
6         a = (a * a) % q; // 视情况将 * 换成Mul(龟速乘)
7         b >>= 1;
8     }
9     return res % q;
10 }
```

- **龟速乘**

在模意义下计算乘法，如果  $c$  较大（但是不超过 `long long` 范围），进行乘法的两个数同样很大，直接**乘会爆掉**（例如快速幂里的乘法），我们可以用类似快速幂的快速乘计算，时间复杂度为  $O(\log b)$ 。因为慢于  $O(1)$  的乘法运算符，所以我们常常把这个叫做龟速乘。经常用与快速幂中代替普通乘法。

```
1 ll Mul(ll a, ll b, ll p)
2 {
3     if(b < 0) a = -a, b = -b;
4     ll res = 0; // 因为是加法模拟乘法，所以res开始为0
5     while(b) {
6         if(b & 1) res = (res + a) % p;
7         a = (a + a) % p;
8         b >>= 1;
9     }
10    return res;
11 }
```

这样普通的快速幂会变成  $O(\log^2 n)$ 。如果需要更快的快速乘，可以用 `long double` 数据类型进行计算，复杂度  $O(1)$ 。（`long double` 有 15/12 bit，可以处理范围在  $1.7 \times 10^{-308} \sim 1.7 \times 10^{308}$  的数据）

```

1 ll Mul(ll a, ll b, ll p) {
2     if(b < 0) a = - a, b = - b;
3     if (p <= 1000000000) return a * b % p;
4     else if ( p <= 100000000000011) return (((a * (b >> 20) % p) << 20) + (a * (b & ((1 << 20) - 1)))) % p;
5     else {
6         ll d = (ll)floor(a * (long double)b / p + 0.5);
7         ll res = (a * b - d * p) % p;
8         if (res < 0) res += p;
9         return res;
10    }
11 }

```

或者换一种好写的方法：

```

1 inline ll Mul(ll x,ll y,ll p)
2 {
3     if(y < 0) x = - x, y = - y;
4     ll z = (long double)x / p * y;
5     ll res = (unsigned long long)x * y - (unsigned long long)z * p;
6     return (res + p) % p;
7 }

```

$O(n \log n)$  的大数快速幂：

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 const int mod = 1e9 + 7;
4 long long quick_mod(long long a, long long b) {
5     long long ans = 1;
6     while (b) {
7         if (b & 1) {
8             ans = (ans * a) % mod;
9             b--;
10        }
11        b /= 2;
12        a = a * a % mod;
13    }
14    return ans;
15 }//内部快速幂
16 long long quickmod(long long a, char *b, int len) {
17     long long ans = 1;
18     while (len > 0) {
19         if (b[len - 1] != '0') {
20             int s = b[len - 1] - '0';
21             ans = ans * quick_mod(a, s) % mod;
22         }
23         a = quick_mod(a, 10) % mod;
24         len--;
25     }
26     return ans;
27 }
28
29 int main() {
30     char s[100050];
31     int a;
32     while (~scanf("%d", &a)) { //a ^ s % mod
33         scanf("%s", s);
34         int len = strlen(s);
35         printf("%I64d\n", quickmod(a, s, len));
36     }
37     return 0;
38 }

```

## 0x22 同余

若正整数  $a$  和  $b$  除以  $m$  的余数相等，则称  $a, b$  模  $m$  同余，记作  $a \equiv b \pmod{m}$ 。

即  $a \% m = b \% m$ 。

• 竞赛例题选讲

Problem A 循环节

$f[1] = a, f[2] = b, f[3] = c$ , 当  $n > 3$  时  $f[n] = (A \times f[n-1] + B \times f[n-2] + C \times f[n-3]) \% 53$ , 给定  $a, b, c, A, B, C$ , 求  $f[n]$  ( $n < 2^{31}$ )。

Solution

由于  $n$  非常大, 循环模拟求解肯定是不现实的, 仔细观察可以发现当  $n > 3$  时,  $f[n]$  的值域为  $[0, 53)$ , 并且连续三个数  $f[n-1]$ 、 $f[n-2]$ 、 $f[n-3]$  一旦确定, 那么  $f[n]$  也就确定了, 而  $f[n-1]$ 、 $f[n-2]$ 、 $f[n-3]$  这三个数的组合数为  $53 \times 53 \times 53$  种情况, 那么对于一个下标  $k < n$ , 假设  $f[k]$  已经求出, 并且满足  $f[k-1] == f[n-1]$  且  $f[k-2] == f[n-2]$  且  $f[k-3] == f[n-3]$ , 则  $f[n]$  必定等于  $f[k]$ , 这里的  $f[k \cdots n-1]$  就被称为这个数列的循环节。

显然, 在  $53 \times 53 \times 53$  次计算之内必定能够找到循环节。打表找规律即可。

21.1 同余的性质

同余的基本性质：

性质21.1.1：(自反性)：  $a \equiv a \pmod{m}$

性质21.1.2：(对称性)：若  $a \equiv b \pmod{m}$ , 则  $b \equiv a \pmod{m}$

性质21.1.3：(传递性)：若  $a \equiv b \pmod{m}, b \equiv c \pmod{m}$ , 则  $a \equiv c \pmod{m}$

性质21.1.4：(同加性)：若  $a \equiv b \pmod{m}$ , 则  $a \pm c \equiv b \pm c \pmod{m}$

性质21.1.5：(同乘性)：若  $a \equiv b \pmod{m}$ , 则  $a \times c \equiv b \times c \pmod{m}$ , 若  $a \equiv b \pmod{m}, c \equiv d \pmod{m}$ , 则  $a \times c \equiv b \times d \pmod{m}$

性质21.1.6：(同幂性)：若  $a \equiv b \pmod{m}$ , 则  $a^c \equiv b^c \pmod{m}$

性质21.1.7：(不满足同除性)：若  $a \equiv b \pmod{m}$  不满足  $a \div c \equiv b \div c \pmod{m}$

性质21.1.8：(满足同除性)：若  $a \equiv b \pmod{m}, c \mid a, c \mid b$ , 则  $\frac{a}{c} \equiv \frac{b}{c} \pmod{\frac{m}{\gcd(m, c)}}$ 。

或者可以换一种表述方式：若  $ca \equiv cb \pmod{m}$  则  $a \equiv b \pmod{\frac{m}{\gcd(m, c)}}$

例如：  $ca \equiv cb \pmod{m}, \gcd(c, m) = 1 \implies a \equiv b \pmod{m}$

该性质会在取遍剩余系会用到。

推论21.1.9：若  $a \equiv b \pmod{m}, m' \mid m, a \equiv b \pmod{m'}$

大致证明 (随便写的, 不严谨, 大致是这么个意思)

若  $a \equiv b \pmod{m}$  显然有:  $a = k_1 m + b$  若  $m' \mid m$  则  $m = k_2 m'$ , 则  $a = k_1 k_2 m' + b$

若  $b \leq m'$   $a \pmod{m'} = k_1 k_2 m' + b \pmod{m'} = b$  即:  $a \equiv b \pmod{m'}$

若  $b > m'$  设  $b = k_3 m' + r$   $a \pmod{m'} = k_1 k_2 m' + k_3 m' + r = r$   $b \pmod{m'} = k_3 m' + r = r$  即:  $a \equiv b \pmod{m'}$

综上所述, 若  $a \equiv b \pmod{m}, m' \mid m$ , 则  $a \equiv b \pmod{m'}$

推论21.1.10：  $a \equiv b \pmod{m_i} (i = 1..k)$  等价于  $a \equiv b \pmod{M} = \text{lcm}(m_1, m_2, \dots, m_k)$

推论21.1.12：  $\begin{cases} a \equiv b \pmod{m} \\ c \equiv d \pmod{m} \end{cases} \implies a + c \equiv b + d \pmod{m}$

推论21.1.13：  $\begin{cases} a \equiv b \pmod{m} \\ c \equiv d \pmod{m} \end{cases} \implies a - c \equiv b - d \pmod{m}$

注意：  $-4 \% 5 = -4, -6 \% 5 = -1$ , 所以如果题目要求的是最小正整数那么我们就需要对答案x:  $(x \% b + b) \% b$

同余类与剩余系

对于  $\forall a \in [0, m-1]$ , 集合  $\{a + km\} (k \in \mathbb{Z})$  的所有数模  $m$  同余, 余数都是  $a$ , 该集合称为一个模  $m$  的**同余类**, 简记为  $\bar{a}$ 。

模  $m$  的同余类显然一共有  $m$  个, 分别为  $\bar{0}, \bar{1}, \bar{2}, \dots, \overline{m-1}$ , 他们构成了  $m$  的**完全剩余系**

$1 \sim m$  中与  $m$  互质的数代表的同余类共有  $\varphi(m)$ , 他们构成了  $m$  的**简化剩余系**。

例如模 10 的完全剩余系为  $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ , 模 10 的简化剩余系为  $\{\bar{1}, \bar{3}, \bar{7}, \bar{9}\}$ 。

简化剩余系关于模  $m$  乘法封闭, 因为若  $a, b (1 \leq a, b \leq m)$  与  $m$  互质, 则  $a \times b$  显然也不会含有与  $m$  相同的质因子, 即  $a \times b$  与  $m$  互质。由余数的定义可得到  $a \times b \pmod{m}$  也与  $m$  互质, 即  $a \times b \pmod{m}$  也属于  $m$  的简化剩余系。

即模  $m$  的简化剩余系关于模  $m$  乘法封闭。

• 重要性质定理

性质21.11：模  $p$  的剩余系  $\{rk\}$ , 若  $\gcd(p, d) = 1$  那么  $\{d \times rk\}$  也是模  $p$  的剩余系

很重要的性质, 如果求  $ax \pmod{p}$  的既约剩余系的大小, 我们只要利用同余式的性质1

转化为  $x \times \frac{a}{\gcd(a,p)} \pmod{\frac{p}{\gcd(a,p)}}$ , 根据这个理论  $x$  取值个数就是  $\frac{p}{\gcd(a,p)}$

**性质21.12**： 设模  $m_1$  的既约剩余系为  $R_1, m_2$  的既约剩余系为  $R_2, m_1, m_2$  互质, 则模  $m_1 m_2$  的既约剩余系为  $R = \{x = m_2 x_1 + m_1 x_2 \pmod{m_1 m_2} | x_1 \in R_1, x_2 \in R_2\}$

这个结论证明可以考虑任意两个  $x$  互不同余即可 (作差判断)

这就告诉我们若  $S(M)$  表示  $M$  既约剩余系的大小的话,  $S(M) = S(m_1) \times S(m_2)$

这个结论推广到  $k$  维就是, 设模  $m_k$  的既约剩余系为  $R_k, m_k$  两两互质,  $M = \prod m_i$  则模  $M$  的既约剩余系为  $R = \{\sum M \times m_i^{-1} \times a_i \pmod{M} | x_i \in R_i\}$  ( $m_i^{-1}$  指  $m_i$  的逆元)

因此以后我们求模  $M$  的剩余系大小, 可以对  $M$  质因数分解分别求出后再相乘。

## 0x21.2 费马小定理

若  $p$  是质数, 则对于任意的整数  $a$  都有  $a^p \equiv a \pmod{p}$ 。若  $\gcd(a, p) = 1$ , 即  $a$  不是  $p$  的倍数, 则有  $a^{p-1} \equiv 1 \pmod{p}$ 。

证明

因为  $p$  是质数, 且  $(a, p) = 1$ , 所以  $\varphi(p) = p - 1$ 。

由欧拉定理可得  $a^{p-1} \equiv 1 \pmod{p}$ 。证毕。(欧拉定理证明见下一小节)

对于该式又有  $a^p \equiv a \pmod{p}$ , 而且此式不需要  $(a, p) = 1$ ,

所以, 费马小定理的另一种表述为: 假如  $p$  是质数,  $a$  是整数, 那么  $a^p \equiv a \pmod{p}$ 。

费马小定理降幂:  $a^k \equiv a^{k \bmod (p-1)} \pmod{p}$  ( $a$  与  $p$  互质)

费马大定理:

- $m > 2$  时,  $x^m + y^m = z^m$  无正整数解
- 当  $m = 2$ , 对于式子  $a^2 + b^2 = c^2$  ( $n$  为任意正整数):
  - 当  $a$  为奇数时:  $a = 2n + 1, c = n^2 + (n + 1)^2, b = c - 1$
  - 当  $a$  为偶数时:  $a = 2n + 2, c = 1 + (n - 1)^2, b = c - 2$

**性质21.2.1**: 对于任意多项式  $F(x) = \sum_{i=0}^{\infty} a_i x^i$ , ( $a_i$  对一个质数  $P$  取模), 若满足  $a_0 \equiv 1 \pmod{P}$ , 则  $\forall n \leq P, F^P(x) \equiv 1 \pmod{x^n}$  (例题)

## 0x21.3 欧拉定理

- 欧拉定理

**定理21.3.1**: 若正整数  $a, n$  互质, 则  $a^{\varphi(n)} \equiv 1 \pmod{n}$  其中  $\varphi(n)$  是欧拉函数。

证明

设  $x_1, x_2, \dots, x_{\varphi(n)}$  是一个以  $n$  为模的简化剩余系, 则  $ax_1, ax_2, \dots, ax_{\varphi(n)}$  也是一个以  $n$  为模的简化剩余系 (因为  $(a, n) = 1$ )。于是有  $ax_1, ax_2, \dots, ax_{\varphi(n)} \equiv x_1, x_2, \dots, x_{\varphi(n)} \pmod{n}$ , 所以  $a^{\varphi(n)} \equiv 1 \pmod{n}$ 。

证毕。

**推论21.3.2**:  $\exists x \in N^*, a^x \equiv 1 \pmod{m} \iff \gcd(a, m) = 1$  (证明) (例题)

- 竞赛例题选讲

### Problem A 好大好大好大好大

整数  $a$  和  $n$  互素, 求  $a$  的  $k$  次幂模  $n$ , 其中  $k = X^Y$ , 正整数  $a, n, X, Y$  ( $X, Y \leq 10^9$ ) 为给定值。

#### Solution

好大好大, 问题要求的是  $a^{k \% n}, k = X^Y$ , 指数上还是存在指数, 需要将指数化简, 注意到  $a$  和  $n$  互素, 所以可以利用欧拉定理, 令  $X^Y = k\varphi(n) + r$ , 那么  $k\varphi(n)$  部分并不需要考虑, 因为他们都与  $1$  模  $n$  同余。问题转化成求  $r = X^Y \% \varphi(n)$ , 可以采用快速幂取模, 得到  $r$  后再采用快速幂取模求解  $a^r \% n$ 。这种思想其实就是拓展欧拉定理, 也叫欧拉降幂。

- 欧拉降幂 (拓展欧拉定理)

若  $a$  与  $m$  互质:  $a^b \equiv a^{b \bmod \varphi(m)} \pmod{m}$

证明:

设  $b = q \times \varphi(m) + r$ , 其中  $0 \leq r < \varphi(m)$ , 即  $r = b \bmod \varphi(m)$ 。

$a^b \equiv a^{q \times \varphi(m) + r} \equiv (a^{\varphi(m)})^q \times a^r \equiv 1^q \times a^r \equiv a^r \equiv a^{b \bmod \varphi(m)} \pmod{m}$

定理得证  $\square$

若不保证  $a$  与  $m$  互质:  $b > \varphi(m)$  时:  $a^b \equiv a^{b \bmod \varphi(m) + \varphi(m)} \pmod{m}$

太长不证, 证明详见: <https://www.cnblogs.com/1024th/p/11349355.html>

在一些计数的问题中, 常常要求对结果取模, 但是在计算非常庞大的次幂的时候, 无法直接取模, 可以先把底数对  $p$  取模, 指数对  $\varphi(p)$  取模, 再计算次幂, 有效地降低时间复杂度。

(模板)

计算  $a^b \bmod m$ ,  $1 \leq a \leq 10^9$ ,  $1 \leq b \leq 10^{20000000}$ ,  $1 \leq m \leq 10^8$ **Code**

```

1 int main()
2 {
3     scanf("%d%d", &a, &m);
4     bool flag = false;
5     int phi_m = getphi(m);
6     cin >> s;
7     for(int i = 0; i < s.length(); ++ i) {
8         b = (b * 10 + s[i] - '0');
9         if(b >= phi_m)
10             flag = 1, b %= phi_m;
11     }
12     if(flag)
13         b += phi_m;
14     int ans = qpow(a, b, m);
15     printf("%d\n", ans);
16     return 0;
17 }

```

**竞赛例题选讲****Problem A 上帝与集合的正确用法 (P4139 )** $2^{2^{2^{\cdots}}} \bmod p$ **Solution**

根据拓展欧拉定理有：

$$2^{2^{2^{\cdots}}} \bmod p = 2^{((2^{2^{\cdots}}) \bmod \varphi(p)) + \varphi(p)} \bmod p$$

然后递归求解即可。

**Code**

```

1 int n, m, mod = 100, p, t;
2 int primes[N];
3 int cnt, phi[N];
4 bool vis[N];
5 void init(int n)
6 {
7     phi[1] = 1;
8     for(int i = 2; i <= n; ++ i) {
9         if(vis[i] == 0) primes[ ++ cnt] = i, phi[i] = i - 1;
10        for(int j = 1; j <= cnt && i * primes[j] <= n; ++ j) {
11            vis[i * primes[j]] = true;
12            if(i % primes[j] == 0) {
13                phi[i * primes[j]] = phi[i] * primes[j];
14                break;
15            }
16            phi[i * primes[j]] = phi[i] * (primes[j] - 1);
17        }
18    }
19 }
20
21 int qpow(int a, int b, int mod)
22 {
23     int res = 1;
24
25     while(b) {
26         if(b & 1) res = 1ll * res * a % mod;
27         a = 1ll * a * a % mod;
28         b >>= 1;
29     }
30     return res;
31 }
32 int tower(int a, int n)
33 {
34     if(n == 1) return a;
35     return qpow(a, tower(a, n - 1), phi[n]);
36 }

```



```
37
38 void solve()
39 {
40     scanf("%d", &p);
41     mod = p;
42     int ans = tower(2, p);
43     printf("%d\n", ans);
44 }
45 int main()
46 {
47     init(N - 7);
48     scanf("%d", &t);
49     while(t -- ) {
50         solve();
51     }
52     return 0;
53 }
```

0x21.4 威尔逊定理

威尔逊定理

**定理21.4.1:** 当  $p$  为质数时有:  $(p - 1)! \equiv p - 1 \equiv -1 \pmod{p}$ ,  $(p - 2)! \equiv 1 \pmod{p}$

其中 **定理21.4.1** 实际上就等价于: 若  $p$  是质数, 则  $(p - 1)! + 1$  能够被  $p$  整除。

$n$  为素数时:  $(n - 1)! \pmod{n} = 1$

$n$  为合数时: 除  $n = 4$  以外,  $(n - 1)! \pmod{n} = 0$

威尔逊定理的逆命题

**定理21.4.2:** 若一个数  $p$ , 满足条件  $(p - 1)! + 1$  可以被  $p$  整除, 那么  $p$  是素数。

即:  $p$  可整除  $(p - 1)! + 1$  是  $p$  为质数的充要条件。

Problem A Faulty Factorial (CERC2017 F)

给定三个数,  $n, p, r$  ( $p > r$ ,  $r$  是余数), 在  $n$  的阶乘  $1 \times 2 \times 3 \times 4 \times \cdots \times n$ , 这个连乘式中的  $n$  个数里, 找到一个数  $k$  ( $2 \leq k \leq n$ ), 将  $k$  换成比  $k$  小的数  $v$  ( $1 \leq v < k$ ), 使得换完之后的连乘式的结果  $res \equiv r \pmod{p}$ 。请你输出任意一组  $k, v$ , 若找不到答案则输出 -1 -1

Input

1 4 5 1

Output

1 3 2

样例解释:  $n = 4$ ,  $n! = 1 \times 2 \times 3 \times 4$ , 将其中的 3 换成 2, 则连乘式的结果变为  $1 \times 2 \times 2 \times 4 = 16 \equiv 1 \pmod{5}$ , 故输出 3 2

Solution

看上去像是一个同余方程, 要求输出一种方案 -> 拓展欧几里德解同余方程得到一组解 / 暴力枚举判断得到一组解...

因为  $n \leq 10^{18}, p \leq 10^7$ ,  $n$  很大, 经验告诉我们, 当  $n$  超过一定的限制之后一定只有一种答案即一定是特解 (盲猜特别大的时候无解), 不然就真的没法玩了。

所以先从分析数据开始入手。显然可以发现:

若  $n \geq 2p$  时,  $n!$  一定含有两个及以上的  $p$ , 而我们只能修改换掉一个  $p$ , 故  $n!$  中一定含有  $p$ , 即  $n!$  是  $p$  的倍数, 则  $n! \equiv 0 \pmod{p}$ , 也就是若  $r! = 0$  就无解, 若  $r = 0$ , 任意修改一个数即可。

有了这个分析之后, 我们就有了一个分类讨论的解题思路。

显然剩余的情况还有两种:

若  $p \leq n < 2p$ , 显然当  $r$  等于 0 的时, 我们任意修改一个, 只要不修改  $p$  即可。当  $r$  不等于 0 的时, 我们必须修改  $p$ , 不然  $\pmod{p}$  一定等于 0。所以我们可以直接暴力枚举  $p$  修改成的数 ( $p \leq 10^7$ ,  $O(p)$  的复杂度没问题), 若膜  $p$  之后能和  $r$  匹配, 则输出修改方案, 否则输出 -1 -1。

若  $n = p$  我们可以使用威尔逊定理, 但没必要, 上面枚举判断就够了。

若  $n < p$ , 因为  $n \leq p \leq 10^7$ , 所以我们可以直接暴力枚举  $k$ , 即题目中要求的修改的点的下标显然其权值就是其下标。对于每一个位置, 其修改的值  $v$ , 显然有

$$r \equiv \frac{n! \times v}{k} \pmod{p}$$

$$v \equiv \frac{r \times k}{n!}$$



因为有两个自变量，却只有一个同余方程，故不能通过拓展欧几里得算法求出特解，所以只能暴力，好在数据小，只有  $10^7$ ，我们是可以承受得了直接暴力枚举判断的。

那么暴力跑，我们可以直接预处理  $n!$  的逆元，然后乘起来计算  $v$ ，然后判断当前的这个  $v$  是否满足题意即可。即判断当前得到的  $v$  合法，即若  $v < k$  且  $v \geq 1$  则合法，直接输出即可。

最后注意多个数相乘的时候记得多取模数，不然你会因爆 `long long` 而 wa 71...

## Code

```

1 ll n, p, r;
2 ll inv[N];
3
4 void init()
5 {
6     inv[1] = 1;
7     for(ll i = 2; i <= p; ++ i) {
8         inv[i] = (p - (p / i)) * inv[p % i] % p;
9     }
10 }
11
12 void solve()
13 {
14
15     scanf("%lld%lld%lld", &n, &p, &r);
16     init();
17     if(n >= 2 * p) {
18         if(r == 0) {
19             printf("2 1\n");
20         }
21         else puts("-1 -1");
22     }
23     else if(n >= p) {
24         if(r == 0) {
25             bool ok = 0;
26             //if(p == 2)
27             for(ll i = 2; i <= n; ++ i) {
28                 if(i != p) {
29                     printf("%lld 1", i);
30                     return ;
31                 }
32             }
33             puts("-1 -1");
34             return ;
35         }
36         else {
37             ll fact = 1;
38             for(int i = 1; i <= n; ++ i) {
39                 if(i == p) continue;
40                 fact = (fact * i) % p;
41             }
42             for(ll i = 1; i < p; ++ i) {
43                 if((fact * i) % p == r) {
44                     printf("%lld %lld", p, i);
45                     return ;
46                 }
47             }
48             puts("-1 -1");
49             return ;
50         }
51     }
52     else if(n < p) {
53         ll fact_inv = 1;
54         for(int i = 1; i <= n; ++ i) {
55             fact_inv = (fact_inv * inv[i]) % p;
56         }
57         for(ll k = 1; k <= n; ++ k) {
58             ll v = r * k % p * fact_inv % p;
59             if(v >= 1 && v < k) {
60                 printf("%lld %lld", k, v);
61                 return ;

```

```

62     }
63     }
64     puts("-1 -1");
65 }
66 return ;
67 }
68
69 int main()
70 {
71     solve();
72 }

```

### Problem B. Fansblog (HDU 6608 19多校)

给定一个质数  $P(10^9 \leq P \leq 10^{14})$ ,  $Q$  是 最大的那个小于  $P$  的质数, 求  $Q! \% P$ 。

### Solution

根据威尔逊定理  $(P-1)! \equiv P-1 \equiv -1 \pmod{P}$ , 所以显然可以构造答案  $Q!$

$$Q! = \frac{(P-1)!}{(P-1) \times (P-2) \times \dots \times (P-Q)}$$

上面就是  $-1$  或者  $P-1$ , 因为题目中求的是正整数, 所以取  $P-1$  即可。下面直接求逆元即可。

由素数分布定理得, 两个素数之间的距离最大不超过 300, 所以直接暴力找上一个素数即可。因为数据范围  $P \leq 10^{14}$ , 我们每次暴力判断素数需要  $O(\sqrt{n})$ , 我们可以先预处理小于  $\sqrt{10^{14}} = 10^7$  的质数, 只有 6e5 个, 这样我们可以直接用质数试除法判断即可。并且因为数据较大, 所以求逆元乘的时候需要用到快速乘。

注意如果传参数多 (多传一个 `mod`) 的话会 T...改了就A了。不然就只能用 Miller-Rabin 判定法了。

### Code

```

1  ll mod;
2  int n, m;
3  ll p, cnt;
4  ll q;
5  bool vis[N];
6  int primes[N];
7
8  void init(int n)
9  {
10     for(int i = 2; i <= n; ++ i) {
11         if(vis[i] == 0) primes[ ++ cnt] = i;
12         for(int j = 1; j <= cnt && i * primes[j] <= n; ++ j) {
13             vis[i * primes[j]] = true;
14             if(i % primes[j] == 0) break;
15         }
16     }
17 }
18
19 bool is_primes(ll x)
20 {
21     for(int i = 1; i <= cnt && 1ll * primes[i] * primes[i] <= x; ++ i) {
22         if(x % primes[i] == 0) return false;
23     }
24     return true;
25 }
26
27 ll mul(ll a, ll b)
28 {
29     if(b < 0) a = - a, b = - b;
30     ll res = 0;
31     while(b) {
32         if(b & 1) res = (res + a) % mod;
33         a = (a + a) % mod;
34         b >>= 1;
35     }
36     return res;
37 }
38
39 ll qpow(ll a, ll b)
40 {
41     ll res = 1;
42     while(b) {
43         if(b & 1) res = mul(res, a);

```

```
44     a = mul(a, a);
45     b >>= 1;
46 }
47 return res;
48 }
49
50 ll inv(ll x)
51 {
52     return qpow(x, mod - 2);
53 }
54
55 void solve()
56 {
57     scanf("%lld", &p);
58     mod = p;
59     q = p - 1;
60     ll ans = p - 1;
61     while(is_primes(q) == false)
62         q -- ;
63     q ++ ;
64     while(q < p)
65         ans = mul(ans, inv(q)), q ++ ;
66     printf("%lld\n", ans);
67     return ;
68 }
69
70 int main()
71 {
72     int t;
73     init(N - 7);
74     scanf("%d", &t);
75     while(t -- ) {
76         solve();
77     }
78 }
```

## Ox22 拓展欧几里德

### Ox22.1 裴蜀（Bézout）定理

**定理22.1.1**：设  $a, b$  是不全为零的整数，存在无穷多组整数对  $(x, y)$ ，满足不定方程  $ax + by = d$ ，其中  $d = \gcd(a, b)$  即：  $ax + by = \gcd(a, b)$ 。

**推论22.1.2**：  $\gcd(a, b) \mid c \iff \exists x, y \in \mathbb{Z}, ax + by = c$

方程  $ax + by = d (d = \gcd(a, b))$  即为丢番图方程。

**推论21.1.3**：  $\forall a, b, z \in \mathbb{N}^*, \gcd(a, b) = 1, \exists x, y \in \mathbb{N}, ax + by = ab - a - b + z$ ，即两互质的数  $a, b$ ，表示不出的最大的数为  $ab - a - b$ 。

**推论21.1.3证明**：不妨设  $a < b$ ，假设答案为  $x$ 。

若  $x \equiv ma \pmod{b} (1 \leq m \leq b - 1)$  (若  $m \geq b$ ,  $m \equiv 0 \pmod{b}$ )

即  $x = ma + nb (1 \leq m \leq b - 1)$

显然当  $n \geq 0$  时  $x$  可以用  $a, b$  表示出来，不合题意。

因此当  $n = -1$  时  $x$  取得最大值，此时  $x = ma - b$ 。

显然当  $m$  取得最大值  $b - 1$  时  $x$  最大，此时  $x = (b - 1)a - b = ab - a - b$

即  $a, b$  所表示不出的最大的数是  $ab - a - b$

#### • 竞赛例题选讲

#### Problem A Fox And Jumping (CF510D)

给出  $n$  张卡片，分别有  $l_i$  和  $c_i$ 。在一条无限长的纸带上，你可以选择花  $c_i$  的钱来购买卡片  $i$ ，从此以后可以向左或向右跳  $l_i$  个单位。问你至少花多少元钱才能够跳到纸带上全部位置（整数数轴）。若不行，输出  $-1$ 。  $1 \leq n \leq 300, 1 \leq l_i, c_i \leq 10^9$ 。

#### Solution （思路较长，比较详细，建议看完）

首先分析子问题，先考虑两个数的情况，因为纸带是无限长的，没有循环，我们发现，要想能够跳到每一个格子上，就必须使得我们选择的数通过数次加减得到的数的绝对值为 1，进而想到了裴蜀定理。

我们知道裴蜀定理的内容是：**设  $a, b$  是不全为零的整数，则存在整数  $x, y$ ，使得  $ax + by = \gcd(a, b)$** 。 我们想要解这个二元一次方程得到答案 1，也就是使得  $\gcd(a, b) = 1$ ，我们可以推出：如果  $a$  与  $b$  互质，则一定存在两个整数  $x, y$ ，使得  $ax + by = 1$ 。

由此我们想到本题的解题思路：选择最便宜的两个或者多个**互质**的数（多个数互质是指  $\gcd(\gcd(\gcd(a, b), c), \dots) = 1$ ）。考虑动态规划。但是我们发现数据的大小达到了  $10^9$ ，我们的动态规划开不了这么大的数组，并且时间复杂度上也会很糟糕（尽管动规实际上是通过本题的hhh，如果可以把数据范围中的  $n = 300$  上调至 500000，可以卡掉除本思路以外的所有做法，那么这道题我将绝杀，可惜调不得 ~）。那么考虑有没有其他的做法。

我们知道动态规划问题实际上就是 DAG 上的递推，动态规划对于状态空间的遍历构成了一张有向无环图 DAG，遍历顺序就是该 DAG 的一个拓扑序。考虑用图论的做法解决这个问题。因为我们想要找到的是最小代价的两个互质的数，从 0 开始出发，选取最小的代价，最后的终点是  $\gcd(a, b) = 1$  或者是  $\gcd(\gcd(\gcd(a, b)c \dots)) = 1$ ，我们发现有一点最短路的感觉。我们发现多个数互质，实际上就是一步步的 gcd 最后推到 1，就类似我们最短路的一步一步的结点转移。最后考虑从 0 出发会不会对我们求解 gcd 造成影响呢？由于我们知道  $\gcd(0, x) = x$ ，所以不会有任何的影响。

至此整体的思路已经非常清晰了：我们从 0 号节点开始，每一步走到的结点编号为  $\gcd(x, y)$ ，其中  $x$  为当前的结点编号，也就是当前已选择的数的总 gcd， $y$  为我们枚举到的下一个结点，也就是下一个待选的数的值  $l[i]$ ，与此同时更新代价，利用 `dijkstra` 算法求得最小的总代价。

最后如果能够到达结点 1，也就意味着我们能够找到若干个使得他们的总 gcd 为 1，输出 `dist[1]` 即可。反之说明无法得到，即无法遍历所有的格点。

最后的一点小细节：由于我们的最短路过程中，需要使用一个 `vis` 数组记录每一个结点是否已经被遍历过了，由于数据过大，开不下这么大的数组，但是数据的数量不大，所以我们可以手写一个 `hash` 或者使用 STL 里自带的 `hash` 表 `unordered_map` 快速地 ( $O(1)$ ) 访问每一个元素。

时间复杂度  $O(n \log n)$

## Code

```
1  const int N = 50007, M = 500007, INF = 0x3f3f3f3f;
2  typedef long long ll;
3  typedef pair<int, int> PII;
4  unordered_map<int, bool> vis;
5  unordered_map<int, ll> dist;
6
7  int n, m;
8  int head[N], ver[N], nex[N], edge[M], tot;
9  int a[N], l[N], c[N];
10
11 int gcd(int a, int b)
12 {
13     if(b == 0) return a;
14     return gcd(b, a % b);
15 }
16 //gcd(0,x) = x;
17 void dijkstra()
18 {
19     priority_queue<PII, vector<PII>, greater<PII> > q;
20     q.push({0, 0});
21     dist[0] = 0;
22
23     while(q.size()) {
24         int x = q.top().second;
25         q.pop();
26         if(x == 1) break;
27
28         if(vis.find(x) != vis.end()) continue;
29         vis[x] = true;
30         for(int i = 1; i <= n; ++ i) {
31             int y = __gcd(x, l[i]), z = c[i];
32             if(dist.find(y) == dist.end()) dist[y] = INF;
33             if(dist[y] > dist[x] + z) {
34                 dist[y] = dist[x] + z;
35                 q.push({dist[y], y});
36             }
37         }
38     }
39 }
40
41 int main()
42 {
43     scanf("%d", &n);
44     for(int i = 1; i <= n; ++ i)
45         scanf("%d", &l[i]);
46
47     for(int i = 1; i <= n; ++ i)
48         scanf("%d", &c[i]);
49     dijkstra();
50
51     if(dist.find(1) == dist.end()) puts("-1");
52     else printf("%lld\n", dist[1]);
53     return 0;
54 }
```

0x22.2 扩展欧几里德算法

我们可以利用欧几里得算法求解  $ax + by = \gcd(a, b)$  中的  $x$  和  $y$ 。

我们知道欧几里得算法利用的核心性质为  $\gcd(a, b) = \gcd(b, a \bmod b) = \gcd(b, a - b\lfloor \frac{a}{b} \rfloor)$

根据裴蜀定理，我们一定可以找到四个整数  $x, y, x', y'$ ，使得  $ax + by = \gcd(a, b)$  且  $bx' + (a - b\lfloor \frac{a}{b} \rfloor)y' = \gcd(b, a - b\lfloor \frac{a}{b} \rfloor)$

由于  $\gcd(a, b) = \gcd(b, a \bmod b) = \gcd(b, a - b\lfloor \frac{a}{b} \rfloor)$

有：

$$ax + by = bx' + (a - b\lfloor \frac{a}{b} \rfloor)y'$$
$$a(x - y') + b(y - (x' - \lfloor \frac{a}{b} \rfloor y')) = 0$$

我们希望这个等式对一切  $a, b$  都成立，于是  $x = y'$  且  $y = x' - \lfloor \frac{a}{b} \rfloor y'$ 。

显然，我们想求  $x$  和  $y$ ，只要求出  $x', y'$ ，由于  $x', y'$  对应的问题相同且规模更小，所以可以进行递归的运算。边界条件为：当  $b = 0$  时， $x = 1, y = 0, a \times 1 + 0 \times 0 = \gcd(a, 0) = a$ 。

由于算法思想与欧几里得相同，我们称之为拓展欧几里得算法。

实现：

在用欧几里德算法求  $d = \gcd(a, b)$  的过程中求方程  $ax + by = d$  的一组整数解  $(x, y)$

若  $d \mid c$ ，不妨设  $c = kd$ ，则有  $a(kx) + b(ky) = c$ ，否则原方程无整数解。

```
1 //在gcd的过程上增加了拓展
2 inline int exgcd(int a, int b, int &x, int &y)
3 {
4     if(b == 0){
5         x = 1, y = 0, return a;
6     }
7     int d = exgcd(b, a % b, x, y);
8     int z = x;x = y, y = z - y * (a / b);
9     return d;
10 }
```

`exgcd` 可得到  $ax + by = \gcd(a, b)$  的解，对于  $ax + by = c$  的解，我们只需要根据 **定理22.3.3** 构造即可。

0x22.3 解二元模线性方程

二元模线性方程（二元一次不定方程）：形如  $ax \equiv c \pmod{b}$  或  $ax + by = c$ 。其中  $a, b, c, x, y$  均为整数。

**定理22.3.1**：上述方程有解的充要条件是  $\gcd(a, b) \mid c$

可以理解为  $\gcd(a, b)$  是  $ax + by$  可以表示出来的最小的正整数。

**定理22.3.2**：方程  $ax + by = d, d = \gcd(a, b)$  的所有解为：

$$\begin{cases} x = x_0 + k\frac{b}{d} \\ y = y_0 - k\frac{a}{d} \end{cases}$$

其中  $x_0, y_0$  是一组特解， $k \in \mathbb{Z}$ 。

**证明：**

方程的特解  $(x_0, y_0)$ ，任取另一组解  $(x, y)$ ，则  $ax_0 + by_0 = ax + by = \gcd(a, b)$ 。变形得  $a(x_0 - x) = b(y - y_0)$ 。设  $\gcd(a, b) = d$ ，方程左右两边同时除以  $d$ （如果  $d = 0$ ，说明  $a$  或  $b$  等于 0），得  $a'(x - x_0) = b'(y_0 - y)$ ，其中  $a' = \frac{a}{d}, b' = \frac{b}{d}$ 。显然此时  $a'$  和  $b'$  互质，因此  $x - x_0$  一定是  $b'$  的整数倍（因为  $a'$  中不包含  $b'$ ，所以  $x_0 - x$  一定包含  $b'$ ）。设它为  $kb'$ ，即  $x - x_0 = k\frac{b}{d}$ ，同理，有  $y_0 - y = k\frac{a}{d}$ 。

**定理22.3.3**：方程  $ax + by = c, \gcd(a, b) \mid c$  的所有解为

$$\begin{cases} x = \frac{c}{d} x_0 + k\frac{b}{d} \\ y = \frac{c}{d} y_0 - k\frac{a}{d} \end{cases}$$

其中  $x_0, y_0$  是方程  $ax + by = d, d = \gcd(a, b)$  的一组特解， $k \in \mathbb{Z}$ 。

$\mathbb{Z}$  是整数集，也就意味着 $k$  可以为负数，即最小正整数解：

若  $x > 0 \rightarrow x \bmod \frac{b}{d}$

- 竞赛例题选讲

Problem A A + B (UVALive6428)

给出  $a, b, S$  三个数，两个格子，第一个格子上的权值是  $a$ ，第二个格子上的权值是  $b$ ，每次可以使得第一个格子上的权值变成原权值加上第二个格子上的权值，也可以，问a使得第二个格子上的权值变成原权值加上第一个格子上的权值。问两个格子上的权值能否等于  $S$ 。

Solution

手动模拟一下：

```
1      (a      ,      b)
2      (a+b , b)      (a , a+b)
3      (a+2b,b) (a+b,a+2b) (2a+b,a+b) (a,2a+b)
```

可以发现实际上就是问有没有两个**非负整数**解  $x, y$ ，满足  $ax + by = S$  且  $\gcd(x, y) = 1$ 。

直接用拓展欧几里德算法求解系，看解系中是否存在满足题意的  $x, y$ 。注意一些细节上的特判即可。

Code

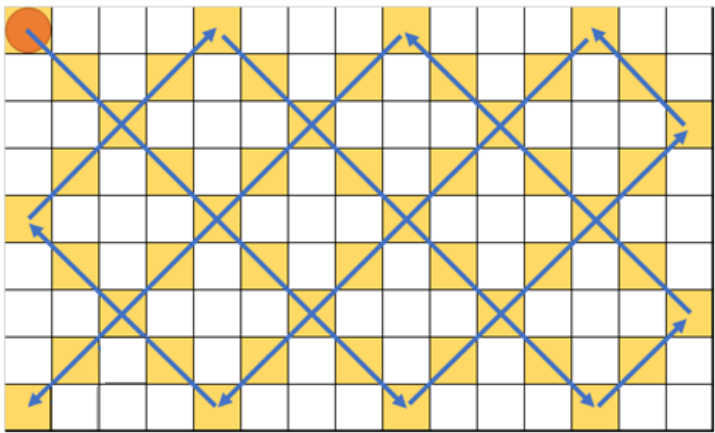
```
1 int n, m, t;
2 int a, b, s, kcase;
3 bool ans;
4
5 int exgcd(int a, int b, int &x, int &y)
6 {
7     if(b == 0) {
8         x = 1, y = 0;
9         return a;
10    }
11    int d = exgcd(b, a % b, x, y);
12    int z = x;
13    x = y;
14    y = z - y * (a / b);
15    return d;
16 }
17
18 void solve()
19 {
20     if(a == 0 && b == 0) {
21         ans = (s == 0);
22         return ;
23     }
24     if(a == 0) {
25         ans = (s % b == 0);
26         return ;
27     }
28     if(b == 0) {
29         ans = (s % a == 0);
30         return ;
31     }
32     int x, y;
33     int d = exgcd(a, b, x, y);
34     if(s % d != 0) {
35         ans = false;
36         return ;
37     }
38     int x0 = b / d;
39     int y0 = a / d;
40     x = ((s / d % x0) * (x % x0) % x0 + x0) % x0;
41     y = (s - x * a) / b;
42     ans = false;
43     while(y > 0) {
44         if(__gcd(x, y) == 1) {
45             ans = true;
46             return ;
47         }
48         else {
49             x += x0;
50             y -= y0;
```



```
51     }
52 }
53 }
54
55 signed main()
56 {
57     while (scanf("%lld%lld%lld", &a, &b, &s) != EOF) {
58         ans = false;
59         solve();
60         if (ans)
61             puts("YES");
62         else puts("NO");
63     }
64     return 0;
65 }
```

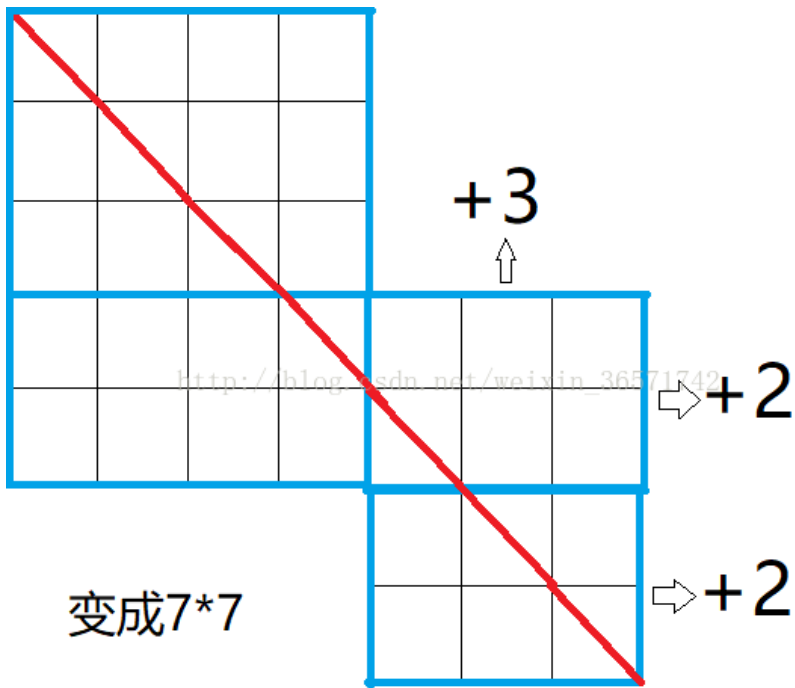
**Problem B Bounce (hihocoder1584 2017ICPC beijing网络赛)**

在  $n \times m$  的网格上，一颗小球左上方出发往右下的格子走，碰到边界就反弹，当小球到达角落就停止运动。求小球只经过一次的格子的数量。



**Solution**

对于这种反弹的问题，我们可以把每次反弹后的线路展开：



每次反弹就增加  $m - 1$ （碰到上下边界）或者  $n - 1$ （碰到左右边界）。因为小球是沿着  $45^\circ$  度方向移动，所以最后展开之后走的整个图形一定是一个正方形。于是可以列出不定方程  $(m - 1) \times x + m = (n - 1) \times y + n$ 。其中  $x$  和  $y$  可以通过扩展欧几里德求出最小正数解。根据图像可以看出  $(m - 1) \times x + m$  就是小球经过的格子总数（重复的也算）。

最后减去重复经过两次的格子即可。之前求得的方程的解中  $x$  就是左右反弹次数之和， $y$  就是上下反弹次数之和。那么相交的就是经过两次的次数，要减去。显然每次反弹的时候，向上下反弹一次，就一定会跟着向左右反弹一次。每两个相反的路线都会相交一次。或者分析其实际的意义，对于一个上下碰撞，会产生一个 'V' 字形，而左右碰撞会产生一个 ' > ' 形。每个 'V' 和 ' > ' 必会产生一个有且只有一个交点，故根据乘法原理，一共有  $x \times y$  个格子重复两次经过。即答案就是  $(m - 1) \times x + m - x \times y$ 。

**Code**

```
1 int n, m, t;
2 int a, b, s, kcase;
3 int ans;
4
5 int exgcd(int a, int b, int &x, int &y)
6 {
7     if (b == 0) {
8         x = 1, y = 0;
9         return a;
10    }
```

```

11     int d = exgcd(b, a % b, x, y);
12     int z = x;
13     x = y;
14     y = z - y * (a / b);
15     return d;
16 }
17
18 void solve()
19 {
20     int x, y;
21     int a = m - 1, b = 1 - n, c = n - m;
22     int d = exgcd(a, b, x, y);
23     int x0 = abs(b / d);
24     int y0 = abs(a / d);
25     x *= c / d;
26     if(x < 0) {
27         x = x + (-x / x0 + 1) * x0;
28     }
29     else x %= x0;
30     y = (c - a * x) / b;
31     printf("%lld\n", (m - 1) * x + m - x * y);
32 }
33
34 signed main()
35 {
36     while( ~ scanf("%lld%lld", &n, &m)) {
37         solve();
38     }
39     return 0;
40 }

```

### Problem C Integer Sequences (SGU 140)

给出一个长度为  $n$  的非负整数序列  $A$  和两个数  $P, B$ , 要求找出同样的非负整数序列  $X$  满足  $A_1 * X_1 + A_2 * X_2 + \dots + A_n * X_n = B \pmod{P}$

#### Solution

$$A_1 * X_1 + A_2 * X_2 + \dots + A_n * X_n = B \pmod{P}$$

显然有

$$A_1 * X_1 + A_2 * X_2 + \dots + A_n * X_n + PQ = B, Q \in \mathbb{Z}$$

看起来是一个多元一次方程, 我们只需要找到一个合法的非负整数序列  $X$  作为解即可, 因此我们可以构造答案。我们可以求出二元一次方程的解, 因此我们可以从前往后, 两个数就可以找到一组解, 这样两两合并即可得到一组合法的解。

即: 先考虑  $A_1 X_1 + A_2 X_2$ , 我们可以求出方程  $A_1 x + A_2 y = \gcd(A_1, A_2)$  的解  $x, y$ , 此时  $x$  就是满足当前条件的  $X$  序列的第一项的解,  $X_1 = x, X_2 = y$ 。我们把  $\gcd(A_1, A_2)$  当作新的元素, 于是得到新的方程:

$$\gcd(A_1, A_2)x + A_3 * X_3 + \dots + A_n * X_n + PQ = B, Q \in \mathbb{Z}$$

我们再合并  $\gcd(A_1, A_2)$  和  $A_3$ , 解出  $\gcd(A_1, A_2)x + A_3 y = \gcd(\gcd(A_1, A_2), A_3)$  的  $x, y$ , 把之前求出的所有的  $X_i$  乘上  $x$  (因为  $A_1 * X_1 + A_2 * X_2 = \gcd(A_1, A_2)$ ,  $\gcd$  的系数也要乘到前面的所有项上),  $X_3 = y$ , 不断重复, 直到合并只剩两项为止。

最后求解  $\gcd(A_1, A_2, A_3 \dots A_n)x + Py$  的  $x, y$ , 判断  $\gcd(A_1, A_2, A_3 \dots A_n)$  能否整除  $B$ , 若不能整除, 显然该丢番图方程无解, 输出  $NO$  即可。

若能整除, 所有的解乘上  $\frac{B}{\gcd(A_1, A_2 \dots A_n, P)}$  即为一组合法的解, 输出即可。

注意我们在求解  $x, y$  的过程中可能得到负数解, 而题目要求输出整数解, 最后输出的时候将  $X_i$  置于  $[0, P]$  之间即可。

#### Code

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 const int maxn = 100 + 7;
4 int n, m, s, t, k, ans;
5 int X[maxn];
6 int A[maxn], p, b;
7
8 int exgcd(int a, int b, int &x, int &y)
9 {
10     if(b == 0) {
11         x = 1, y = 0;
12         return a;
13     }
14     int d = exgcd(b, a % b, x, y);

```

```

15     int z = x;
16     x = y, y = z - y * (a / b);
17     return d;
18 }
19
20 int main()
21 {
22     scanf("%d%d%d", &n, &p, &b);
23     for (int i = 1; i <= n; ++ i)
24         scanf("%d", &A[i]), A[i] %= p;
25     int gcd = A[1];
26     X[1] = 1;
27     for (int i = 2; i <= n; ++ i) {
28         int x, y;
29         gcd = exgcd(gcd, A[i], x, y);
30         for (int j = 1; j < i; ++ j)
31             X[j] = X[j] * x % p;
32         X[i] = y;
33     }
34     int x, y;
35     gcd = exgcd(gcd, p, x, y);
36     for (int i = 1; i <= n; ++ i)
37         X[i] = X[i] * x % p;
38     if(b % gcd != 0) {
39         puts("NO");
40         return 0;
41     }
42     else {
43         puts("YES");
44         for (int i = 1; i <= n; ++ i) {
45             X[i] = X[i] * b / gcd % p;
46             printf("%d ", (X[i] + p) % p);
47         }
48     }
49     puts("");
50     return 0;
51 }

```

## OX22.4 类欧几里德算法（一个求和技巧）

- 竞赛例题选讲

### Problem A 类欧几里德算法 1

求  $\sum_{i=0}^n \lfloor \frac{ai+b}{c} \rfloor$

其中  $\lfloor \rfloor$  表示取整。

#### Solution

```

1 ll sum_pow(ll n, ll k) {
2     if (k == 0) return n;
3     else if (k == 1) return n * (n + 1) / 2;
4     else if (k == 2) return n * (n + 1) * (2 * n + 1) / 6;
5     else if (k == 3) return n * n * (n + 1) * (n + 1) / 4;
6     else if (k == 4) return n * (2 * n + 1) * (n + 1) * (3 * n * n + 3 * n - 1) / 30;
7     else assert(false);
8 }
9 ll EuclidLike1(ll a, ll b, ll c, ll n) {
10     if (a == 0) return b / c * (n + 1);
11     else if (a >= c || b >= c)
12         return (a / c) * sum_pow(n, 1) + (b / c) * (n + 1) + EuclidLike1(a % c, b % c, c, n);
13     else
14         return (a * n + b) / c * n - EuclidLike1(c, c - b - 1, a, (a * n + b) / c - 1);
15 }

```

### Problem B 类欧几里德算法 2

给定  $n, a, b, c$ , 求

$$\sum_{i=0}^n \lfloor \frac{ai+b}{c} \rfloor, \sum_{i=0}^n \lfloor \frac{ai+b}{c} \rfloor^2, \sum_{i=0}^n i \lfloor \frac{ai+b}{c} \rfloor$$

#### Solution

```

1 #define Int register int

```

```

2 #define mod 99824435311
3 #define int long long
4 int inv2 = 49912217711, inv6 = 16637405911;
5 struct Ans{int f,g,h;};
6 Ans Solve (int a,int b,int c,int n)
7 {
8     if (!a) {
9         int f = (n + 1) * (b / c) % mod;
10        int g = (n + 1) * (b / c) % mod * (b / c) % mod;
11        int h = n * (n + 1) % mod * inv2 % mod * (b / c) % mod;
12        return Ans {f % mod,g % mod,h % mod};
13    }
14    else if (a >= c || b >= c) {
15        Ans fucker = Solve (a % c,b % c,c,n);
16        int F = fucker.f + n * (n + 1) / 2 % mod * (a / c) % mod + (n + 1) * (b / c) % mod;
17        int G = fucker.g + 2 * (a / c) % mod * fucker.h % mod + 2 * (b / c) % mod * fucker.f +
18        n % mod * (n + 1) % mod * (2 * n % mod + 1) % mod * inv6 % mod * (a / c) % mod * (a / c) % mod +
19        n % mod * (n + 1) % mod * (a / c) % mod * (b / c) % mod + (n + 1) * (b / c) % mod * (b / c) % mod;
20        int H = fucker.h + n % mod * (n + 1) % mod * (2 * n + 1) % mod * inv6 % mod * (a / c) % mod + n % mod * (n + 1) % mod * inv2 %
mod * (b / c) % mod;
21        return Ans {F % mod,G % mod,H % mod};
22    }
23    else {
24        int M = (a * n + b) / c;
25        Ans fucker = Solve (c,c - b - 1,a,M - 1);
26        int F = n * M % mod - fucker.f;
27        int G = n * M % mod * (M + 1) % mod - 2 * fucker.h % mod + mod - 2 * fucker.f % mod + mod - F % mod;
28        int H = (M * n % mod * (n + 1) % mod - fucker.g + mod - fucker.f) % mod * inv2 % mod;
29        return Ans {F % mod,G % mod,H % mod};
30    }
31 }
32 int read ()
33 {
34     int x = 0;char c = getchar();int f = 1;
35     while (c < '0' || c > '9'){if (c == '-') f = -f;c = getchar();}
36     while (c >= '0' && c <= '9'){x = (x << 3) + (x << 1) + c - '0';c = getchar();}
37     return x * f;
38 }
39 void write (int x)
40 {
41     if (x < 0){x = -x;putchar ('-');}
42     if (x > 9) write (x / 10);
43     putchar (x % 10 + '0');
44 }
45 signed main()
46 {
47     int times = read ();
48     while (times --)
49     {
50         int n = read (),a = read (),b = read (),c = read ();
51         Ans Putout = Solve (a,b,c,n);
52         write ((Putout.f + mod) % mod),putchar (' '),write ((Putout.g + mod) % mod),putchar (' '),write ((Putout.h + mod) % mod),putchar
('\n');
53     }
54     return 0;
55 }

```

## 0x22.5 整式方程

整式方程就是方程中所有的未知数均在分子上，分母只是常数且无未知数。

通常情况下，常年用字母  $x, y, z$  来表示未知数，方程中含有几个不同的未知数就叫做几元，未知数的最高次数是几就叫做几次。

例如： $ax + b = c$  就是一个一元一次整式方程

### • 一元一次整式方程

对于方程  $ax + b = c$ ，有： $x = \frac{c-b}{a}$

```
1 double calculate(double a, double b, double c){
2     return (c - b) / a;
3 }
```

• 一元二次整式方程

```
1 double x1, x2;
2 bool calculate(double a, double b, double c){
3     double delta = b * b - 4 * a * c;
4     if(delta < 0) {
5         return false;
6     }
7     else if(delta == 0) {
8         x1 = ( - 1 * b + sqrt(delta) / (2 * a) );
9         x2 = x1;
10    }
11    else {
12        x1 = ( - 1 * b + sqrt(delta) / (2 * a) );
13        x2 = ( - 1 * b - sqrt(delta) / (2 * a) );
14    }
15    return true;
16 }
```

## Ox23 乘法逆元

### Ox23.1 乘法逆元定义与性质

**定义23.1.1**: 若 $a \times x \equiv 1 \pmod{b}$ , 且 $a$ 与 $b$ 互质, 那么我们就定义:  $x$  为  $a$  的逆元, 记为 $a^{-1}$ , 所以我们可以称  $x$  为  $a$  在  $\text{mod } b$  意义下的倒数。

由于除法不能直接取模, 但是可以用乘法逆元

对于 $\frac{a}{b} \text{ mod } p$ , 我们就可以求出  $b$  在  $\text{mod } p$  下的逆元, 然后乘上  $a$ , 再  $\text{mod } p$ , 即可得到该分数的值。

当然满足条件的逆元不止一个, 通常情况下我们只用最小正整数的逆元, 并且逆元也不在任何条件下都有逆元

有了乘法逆元, 我们在求计数类问题中遇到 $\frac{a}{b}$ 的除法算式的时候, 可以先把 $a, b$ 各自对模数 $p$ 取模, 然后再计算 $a \times b^{-1} \pmod{p}$ 作为最终的结果。(前提是保证 $b, p$ 互质, 当 $p$ 是质数的时候等价于 $b$ 不是 $p$ 的倍数)

若 $b, q$ 不互质, 显然 $bx \equiv 1 \pmod{p}$ 无解, 即不存在 $b$ 模 $p$ 的乘法逆元。

### Ox23.2 费马小定理求乘法逆元

**保证 $b, m$ 互质且 $m$ 是质数。**

费马小定理 $a^{p-1} \equiv 1 \pmod{p}$ , 其中 $p$ 为素数。

根据费马小定理变形得 $a \times a^{p-2} \equiv 1 \pmod{p}$ 。

则 $a^{p-2} \pmod{p}$ 就是逆元, 直接快速幂求得:

```
1 int inv(int x,int p) {return qpow(x, mod - 2, p) % p;}
```

### Ox23.3 扩展欧几里得求乘法逆元

若题目仅保证 $b, m$ 互质, 根据逆元的定义 $ax \equiv 1 \pmod{m}$ 可得丢番图方程组:  $ax + my = 1$ 。

解方程得 $x \text{ mod } m$ 得到的就是 $a$ 的乘法逆元, 同时可得逆元存在的条件 $\text{gcd}(a, m) = 1$ 。

```
1 inline int invers(int a,int mod) {
2     register int x,y;
3     exgcd(a,mod,x,y);
4     return (x % mod + mod) % mod;
5 }
```

### Ox23.4 线性递推求乘法逆元

给定 $n, p$ 求 $1 \sim n$ 中所有的整数在模 $p$ 意义下的乘法逆元。 $1 \leq n \leq 3 \times 10^6, n < p < 20000528$ , 输入保证 $p$ 是质数。

数据较大, 我们需要一个复杂度为 $O(n)$ 的算法: 线性递推。

首先, 很显然的 $1^{-1} \equiv 1 \pmod{p}$

对于 $\forall p \in \mathbb{Z}$ , 有 $1 \times 1 \equiv 1 \pmod{p}$ 恒成立, 故在 $p$ 下1的逆元是1, 而这是推算出其他情况的基础。

其次对于递归情况  $i^{-1}$ , 我们令  $k = \left\lfloor \frac{p}{i} \right\rfloor$ ,  $j = p \bmod i$ , 有  $p = ki + j$ 。再放到  $\bmod p$  意义下就会得到:  $ki + j \equiv 0 \pmod p$

两边同时乘  $i^{-1} \times j^{-1}$ :

$$\begin{aligned}kj^{-1} + i^{-1} &\equiv 0 \pmod p \\ i^{-1} &\equiv -kj^{-1} \pmod p\end{aligned}$$

再将  $k = \left\lfloor \frac{p}{i} \right\rfloor$ ,  $j = p \bmod i$  带入有:

$$i^{-1} \equiv -\left\lfloor \frac{p}{i} \right\rfloor (p \bmod i)^{-1} \pmod p$$

我们注意到  $p \bmod i < i$ , 显然我们在循环计算的时候, 在计算  $i$  的时候, 我们已经得到了所有的  $j < i$  的模  $p$  下的逆元  $j^{-1}$ 。

即:

$$i^{-1} \equiv \begin{cases} 1, & \text{if } i = 1, \\ -\left\lfloor \frac{p}{i} \right\rfloor (p \bmod i)^{-1}, & \text{otherwise.} \end{cases} \pmod p$$

```
1 inv[1] = 1;
2 for (int i = 2; i <= n; ++i) {
3     inv[i] = (long long)(p - p / i) * inv[p % i] % p;
4 }
```

其中我们使用  $p - \left\lfloor \frac{p}{i} \right\rfloor$  来防止出现负数。

显然时间复杂度为  $O(n)$ 。

另外我们注意到我们没有对 `inv[0]` 进行定义却可能会使用它: 当  $i \mid p$  成立时, 我们在代码中会访问 `inv[p % i]`, 也就是 `inv[0]`, 这是因为当  $i \mid p$  时不存在  $i$  的逆元  $i^{-1}$ 。显然如果  $i$  与  $p$  不互素时不存在相应的逆元 (当一般而言我们会使用一个大素数, 比如  $10^9 + 7$  来确保它有着有效的逆元)。因此需要指出的是: 如果没有相应的逆元的时候, `inv[i]` 的值是未定义的。

值得一提的是, 我们使用上述递归式求解单个数的逆元, 理论时间复杂度的上界为  $O(n^{\frac{1}{3}})$ , 高于拓展欧几里得的  $O(\log n)$ 。 (详见 [知乎回答](#))

```
1 int n, m;
2 int inv[N], p;
3 int main()
4 {
5     scanf("%d%d", &n, &p);
6     inv[1] = 1;
7     puts("1");
8     for(int i = 2; i <= n; ++ i) {
9         inv[i] = (ll)(p - p / i) * inv[p % i] % p;
10        printf("%d\n", inv[i]);
11    }
12    return 0;
13 }
```

Ox23.5 求阶乘的逆元

定义  $\text{inv}[i]$  为  $i!$  的逆元

我们知道  $\text{inv}[i + 1] = \frac{1}{i + 1!}$

两边同时乘上  $i + 1$  得  $\text{inv}[i + 1] \times (i + 1) = \frac{1}{i!} = \text{inv}[i]$ 。

我们只需要先求出  $\text{inv}[n]$ 然后往回递推即可。

当然我们也可以先  $O(n)$  求出  $1 \sim n$  的所有数的逆元, 然后求阶乘即可。

Ox24 线性同余方程

Ox24.1 同余方程

形如  $ax \equiv b \pmod m$  的方程称为同余方程。

根据同余的定义, 同余方程等价于  $ax + mt = b(t \in \mathbb{Z})$ , 可以用扩展欧几里得求解

$ax \equiv b \pmod m \Rightarrow ax \bmod m = b \bmod m \Rightarrow ax = b + mt \Rightarrow ax + mt = b(t \in \mathbb{Z})$

显然同余方程有解的条件是  $\gcd(a, m) \mid b$ 。

这个方程就是二维空间中的直线方程, 但是由于我们的  $x$  和  $y$  的取值均为整数, 所以这个方程的解是一些排列成直线的点集。

- 竞赛例题选讲

Problem A 同余方程 (NOIP 2012)



输入  $a, b$ , 求关于  $x$  的同余方程  $ax \equiv 1 \pmod{b}$  的**最小正整数解**。 ( $2 \leq a, b \leq 2 \times 10^9$ )

### Solution

```
1 inline ll exgcd(ll a, ll b, ll& x, ll& y) {
2     if(b == 0){x = 1, y = 0; return a;}
3     ll d = exgcd(b, a % b, x, y);
4     ll z = x; x = y, y = z - y * (a / b);
5     return d;
6 }
7 ll x, y, a, b;
8 int main()
9 {
10     scanf("%lld%lld", &a, &b);
11     exgcd(a, b, x, y);
12     printf("%lld\n", (x % b + b) % b);
13     //通过取模压缩到1~b之间即是最小正整数解
14     return 0;
15 }
```

### Problem B 青蛙的约会 (AcWing 222)

两只青蛙在网上相识了，它们决定见上一面。我们把这两只青蛙分别叫做青蛙  $A$  和青蛙  $B$ ，它们在一条单位长度 1 米首尾相接的数轴。设青蛙  $A$  的出发点坐标是  $x$ ，青蛙  $B$  的出发点坐标是  $y$ 。青蛙  $A$  一次能跳  $m$  米，青蛙  $B$  一次能跳  $n$  米，两只青蛙跳一次所花费的时间相同。纬度线总长  $L$  米。现在要你求出它们跳了几次以后才会碰面。

### Solution

$A, B$  不在同一位置，设两只青蛙  $A, B$  一共跳了  $t$  次后碰面

那么  $A$  要追  $B$  ( $y - x$ ) 米 (可能为负)

每跳一次  $A$  追  $B$  ( $m - n$ ) 米，一圈  $L$  米

$$mt + x \equiv nt + y \pmod{L}$$

$$(m - n)t \equiv y - x \pmod{L}$$

设  $a = m - n$ ,  $c = y - x$

得:  $at + Ly = c$

我们直接使用扩展欧几里得求解即可。

### Code

```
1 ll n, m, x, y;
2 ll a, b, L;
3 ll exgcd(ll a, ll b, ll &x, ll &y){
4     if(b == 0){x = 1; y = 0; return a;}
5     ll d = exgcd(b, a % b, x, y);
6     ll z = x; x = y; y = z - (a / b) * x;
7     return d;
8 }
9 int main()
10 {
11     scanf("%lld%lld%lld%lld%lld", &a, &b, &m, &n, &L);
12     ll d = exgcd(m - n, L, x, y);
13     if((b - a) % d != 0){
14         puts("Impossible");
15     }
16     else {
17         x *= (b - a) / d;
18         ll t = abs(L / d);
19         printf("%lld\n", (x % t + t) % t);
20     }
21     return 0;
22 }
```

### Problem B. Rise of Shadows (2020 ICPC shenyang I)

给定一个时钟，时针  $H$  小时转一圈，分针  $M$  分钟转一圈，求有多少时刻两个针的夹角  $\leq \alpha$ ，其中  $\alpha = \frac{2\pi A}{HM}$

### Solution

假设当前时间为  $t$ ，我们需要求夹角小于  $\alpha = \frac{2\pi A}{HM}$ ，当前时间  $t \in [0, HM)$ ，时针的角度为  $\frac{2\pi t}{HM}$ ，分钟角度为  $\frac{2\pi t}{M}$ 。

则方程  $|\frac{2\pi t}{HM} \% 2\pi - \frac{2\pi t}{M} \% 2\pi| \leq \frac{2\pi A}{HM}$  的所有正整数解  $t$  的个数即问题的答案。

将方程化简得：

$$\begin{aligned} t(H-1) \bmod HM &\leq A \\ \text{or} \\ t(H-1) \bmod HM &\geq HM-A \end{aligned}$$

根据同余的同除性：

若  $ca \equiv cb \pmod m$  则  $a \equiv b \pmod{\frac{m}{\gcd(m,c)}}$

$t(H-1) \bmod HM$  的取值会重复  $\gcd(H-1, HM)$  次。

因此我们将不等式同除  $d = \gcd(H-1, HM)$ ，得：

$$\begin{aligned} \frac{t(H-1)}{d} \bmod \frac{HM}{d} &\leq \frac{A}{d} \\ \text{or} \\ \frac{t(H-1)}{d} \bmod \frac{HM}{d} &\geq \frac{HM-A}{d} \end{aligned}$$

显然第一个式子在  $[0, \frac{HM}{d}]$  中解  $t$  共有  $0 \sim \frac{A}{d}$ ，共  $\frac{A}{d} + 1$  种取值。

第二个式子共有  $\frac{HM}{d} - \frac{HM-A}{d} = \frac{A}{d}$  种取值，共  $2 \times \frac{A}{d} + 1$  种取值。

乘上除去的  $d$ ，原式在  $[0, HM]$  共有  $d \times (2 \times \frac{A}{d} + 1)$  种取值。

注意当  $A = \frac{HM}{2}$  时，显然答案为  $HM$ ，特判即可。

Code

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using ll = long long;
4 const int maxn = 1e5 + 7;
5 int n, m, s, t, k;
6 ll ans;
7 ll H, M, A;
8 int main()
9 {
10     scanf("%lld%lld%lld", &H, &M, &A);
11     if(A == H * M / 2 ) {
12         printf("%lld\n", H * M);
13         return 0;
14     }
15     ll d = __gcd(H - 1, H * M);
16     ans = d * (2ll * (A / d) + 1);
17     cout << ans << endl;
18     return 0;
19 }
```

0x24.2 中国剩余定理

- 同余方程组

$$\begin{cases} x \equiv a_1 \pmod{m_1} \\ x \equiv a_2 \pmod{m_2} \\ \dots \\ x \equiv a_n \pmod{m_n} \end{cases}$$

其中  $a_1, a_2, \dots, a_n$  是整数,  $m_1, m_2, \dots, m_n$  是正整数且两两互质。

- 中国剩余定理

我们设：

$$M = m_1 \times m_2 \times \dots \times m_n, \quad M_i = \frac{M}{m_i}$$

$M_i \times M_i^{-1} \equiv 1 \pmod{m_i}$ ，其中  $M_i^{-1}$  是  $M_i$  的逆元。

我们可以构造出一个解  $x = \sum_{i=1}^k a_i M_i M_i^{-1}$

由此，任意解  $x_0$  即为  $x + k \times M$

最小正整数解  $x_{\min} = x_0 \% M$

如何证明构造出来的解  $x$  对于所有的同余方程都成立呢？

我们首先对第一个式子,  $x \bmod m_1$ , 其中  $i \in 2 \sim k$  中的  $M_i$  里面都是  $m_i$  的倍数,  $M_i \bmod m_i$  都等于 0,  $M_1 \times M_1^{-1} \equiv 1 \pmod{m_i}$ , 也就是说只有第一项  $\bmod m_i$  不为 0, 等于  $a_i$ , 那么也就是说满足第一个同余方程, 同理也满足所有的同余方程。

## Code

```
1  const ll N = 5e5 + 7;
2  int n, a[N], m[N];
3  ll exgcd(ll a, ll b, ll &x, ll &y) {
4      if(b == 0){
5          x = 1; y = 0;
6          return a;
7      }
8      ll d = exgcd(b, a % b, x, y);
9      ll z = x; x = y; y = z - (a / b) * x;
10     return d;
11 }
12 int main()
13 {
14     ll M = 1;
15     scanf("%d", &n);
16     for(int i = 1; i <= n; ++ i) {
17         scanf("%d%d", &m[i], &a[i]);
18         M *= m[i];
19     }
20     ll res = 0;
21     for(int i = 1; i <= n; ++ i) {
22         ll Mi = M / m[i];
23         ll ti, y;
24         //exgcd求逆元: 解同余方程: ax + my = 1; (ax ≡ 1 mod m)
25         ll d = exgcd(Mi, m[i], ti, y);
26         ti = (ti % m[i] + m[i]) % m[i];
27         res += a[i] * ti * Mi;
28     }
29     printf("%lld\n", (res % M + M) % M); //可能为负数, 所以需要处理一下
30     return 0;
31 }
```

## 0x24.3 拓展中国剩余定理

仍然是上面的问题, 只是  $m_1 c \dots m_n$  不保证两两互质了。

考虑如果只有两个方程如何求解：

$$\begin{cases} x \equiv a_1 \pmod{m_1} & (1) \\ x \equiv a_2 \pmod{m_2} & (2) \end{cases}$$

对于第一个方程而言, 解的形式是  $x = a_1 + km_1$ 。带入第二个方程, 得到:  $a_1 + km_1 \equiv a_2 \pmod{m_2}$

这个方程中只有  $k$  是未知的。用扩展欧几里得解出来一个  $k_0$ , 则任意  $km_1 = k_0m_1 + zm_2$  对于等式 (2) 都是合法的, 为了满足等式 (1) 所以要求  $zm_2 = u \times \text{lcm}(m_1, m_2)$ 。现在,  $x$  的表现形式为  $a_1 + k_0m_1 + u \times \text{lcm}(m_1, m_2)$ , 我们合并出了一个新的方程:

$$x \equiv a_1 + k_0m_1 \pmod{\text{lcm}(m_1, m_2)}$$

$z, u \in \mathbb{Z}$ 。

然后再合并  $n$  次即可。

## Code

```
1  typedef long long ll; const int N = 100007, M = 1000007, INF = 0x3f3f3f3f;
2  const double eps = 1e-8;
3  const int mod = 10007;
4  ll n, m;
5  ll bi[N], ai[N];
6  ll exgcd(ll a, ll b, ll &x, ll &y)
7  {
8      if(b == 0){x = 1; y = 0; return a;}
9      ll d = exgcd(b, a % b, x, y);
10     ll z = x; x = y; y = z - a / b * y;
11     return d;
12 }
13 ll mul(ll a, ll b, ll c) //注意数据范围可能会爆long long需要用到龟速乘
14 {
15     if(b < 0) a = -a, b = -b;
16     ll res = 0;
17     while(b){
```

```
18     if(b & 1)res  = (res + a) % c;
19     a = (a + a) % c;
20     b >>= 1;
21 }
22 return res;
23 }
24 ll excrt()//拓展中国剩余定理
25 {
26     ll x, y, k;
27     ll M = bi[1], ans = ai[1];//第一个方程的特解
28     for(int i = 2; i <= n; ++ i) {
29         ll a = M, b = bi[i], c = (ai[i] - ans % b + b) % b;
30         ll d = exgcd(a, b, x, y);
31         ll bg = b / d;//lcm
32         if(c % d != 0)return -1; //判断是否无解，然而这题其实不用
33         x = mul(x, c / d, bg);
34         ans += x * M;//更新前k个方程组的答案
35         M *= bg;//M为前k个m的lcm
36         ans = (ans % M + M) % M;
37     }
38     ans = (ans % M + M) % M;
39     //if(ans == 0) ans = M;//视情况而定，等于0的时候是因为给定的模数均为1，此时答案应该取任意值均可，而不是只有解 0 ， 有时需要特判一下。
40     return ans;
41 }
42
43 int main()
44 {
45     scanf("%lld", &n);
46     //bi -> m[i], ai -> a[i]
47     for(int i = 1; i <= n; ++ i)
48         scanf("%lld%lld", &bi[i], &ai[i]);
49     printf("%lld\n", excrt());
50     return 0;
51 }
```

A、 (P3868 [TJOI2009]) 猜数字

Weblink

<https://www.luogu.com.cn/problem/P3868>

Problem

现有两组数字，每组  $k$  个。

第一组中的数字分别用  $a_1, a_2, \cdots, a_k$  表示，第二组中的数字分别用  $b_1, b_2, \cdots, b_k$  表示。

其中第二组中的数字是两两互素的。求最小的  $n \in \mathbb{N}$ ，满足对于  $\forall i \in [1, k]$ ，有  $b_i | (n - a_i)$ 。

Solution

$$b_i \mid (n - a_i)$$
$$(n - a_i) \% b_i = 0$$
$$n - a_i \equiv 0 \pmod{b_i}$$
$$n \equiv a_i \pmod{b_i}$$

显然有  $n$  个同余方程，直接 CRT 解方程组即可。

注意数据保证  $\prod_{i=1}^k b_i \leq 10^{18}$ ，即  $M \leq 10^{18}$ ，那么CRT的过程中随便一乘就会爆 `long long`，所以要用快速乘。用  $\log n$  的快速乘竟然 T 了... 所以需要加一些经典优化或者用  $O(1)$  的 `long double` 快速乘，可以处理小于  $1.7 \times 10^{308}$  的数据。

Code

```
1 #include <bits/stdc++.h>
2 #include <algorithm>
3 #include <cstring>
4 #include <cmath>
5 #include <map>
6 #include <queue>
7 using namespace std;
8 #define int long long
9 typedef long long ll;
```

```

10 typedef int itn;
11 typedef pair<int, int> PII;
12 typedef pair<double, int> PDI;
13 const int N = 50 + 7, mod = 1e18 + 7, INF = 0x3f3f3f3f;
14 const double PI = acos(-1.0), eps = 1e-8;
15
16 int n, t, kcase;
17 int M, m[N], a[N], k;
18 /*
19 int mul(int a, int b, int mod)
20 {
21     int res = 0;
22     while(b) {
23         if(b & 1) res = (res + a) % mod;
24         a = (a + a) % mod;
25         b >>= 1;
26     }
27     return res;
28 }*/
29
30 int mul(int x, int y, int p)
31 {
32     int z = (long double)x / p * y;
33     ll res = (unsigned long long)x * y - (unsigned long long) z * p;
34     return (res + p) % p;
35 }
36
37 ll exgcd(int a, int b, int &x, int &y)
38 {
39     if(b == 0) {
40         x = 1, y = 0;
41         return a;
42     }
43     ll d = exgcd(b, a % b, x, y);
44     ll z = x;
45     x = y;
46     y = z - y * (a / b);
47     return d;
48 }
49
50 void solve()
51 {
52     ll M = 1;
53     scanf("%lld", &k);
54     for(int i = 1; i <= k; ++ i) {
55         scanf("%lld", &a[i]);
56     }
57     for(int i = 1; i <= k; ++ i) {
58         scanf("%lld", &m[i]);
59         M *= m[i];
60     }
61     for(int i = 1; i <= k; ++ i) a[i] = (a[i] % m[i] + m[i]) % m[i];
62     ll res = 0;
63     for(int i = 1; i <= k; ++ i) {
64         ll Mi = M / m[i];
65         ll ti, y;
66         ll d = exgcd(Mi, m[i], ti, y);
67         ti = (ti % m[i] + m[i]) % m[i];
68         res += mul(mul(a[i], ti, M), Mi, M);
69     }
70     printf("%lld\n", (res % M + M) % M);
71 }
72
73 signed main()
74 {
75     solve();
76     return 0;
77 }

```

## B、古代猪文 (P2480 [SDOI2010])

## Problem

猪王国的文明源远流长，博大精深。

iPig 在大肥猪学校图书馆中查阅资料，得知远古时期猪文文字总个数为  $n$ 。当然，一种语言如果字数很多，字典也相应会很大。当时的猪王国国王考虑到如果修一本字典，规模有可能远远超过康熙字典，花费的猪力、物力将难以估量。故考虑再三没有进行这一项劳猪伤财之举。当然，猪王国的文字后来随着历史变迁逐渐进行了简化，去掉了一些不常用的字。

iPig 打算研究古时某个朝代的猪文文字。根据相关文献记载，那个朝代流传的猪文文字恰好为远古时期的  $1/k$ ，其中  $k$  是  $n$  的一个正约数（可以是 1 或  $n$ ）。不过具体是哪  $1/k$ ，以及  $k$  是多少，由于历史过于久远，已经无从考证了。

iPig 觉得只要符合文献，每一种  $k|n$  都是有可能的。他打算考虑到所有可能的  $k$ 。显然当  $k$  等于某个定值时，该朝的猪文文字个数为  $n/k$ 。然而从  $n$  个文字中保留下  $n/k$  个的情况也是相当多的。iPig 预计，如果所有可能的  $k$  的所有情况数加起来为  $p$  的话，那么他研究古代文字的代价将会是  $g^p$ 。

现在他想知道猪王国研究古代文字的代价是多少。由于 iPig 觉得这个数字可能是天文数字，所以你只需要告诉他答案除以 999911659 的余数就可以了。

[https://blog.csdn.net/weixin\\_45697774](https://blog.csdn.net/weixin_45697774)

## Solution

void 函数写成 ll 了，没有返回值 RE 了...

懒得写题解了...

题目大意：求  $G^{\sum d|n C_n^d} \bmod 999911659$

思路与其他题解相像，考虑到 999911659 是质数，那么就用欧拉定理的推论得：

$$G^{\sum d|n C_n^d \bmod 999911659} = G^{\sum d|n C_n^d \bmod 999911658} \bmod 999911659$$

那么关键计算  $\sum d|n C_n^d \bmod 999911658$ 。直接 Lucas 绝对挂，那么尝试把模数缩小再合并

将 999911658 因数分解，可得  $999911658 = 2 \times 3 \times 4679 \times 35617$ 。那么把模数缩小，枚举  $n$  的因数  $d$ ，然后运用 Lucas 定理把  $C_n^d$  算出来，分别计算出  $\sum d|n C_n^d$  对 2, 3, 4679, 35617 四个质数取模的结果，记为  $a_1, a_2, a_3, a_4$ 。

最后，用中国剩余定理求解一下方程组：

$$\begin{cases} x \equiv a_1 \pmod{2} \\ x \equiv a_2 \pmod{3} \\ x \equiv a_3 \pmod{4679} \\ x \equiv a_4 \pmod{35617} \end{cases}$$

然后就得到了最小的非负整数解  $x$ ，之后用快速幂求一下  $G^x$  就得到答案

[https://blog.csdn.net/weixin\\_45697774](https://blog.csdn.net/weixin_45697774)

[上述题解来源...](#)

## Code

简单实现一下就行了

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4 typedef int itn;
5 const ll N = 5e5 + 7, p = 999911659, phi = 999911658, INF = 0x3f3f3f3f;
6 const double PI = acos(-1.0), eps = 1e-8;
7
8 ll val;
9 ll a[10];
10 ll M = 1;
11 ll fact[N];
12 ll infact[N];
13 ll n, g, t, cnt;
14 ll mo[10] = {0, 2, 3, 4679, 35617};
15
16 ll qpow(ll a, ll b, ll mod)
17 {
18     ll res = 1;
19     while(b) {
20         if(b & 1) res = res * a % mod;
21         a = a * a % mod;
```



```

22     b >>= 1;
23 }
24 return res;
25 }
26
27 ll exgcd(ll a, ll b, ll &x, ll &y)
28 {
29     if(b == 0) {
30         x = 1, y = 0;
31         return a;
32     }
33     ll d = exgcd(b, a % b, x, y);
34     ll z = x; x = y, y = z - y * (a / b);
35     return d;
36 }
37
38 void init(ll p)
39 {
40     fact[0] = infact[0] = 1;
41     for(ll i = 1; i < p; ++ i) {
42         fact[i] = fact[i - 1] * i % p;
43         infact[i] = infact[i - 1] * qpow(i, p - 2, p) % p;
44     }
45 }
46
47 ll C(ll a, ll b, ll p)
48 {
49     if(a < b) return 0;
50     return fact[a] * infact[b] % p * infact[a - b] % p;
51 }
52
53
54 ll lucas(ll a, ll b, ll p)
55 {
56     if(b == 0) return 1;
57     if(a < p && b < p) return C(a, b, p);
58     return C(a % p, b % p, p) * lucas(a / p, b / p, p) % p;
59 }
60
61 ll CRT()
62 {
63     ll res = 0;
64     for(ll i = 1; i <= 4; ++ i) {
65         ll Mi = M / mo[i];
66         ll ti, y;
67         ll d = exgcd(Mi, mo[i], ti, y);
68         ti = (ti % mo[i] + mo[i]) % mo[i];
69         res = (res + a[i] * ti % M * Mi % M) % M;
70     }
71     return (res % M + M) % M;
72 }
73
74 void solve()
75 {
76     M = phi;
77     scanf("%lld%lld", &n, &g);
78     if(g % p == 0) {
79         puts("0");
80         return ;
81     }
82
83     for(ll k = 1; k <= 4; ++ k) {
84         init(mo[k]);
85         for(ll i = 1; i * i <= n; ++ i) {
86             if(n % i == 0) {
87                 a[k] = (a[k] + lucas(n, i, mo[k])) % mo[k];
88                 if(i * i != n)
89                     a[k] = (a[k] + lucas(n, n / i, mo[k])) % mo[k];
90             }
91         }
92     }

```

```
93     printf("%lld\n", qpow(g, CRT(), p));
94 }
95
96 int main()
97 {
98     solve();
99     return 0;
100 }
```

Problem C. 屠龙勇士 (P4774 [NOI2018])

小 D 最近在网上发现了一款小游戏。游戏的规则如下：

- 游戏的目标是按照编号  $1 \rightarrow n$  顺序杀掉  $n$  条巨龙，每条巨龙拥有一个初始的生命值  $a_i$ 。同时每条巨龙拥有恢复能力，当其使用恢复能力时，它的生命值就会每次增加  $p_i$ ，直至生命值非负。只有在攻击结束后且当生命值 恰好 为 0 时它才会死去。
- 游戏开始时玩家拥有  $m$  把攻击力已知的剑，每次面对巨龙时，玩家只能选择一把剑，当杀死巨龙后这把剑就会消失，但作为奖励，玩家会获得全新的一把剑。

小 D 觉得这款游戏十分无聊，但最快通关的玩家可以获得 ION2018 的参赛资格，于是小 D 决定写一个笨笨的机器人帮她通关这款游戏，她写的机器人遵循以下规则：

- 每次面对巨龙时，机器人会选择当前拥有的，攻击力不高于巨龙初始生命值中攻击力最大的一把剑作为武器。如果没有这样的剑，则选择 攻击力最低 的一把剑作为武器。
- 机器人面对每条巨龙，它都会使用上一步中选择的剑攻击巨龙固定的  $x$  次，使巨龙的生命值减少  $x \times ATK$ 。
- 之后，巨龙会不断使用恢复能力，每次恢复  $p_i$  生命值。若在使用恢复能力前或某一次恢复后其生命值为 0，则巨龙死亡，玩家通过本关。

那么显然机器人的攻击次数是决定能否最快通关这款游戏的关键。小 D 现在得知了每条巨龙的所有属性，她想考考你，你知道应该将机器人的攻击次数  $x$  设置为多少，才能用最少的攻击次数通关游戏吗？

当然如果无论设置成多少都无法通关游戏，输出  $-1$  即可。

[https://blog.csdn.net/weixin\\_45697774](https://blog.csdn.net/weixin_45697774)

Solution

可以直接用 `multiset` 代替平衡树，计算出每条龙所需要的剑，并且这把剑是一定能打败这条龙的，不然最后方程组无解，就会输出 -1，所以我们直接把打败龙的奖励放入 `set` 里，再选下一把剑。显然题目中的恢复机制我们可以得到一个同余方程：

$$C_i x \equiv A_i \pmod{P_i}$$

题目数据不保证 `m[i]` 一定互质  $\Rightarrow$  exCRT

一般的同余方程为：

$$x \equiv A_i \pmod{P_i}$$

可以直接用拓展中国剩余定理。

但是本题的式子长这个样子：

$$C_i x \equiv A_i \pmod{P_i}$$

考虑转成标准式子

$$C_i x \equiv A_i \pmod{P_i}$$

显然有  $C_i x + P_i y = A_i$

exgcd解得  $x'$   $y'$

$$\begin{aligned} x &= x' + k \frac{b}{d} \\ &= x' + k \frac{P_i}{\gcd(P_i, C_i)} \\ &\pmod{\frac{P_i}{\gcd(P_i, C_i)}} \\ \Rightarrow x &\equiv x' \pmod{\frac{P_i}{\gcd(P_i, C_i)}} \end{aligned}$$

拓展中国剩余定理求解即可。

注意特判  $A_i > P_i$  的情况即可，此时  $P_i = 1$ ，答案显然就是  $\max\{\frac{A[i]}{use[i]}\}$

Code

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4 #define int long long
```

```

5 typedef pair<int, int> PII;
6 const ll N = 5e6 + 7, INF = 0x3f3f3f3f;
7 const double PI = acos(-1.0), eps = 1e-8;
8
9 int n, m, k;
10 int A[N], P[N], award[N], atk[N], one;
11 multiset<int> st;
12 int C[N], use[N];
13 int ai[N], bi[N];
14
15 void init()
16 {
17     one = true;
18     st.clear();
19     memset(use, 0, sizeof use);
20 }
21
22 int mul(int x, int y, int p)
23 {
24     int z = (long double)x / p * y;
25     int res = (unsigned long long)x * y - (unsigned long long) z * p;
26     return (res + p) % p;
27 }
28
29 int exgcd(int a, int b, int &x, int &y)
30 {
31     if(b == 0) {
32         x = 1, y = 0;
33         return a;
34     }
35     int d = exgcd(b, a % b, x, y);
36     int z = x;
37     x = y, y = z - (a / b) * y;
38     return d;
39 }
40
41 int excrt()
42 {
43     int x, y, k;
44     int M = bi[1], ans = ai[1];
45     for(int i = 2; i <= n; ++i) {
46         int a = M, b = bi[i], c = (ai[i] - ans % b + b) % b;
47         int d = exgcd(a, b, x, y);
48         int bg = b / d;
49         if(c % d != 0) return -1;
50         x = mul(x, c / d, bg);
51         ans += x * M;
52         M *= bg;
53         ans = (ans % M + M) % M;
54     }
55     return (ans % M + M) % M;
56 }
57
58 void sp_work()
59 {
60     int ans = - INF;
61     for(int i = 1; i <= n; ++i) {
62         if(use[i]) {
63             ans = max(ans, (int)ceil((double)A[i] / use[i]));
64         }
65     }
66     printf("%lld\n", ans);
67 }
68
69 bool work()
70 {
71     int x, y;
72     for(int i = 1; i <= n; ++i) {
73         int a = use[i], b = P[i], c = A[i];
74         int d = exgcd(a, b, x, y), bg = b / d;
75         if(c % d != 0)

```

```

76         return false;
77         x = mul(x, c / d, bg);
78         x = (x % b + b) % b;
79         ai[i] = x;
80         bi[i] = bg;
81     }
82     return true;
83 }
84
85 void solve()
86 {
87     init();
88     scanf("%lld%lld", &n, &m);
89     for(int i = 1; i <= n; ++ i) scanf("%lld", &A[i]);
90     for(int i = 1; i <= n; ++ i) {
91         scanf("%lld", &P[i]);
92         if(A[i] > P[i])
93             one = false;
94     }
95     for(int i = 1; i <= n; ++ i) scanf("%lld", &award[i]);
96     for(int i = 1; i <= m; ++ i) {
97         scanf("%lld", &atk[i]);
98         st.insert(atk[i]);
99     }
100
101     for(int i = 1; i <= n; ++ i) {
102         multiset<int> :: iterator it;
103         if(A[i] < *st.begin()) it = st.begin();
104         else it = -- st.upper_bound(A[i]);
105         use[i] = *it;
106         st.insert(award[i]);
107         st.erase(it);
108     }
109     if(one == false) {
110         sp_work();
111         return ;
112     }
113     bool win = work();
114     if(win == 0) {
115         puts("-1");
116         return ;
117     }
118
119     printf("%lld\n", excrt());
120     return ;
121 }
122
123
124 signed main()
125 {
126     int t;
127     scanf("%lld", &t);
128     while(t -- ) {
129         solve();
130     }
131     return 0;
132 }
133
134 /*
135 487472861809
136 3871865111
137 7560798679
138 584853762636
139 670310334583
140 */

```

## 0x25 高次同余方程（一）

关于高次同余方程，一般有  $a^x \equiv b \pmod{p}$  以及  $x^a \equiv b \pmod{p}$  两种，这里只讨论第一种高次同余方程，第二种高次同余方程需要利用原根，阶，指标等概念，我们将在下面介绍这些概念，第二种高次同余方程的解法，详见本文0x64.5 高次同余方程（二）

0x25.1 BSGS算法

我们这里使用BSGS（Baby Step,Giant Step）算法求解第一类高次同余方程。

给定正整数  $a, b, p$ , 保证  $a, p$  互质, 求最小的非负整数  $t$ , 满足  $a^t \equiv b \pmod{p}$ 。

显然, 由欧拉定理,  $a^{\varphi(p)} \equiv 1 \pmod{p}$ , 故  $a^t \equiv a^{x \bmod \varphi(p)} \pmod{p}$ , 故  $t < \varphi(p)$ , 即  $t \in [0, \varphi(p))$ , 换句话说,  $a^t$  在模  $p$  意义下有一个长度为  $\varphi(p)$  的**循环节**, 显然我们只需要暴力枚举  $t \in [0, \varphi(p))$  判断是否满足该高次同余方程即可, 因为之后是无限的循环之中。时间复杂度为  $O(\varphi(p))$ , 但是当  $p$  是质数的时候,  $\varphi(p) = p - 1$ , 所以是一个  $O(n)$  的算法, 考虑优化。

我们设  $t = kx - y$ , 其中  $k$  为我们自己选定的一个定值, 则原同余方程变为  $a^{kx-y} \equiv b \pmod{p}$

$$\begin{aligned} a^{kx-y} &\equiv b \pmod{p} \\ a^{kx} \times a^{-y} &\equiv b \pmod{p} \\ a^{kx} &\equiv ba^y \pmod{p} \end{aligned}$$

此时  $x, y$  未知, 其中显然有  $x < \frac{\varphi(p)}{k}, y < k$ 。

我们显然可以从 1 到  $k$  枚举  $y$ , 预处理出所有的  $ba^y \bmod p$ , 相同的只保留最小的  $y$  (因为我们要求的是最小的正整数解, 如果求的是最大的正整数解的话就要保留最大的  $y$ ), 时间复杂度为  $O(k)$ 。

然后再从 1 到  $\frac{\varphi(p)}{k}$  枚举  $x$ , 在哈希表中查询判断是否有  $a^{kx} \equiv ba^y \pmod{p}$ , 找到则时间复杂度为  $O(\frac{\varphi(p)}{k})$ 。

这样处理给定  $a, p$ , 询问给出  $b$  的问题, 时间复杂度为  $O(\frac{\varphi(p)}{k} + k)$ , 显然要想  $\frac{\varphi(p)}{k} + qk$  最小, 取  $\frac{\varphi(p)}{k} = k, k = \sqrt{\varphi(p)}$  时, 时间复杂度最优。但是我们没必要专门再计算出  $\varphi(p)$  的值, 显然有  $\sqrt{\varphi(p)} < \lceil \sqrt{p} \rceil$ , 我们直接取  $k = \lceil \sqrt{p} \rceil$ , 则整体的时间复杂度为  $O(\sqrt{p})$ 。

Hint

- 这里取  $\lceil \sqrt{p} \rceil$  是因为我们的  $t \in [0, \varphi(p) - 1]$ , 但是如果对于  $p$  求  $\varphi(p)$ , 若  $p$  过大, 开销较大, 没有必要, 而我们知道  $\sqrt{\varphi(p)} < \lceil \sqrt{p} \rceil$ , 故我们可以就近取  $k = \lceil \sqrt{p} \rceil$  作为代替, 只要完全包含整个区间, 枚举的时候可以覆盖整个区间, 稍微多一点无所谓。
- 我们设  $t = kx - y$ , 选择  $-y$  是因为我们下面要将  $y$  放到右边, 减去, 所以这里设为  $-y$  会变成  $+y$ , 不用求逆元了, 因为  $y$  是常数, 所以取正负无影响。
- $0 \leq x \leq k, k = \lceil \sqrt{p} \rceil$ , 但是我们循环的时候要从 0 开始循环,  $x = 0$  的情况单独出来特判, 因为我们要求的是最小的正整数解  $t$ , 而如果  $x = 0$ ,  $t = kx - y$ ,  $t$  会为负数, 不符合题意, 故将  $x = 0$  的情况单独拿出来特判。

Code

```
1 typedef long long ll;
2 int BSGS(int a, int b, int p)
3 {
4     if (1 % p == b % p) return 0;
5     int k = sqrt(p) + 1;
6     unordered_map<int, int> hash;
7     for (int i = 0, j = b % p; i < k; i++)
8     {
9         hash[j] = i;
10        j = (1ll)j * a % p;
11    }
12    int ak = 1;
13    for (int i = 0; i < k; i++) ak = (1ll)ak * a % p;
14
15    for (int i = 1, j = ak; i <= k; i++)
16    {
17        if (hash.count(j)) return (1ll)i * k - hash[j];
18        j = (1ll)j * ak % p;
19    }
20    return -1;
21 }
22
23 int main()
24 {
25     int a, p, b;
26     //a^x ≡ b (mod p)
27     cin >> p >> a >> b;
28
29     int res = BSGS(a, b, p);
30     if (res == -1) puts("no solution");
31     else cout << res << endl;
```

```

32
33     return 0;
34 }

```

## ox25.2 拓展BSGS算法

给定  $a, p, b$ , 求满足  $a^x \equiv b \pmod{p}$  的最小自然数  $x$ ,  $a, p, b \leq 10^9$ 。不保证  $a, p$  互质。

如果  $a, p$  不互质, 就需要进行一些扩展

如果  $b = 0$ , 则  $p = 1$  时有解, 否则无解

如果  $b = 1$ , 则  $x = 0$

否则, 设  $d = \gcd(a, p)$ , 如果  $d \nmid b$  则无解, 否则两边同除以  $d$ , 得到:

$$\left(\frac{a}{d}\right)a^{x-1} \equiv \frac{b}{d} \pmod{\frac{p}{d}}$$

因为  $\frac{a}{d}, \frac{p}{d}$  互质, 所以

$$a^{x-1} \equiv \left(\frac{a}{d}\right)^{-1} \left(\frac{b}{d}\right) \pmod{\frac{p}{d}}$$

多次执行上面的过程, 直到  $a, p$  互质, 然后使用BSGS求解即可。

### Code

```

1  typedef long long ll;
2  map<ll, int> mp;
3  inline int read() {
4      int ans = 0;
5      char ch = getchar();
6      while (ch < '0' || ch > '9') {
7          ch = getchar();
8      }
9      while (ch >= '0' && ch <= '9') {
10         ans = ans * 10 + (ch ^ 48);
11         ch = getchar();
12     }
13     return ans;
14 }
15
16 inline ll qpow(ll x, ll p, ll mod) {
17     ll ans = 1;
18     while (p) {
19         if (p & 1)
20             ans = ans * x % mod;
21         x = x * x % mod;
22         p >>= 1;
23     }
24     return ans;
25 }
26
27 inline int bsgs(int a, ll b, int p) {
28     if (p == 1)
29         return 0;
30     a %= p;
31     b %= p;
32     if (b == 1)
33         return 0;
34     int m = ceil(sqrt(p - 1)), i = 0;
35     ll t = qpow(a, m, p);
36     mp.clear();
37     for (register ll j = b; i < m; i++, j = j * a % p) {
38         mp[j] = i;
39     }
40     i = 1;
41     for (register ll j = t; i <= m; i++, j = j * t % p) {
42         if (mp.count(j))
43             return i * m - mp[j];
44     }
45     return -1;
46 }
47
48 int gcd(int a, int b) {
49     return b == 0 ? a : gcd(b, a % b);
50 }

```



```

51
52 void exgcd(ll a, ll b, ll &x, ll &y) {
53     if (b == 0) {
54         x = 1;
55         y = 0;
56         return;
57     }
58     ll t;
59     exgcd(b, a % b, x, y);
60     t = x;
61     x = y;
62     y = t - a / b * y;
63 }
64
65 inline ll inv(ll a, ll b) {
66     ll x, y;
67     exgcd(a, b, x, y);
68     return (x % b + b) % b;
69 }
70
71 inline int exbsgs(int a, int b, int p) {
72     if (p == 1)
73         return 0;
74     a %= p;
75     b %= p;
76     if (b == 1)
77         return 0;
78     int x = 0, t, ans;
79     ll y = 1;
80     while ((t = gcd(a, p)) != 1) {
81         if (b % t != 0)
82             return -1;
83         b /= t;
84         p /= t;
85         x++;
86         y = y * (a / t) % p;
87         if (b == y)
88             return x;
89     }
90     ans = bsgs(a, b * inv(y, p) % p, p);
91     if (ans == -1)
92         return -1;
93     return ans + x;
94 }
95
96 void write(int n) {
97     if (n >= 10)
98         write(n / 10);
99     putchar(n % 10 + '0');
100 }
101
102 int main() {
103     while (true) {
104         int a = read(), p = read(), b = read(), ans;
105         if (a == 0 && p == 0 && b == 0)
106             break;
107         ans = exbsgs(a, b, p);
108         if (ans == -1) {
109             cout << "No Solution" << endl;
110         } else {
111             write(ans);
112             putchar('\n');
113         }
114     }
115     return 0;
116 }

```

## ● 竞赛例题选讲

**Problem A 随机数生成器** [luogu P3306 [SDOI2013]](<https://www.luogu.com.cn/problem/P3306>)

最近小 W 准备读一本新书，这本书一共有  $p$  页，页码范围为  $0 \sim p - 1$ 。

小 W 很忙，所以每天只能读一页书。为了使事情有趣一些，他打算使用 NOI2012 上学习的线性同余法生成一个序列，来决定每天具体读哪一页。

我们用  $x_i$  来表示通过这种方法生成出来的第  $i$  个数，也即小 W 第  $i$  天会读哪一页。这个方法需要设置 3 个参数  $a, b, x_1$ ，满足  $0 \leq a, b, x_1 < p$ ，且  $a, b, x_1$

都是整数。按照下面的公式生成出来一系列的整数：

$$x_{i+1} \equiv a \times x_i + b \pmod{p}$$

其中  $mod$  表示取余操作。

但是这种方法可能导致某两天读的页码一样。

小 W 要读这本书的第  $t$  页，所以他想知道最早在哪一天能读到第  $t$  页，或者指出他永远不会读到第  $t$  页。

Solution

根据题意我们可以得到一个递推公式：

$$x_n = ax_{n+1} + b$$

(经典高中数列套路) 设：

$$\begin{aligned} x_n + c &= a(x_{n-1} + c) \\ x_n + c &= ax_{n-1} + ac \\ x_n &= ax_{n-1} + ac - c \\ x_n &= ax_{n-1} + (a - 1)c \\ &\rightarrow (a - 1)c = b \rightarrow c = \frac{b}{a - 1} \\ x_n + \frac{b}{a - 1} &= a(x_{n-1} + \frac{b}{a - 1}) \end{aligned}$$

即可得到：

$$\begin{aligned} x_n + \frac{b}{a - 1} &= a(x_{n-1} + \frac{b}{a - 1}) \\ &= a(a(x_{n-2} + \frac{b}{a - 1})) \\ &= \dots \\ &= a^{n-1}(x_1 + \frac{b}{a - 1}) \end{aligned}$$

显然根据题意， $x_n = t$

我们就可以将答案的表达式转化为标准的高次同余方程的形式：

$$a^{n-1} \equiv \frac{t + \frac{b}{a-1}}{x_1 + \frac{b}{a-1}} \pmod{p}$$

使用 BSGS 算法求解即可。

注意观察等式，需要特判几个特殊情况，因为有分数，分母不能为 0：

1.  $a = 1 \rightarrow a - 1 = 0$
2.  $a = 0$
3.  $x_1 + \frac{b}{a - 1} = 0$
4.  $a = 1, b = 0$ , 当  $n = 1$ ,  $x_n = x_1$ , 当  $n \geq 2$ ,  $x_n = b$
5.  $a = 1, b \neq 0$ ,  
 $x_n \equiv x_{n-1} + b \pmod{p} \rightarrow x_n = x_{n-1} + b \rightarrow x_n = (n - 1)b + x_1 \rightarrow x_n \equiv (n - 1)b + x_1 \pmod{p} \rightarrow b(n - 1) + py = t - x_1 \rightarrow exgcd, x = n - 1$

Code

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 #define int long long
4 typedef long long ll;
5 typedef pair<int, int> PII;
6 const ll N = 5e6 + 7, INF = 0x3f3f3f3f;
7 const double PI = acos(-1.0), eps = 1e-8;
8
9 int n, m, p, a, b, x1, t, T;
10
11 int qpow(int a, int b, int mod)
12 {
13     int res = 1;
```

```

14     while(b) {
15         if(b & 1) res = res * a % mod;
16         a = a * a % mod;
17         b >>= 1;
18     }
19     return res;
20 }
21
22 int inv(int a, int p)
23 {
24     return qpow(a, p - 2, p);
25 }
26
27 int exgcd(int a, int b, int &x, int &y)
28 {
29     if(b == 0) {
30         x = 1; y = 0;
31         return a;
32     }
33     int d = exgcd(b, a % b, x, y);
34     int z = x;
35     x = y, y = z - (a / b) * y;
36     return d;
37 }
38
39
40 int BSGS(int a, int b, int p)
41 {
42     if (1 % p == b % p) return 0;
43     int k = sqrt(p) + 1;
44     unordered_map<int, int> hash;
45     for (int i = 0, j = b % p; i < k; i ++ )
46     {
47         hash[j] = i;
48         j = j * a % p;
49     }
50     int ak = 1;
51     for (int i = 0; i < k; i ++ ) ak = ak * a % p;
52
53     for (int i = 1, j = ak; i <= k; i ++ )
54     {
55         if (hash.count(j)) return i * k % p - hash[j] % p;
56         j = j * ak % p;
57     }
58     return -1;
59 }
60
61 void solve()
62 {
63     scanf("%lld%lld%lld%lld%lld", &p, &a, &b, &x1, &t);
64     if(a == 0) {
65         if(x1 == t) puts("1");
66         else if(b == t) puts("2");
67         else puts("-1");
68         return ;
69     }
70     else if(a == 1) {
71         if(b == 0) {
72             if(x1 == t) puts("1");
73             else puts("-1");
74             return ;
75         }
76         int x, y;
77         int d = exgcd(b, p, x, y);
78         x = (x * (t - x1) % p / d + p) % p;
79         printf("%lld\n", x + 1);
80         return ;
81     }
82     else {
83         int C = b * inv(a - 1, p) % p;
84         int A = (x1 + C) % p;

```

```
85     if(A == 0) {
86         int res = (-C + p) % p;
87         if(res == t) puts("1");
88         else puts("-1");
89         return ;
90     }
91     else {
92         int B = (t + C) % p * inv(A, p) % p;
93         int n = BSGS(a, B, p);
94         if(n == -1) puts("-1");
95         else printf("%lld\n", n + 1);
96         return ;
97     }
98 }
99 }
100 signed main()
101 {
102     scanf("%lld", &T);
103     while(T -- ) {
104         solve();
105     }
106     return 0;
107 }
```

## Ox26 【题型探究】公约数之和

### ox26.1 母题：UVA11417 GCD

#### Problem

##### 题目描述

给定  $n$ ，求

$$\sum_{i=1}^n \sum_{j=i+1}^n \gcd(i, j)$$

其中  $\gcd(i, j)$  指的是  $i$  和  $j$  的最大公约数。

##### 输入格式

本题有多组数据。

对于每组数据，输出一个整数  $n$ ，如果  $n = 0$  就终止程序。

##### 输出格式

对于每组数据，输出计算结果。

##### 说明 / 范围

对于 100% 的数据， $1 \leq n \leq 501$ 。

[https://blog.csdn.net/weixin\\_45697774](https://blog.csdn.net/weixin_45697774)

#### Solution

数据很小，我们只需要直接暴力枚举即可。

#### Code

```
1 #include <cstdio>
2 #include <algorithm>
3 #include <cstring>
4 #include <vector>
5 #include <iostream>
6
7 using namespace std;
8 typedef long long ll;
9 const int N = 500007;
10
11 int gcd(int a, int b)
12 {
```

```
13     if(b == 0) return a;
14     return gcd(b, a % b);
15 }
16
17 ll solve(int n)
18 {
19     ll res = 0;
20     for(int i = 1; i <= n; ++ i){
21         for(int j = i + 1; j <= n; ++ j){
22             res += gcd(i, j);
23         }
24     }
25     return res;
26 }
27 int n;
28 int main()
29 {
30     while(scanf("%d", &n) != EOF && n){
31         printf("%d\n", solve(n));
32     }
33     return 0;
34 }
```

ox26.2 拓展一、UVA11426 GCD - Extreme (II)

Problem

题目描述

得定  $n$  , 求

$$\sum_{i=1}^n \sum_{j=i+1}^n \gcd(i, j)$$

其中  $\gcd(i, j)$  指的是  $i$  和  $j$  的最大公约数。

[https://blog.csdn.net/weixin\\_45697774](https://blog.csdn.net/weixin_45697774)

Solution

本题的数据开到了  $4 * 10^6$  , 所以暴力一定TLE。

啊, 这, 范围太大了, 我可以莫比乌斯反演 + 整除分块,  $O(\sqrt{n})$  解决!

《算法竞赛入门经典训练指南》上提供了一个构造函数的方法:

我们设  $f(n) = \gcd(1, n) + \gcd(2, n) + \gcd(3, n) + \dots + \gcd(n - 1, n)$

答案很明显就是  $S(n) = f(2) + f(3) + \dots + f(n)$

我们首先考虑如何求  $f(n)$

我们可以先把  $\gcd(x, n)$  的值分类, 我们发现  $\gcd(x, n)$  肯定是  $n$  的约数, 再设  $g(n, x)$  表示  $\gcd(x, n) = i$  的小于  $n$  的正整数的个数。

我们发现 $[\gcd(x, n) = i] \rightarrow [\gcd(\frac{x}{i}, \frac{n}{i}) == 1]$ , 和  $\frac{n}{i}$  互质的数的个数即为  $\varphi(\frac{n}{i})$ 个。

显然  $f(n) = \sum_{i|n} i * g(n, i)$

我们直接求一下 $S(n)$ , 既是答案。

Code

```
1  #include <cstdio>
2  #include <algorithm>
3  #include <cstring>
4  #include <vector>
5  #include <iostream>
6
7  using namespace std;
8  typedef long long ll;
9  const int N = 5000007;
10
11 ll s[N], primes[N];
12 ll f[N];
13 ll n, m, cnt;
14 ll phi[N];
15 bool vis[N];
```

```
16
17 void get_phi(ll n)
18 {
19     vis[1] = true;
20     for(int i = 2; i < n; ++ i){
21         if(vis[i] == 0){
22             primes[ ++ cnt] = i;
23             phi[i] = i - 1;
24         }
25         for(int j = 1; j <= cnt && primes[j] * i < n; ++j){
26             vis[i * primes[j]] = true;
27             if(i % primes[j] == 0){
28                 phi[i * primes[j]] = phi[i] * primes[j];
29                 break;
30             }
31             else phi[i * primes[j]] = phi[i] * (primes[j] - 1);
32         }
33     }
34 }
35
36 int main()
37 {
38     get_phi(N - 5);
39     for(int i = 1; i <= N - 1; ++ i){
40         for(int j = i * 2; j < N; j += i)
41             f[j] += i * phi[j / i];
42     }
43     for(int i = 1; i <= N - 1; ++ i)
44         s[i] = s[i - 1] + f[i];
45     while(scanf("%lld\n", &n) == 1 && n){
46         printf("%lld\n", s[n]);
47     }
48     return 0;
49 }
```

ox26.3 扩展二、luogu P2398 GCD SUM

Problem

题目描述

求

$$\sum_{i=1}^n \sum_{j=1}^n \gcd(i, j)$$

输入格式

第一行一个整数  $n$ 。

[https://blog.csdn.net/weixin\\_45697774](https://blog.csdn.net/weixin_45697774)

Solution

这道题要求的式子是： $\sum_{i=1}^n \sum_{j=1}^n \gcd(i, j)$

其实就是： $\left(\sum_{i=1}^n \sum_{j=1}^n \gcd(i, j)[i < j]\right) \times 2 + \sum_{i=1}^n i$  答案就应该是：

$$S(n) \times 2 + \sum_{i=1}^n i$$



$S(n)$ 是上一题的函数。

$$\sum_{i=1}^3 \sum_{j=1}^3 \gcd(i, j)$$

$ans = (1,1) + (1,2) + (1,3) + (2,1) + (2,2) + (2,3) + (3,1) + (3,2) + (3,3)$

(1,1) is crossed out. (1,2) and (2,1) are circled and connected by a line. (1,3) and (3,1) are circled and connected by a line. (2,2) is circled. (2,3) and (3,2) are circled and connected by a line. (3,3) is circled.

= 3 (next to (1,2))  
 =  $\sum_{i=1}^3 i$  (next to (3,3))

[https://blog.csdn.net/weixin\\_45697774](https://blog.csdn.net/weixin_45697774)

## Code

```

1 #include <cstdio>
2 #include <algorithm>
3 #include <cstring>
4 #include <vector>
5 #include <iostream>
6
7 using namespace std;
8 typedef long long ll;
9 const int N = 100007;
10
11 ll s[N], primes[N];
12 ll f[N];
13 ll n, m, cnt;
14 ll phi[N];
15 bool vis[N];
16
17 void get_phi(ll n)
18 {
19     vis[1] = true;
20     for(int i = 2; i < n; ++i){
21         if(vis[i] == 0){
22             primes[ ++ cnt] = i;
23             phi[i] = i - 1;
24         }
25         for(int j = 1; j <= cnt && primes[j] * i < n; ++j){
26             vis[i * primes[j]] = true;
27             if(i % primes[j] == 0){
28                 phi[i * primes[j]] = phi[i] * primes[j];
29                 break;
30             }
31             else phi[i * primes[j]] = phi[i] * (primes[j] - 1);
32         }
33     }
34 }
35
36 int main()
37 {
38     get_phi(N - 5);
39     for(int i = 1; i <= N - 1; ++i){
40         for(int j = i * 2; j < N; j += i)
41             f[j] += i * phi[j / i];
42     }
43     for(int i = 1; i <= N - 1; ++i)
44         s[i] = s[i - 1] + f[i];
45     cin >> n;
46     ll ans = 0;

```

```

47     for(int i = 1; i <= n; ++ i)
48         ans += i;
49     ans += 2 * s[n];
50     printf("%lld\n", ans);
51     return 0;
52 }

```

## 0x26.4 扩展三、luogu P2568 GCD

### Problem

#### 题目描述

给定正整数  $n$ ，求  $1 \leq x, y \leq n$  且  $\gcd(x, y)$  为素数的数对  $(x, y)$  有多少对。

#### 输入格式

只有一行一个整数，代表  $n$ 。

[https://blog.csdn.net/weixin\\_45697774](https://blog.csdn.net/weixin_45697774)

### Solution

这道题要求的式子是： $\sum_{i=1}^n \sum_{j=1}^n [\gcd(i, j) \text{ is prime}]$

其实就是： $(\sum_{i=1}^n \text{sumphi}(\frac{n}{i})[i \text{ is prime}]) \times 2 - \sum_{i=1}^n [i \text{ is prime}]$

我们只需要用线性筛筛一下素数即可。

$$1 \leq x, y \leq N$$

$$\gcd(x, y) = \text{质数的数对个数}$$

$$1 \sim N \text{ 中所有质数 } p$$

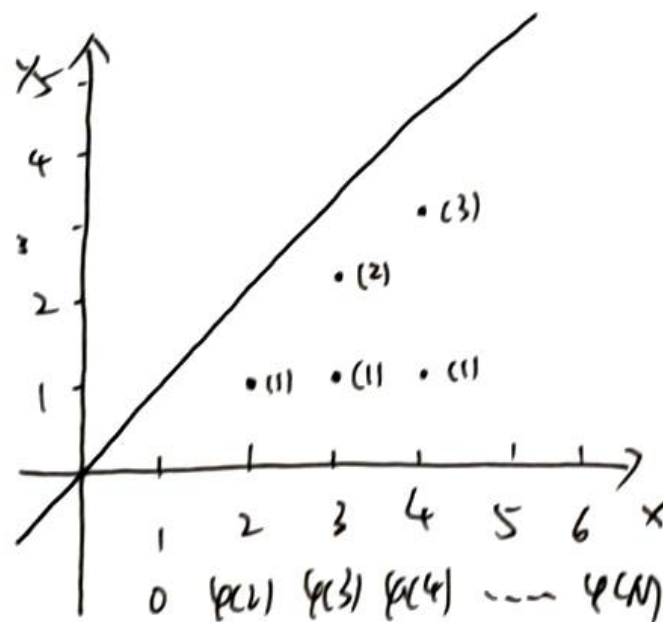
$$\gcd(x, y) = p$$

$$\gcd(\frac{x}{p}, \frac{y}{p}) = 1$$

$$\text{等价于: } 1 \leq x', y' \leq \frac{N}{p}$$

$$(x', y') = 1$$

$$\rightarrow \text{互质的个数}$$



$$\text{ans} = (\phi(1) + \phi(3) + \dots + \phi(N)) \times 2 + 1$$

统计欧拉数组的前缀和

这里定义  $\phi(1) = 0$  即可

$$\text{res} = \sum_{i=1}^N \text{每个质数 } p \text{ 的 } \text{sum}[N/p] \times 2 + 1$$

[https://blog.csdn.net/weixin\\_45697774](https://blog.csdn.net/weixin_45697774)

### Code

```

1 #include<cstdio>
2 #include<algorithm>
3 #include<cstring>
4 #include<iostream>

```

```

5
6 using namespace std;
7 typedef long long ll;
8 const int N = 10000007;
9
10 int n, m;
11 int primes[N];
12 int phi[N];
13 int cnt;
14 bool vis[N];
15 ll sum[N];
16
17 void init(int n)
18 {
19     phi[1] = 0; //这里应该是0
20     for(int i = 2; i <= n; ++ i){
21         if(vis[i] == 0){
22             primes[ ++ cnt] = i;
23             phi[i] = i - 1;
24         }
25         for(int j = 1; j <= cnt && primes[j] * i <= n; ++ j){
26             vis[i * primes[j]] = true;
27             if(i % primes[j] == 0){
28                 phi[i * primes[j]] = phi[i] * primes[j];
29                 break;
30             }
31             phi[i * primes[j]] = phi[i] * (primes[j] - 1);
32         }
33     }
34
35     for(int i = 1; i <= n; ++ i) //求phi[n]的前缀和
36         sum[i] = sum[i - 1] + phi[i];
37 }
38
39 int main()
40 {
41     scanf("%d", &n);
42     init(n);
43
44     ll ans = 0;
45     for(int i = 1; i <= cnt; ++ i){
46         int p = primes[i];
47         ans += sum[n / p] * 2 + 1;
48     }
49     printf("%lld\n", ans);
50     return 0;
51 }

```

## 0x30 积性函数

### 一些定义

- **数论函数**  
定义域为**正整数**的函数称为数论函数。
- **积性函数**  
对于数论函数  $f$ , 若任意**互质**的  $p, q$  都有  $f(pq) = f(p)f(q)$ , 则称  $f$  是积性函数。
- **完全积性函数**  
对于数论函数  $f$ , 若任意  $p, q$  都有  $f(pq) = f(p)f(q)$ , 则称  $f$  是完全积性函数
- **定义逐点加法**  
 $(f + g)(x) = f(x) + g(x)$ ,  $(f \cdot g)(x) = f(x)g(x)$

**定理30.1:** 积性函数一定满足  $f(1) = 1$ 。

考虑证明:

显然 1 与任何数都互质，满足积性函数的定义，那么我们假设存在一个正整数  $a$  满足  $f(a) \neq 0$ ，显然有： $f(a) = f(1 \times a) = f(1) \times f(a)$ ，两端同除  $f(a)$ ，得： $f(1) = 1$ ，性质得证  $\square$

**定理30.2：** 对于一个大于 1 的正整数  $N$ ，根据唯一分解定理有  $N = \prod p_i^{a_i}$ ，则对于任意积性函数  $f$ ，有： $f(N) = f(\prod p_i^{a_i}) = \prod f(p_i^{a_i})$  若  $f$  完全积性，则  $f(N) = \prod f(p_i)^{a_i}$ 。

由此可得推论：**凡是积性函数均可用线性筛法求解**

**性质30.3：** 对于一个大于 1 的整数由唯一分解定理有： $n = \prod p_i^{a_i}$ ，其中  $p_i$  为互不相同的素数。

对于一个**积性函数**  $f$ ， $f(n) = f(\prod p_i^{a_i}) = \prod f(p_i^{a_i})$ （不互质不能提出来）

对于一个**完全积性函数**  $f$ ， $f(n) = \prod f(p_i)^{a_i}$

**性质30.4：** 若  $f(x)$  和  $g(x)$  均为积性函数，则以下函数也为积性函数：

$$\begin{aligned} h(x) &= f(x^p) \\ h(x) &= f^p(x) \\ h(x) &= f(x)g(x) \\ h(x) &= \sum_{d|x} f(d)g(\frac{x}{d}) \end{aligned}$$

## 0x31 常见积性函数

- $\varphi(n)$  - 欧拉函数  $\varphi(n) = \sum_{i=1}^n [i \perp n]$
- $\mu(n)$  - 莫比乌斯函数，关于非平方数的质因子数目  $\mu(n) = \begin{cases} 1 & n = 1 \\ 0 & \exists d > 1, d^2 \mid n, \text{ 其中 } \omega(n) \text{ 表示 } n \text{ 的本质不同质因子个数, 它是一个加性函数。} \\ (-1)^{\omega(n)} & \text{otherwise} \end{cases}$

### 加性函数

此处的加性函数指数论上的加性函数 (Additive function)。对于加性函数  $f$ ，当整数  $a, b$  互质时，均有  $f(ab) = f(a) + f(b)$ 。应与代数中的加性函数 (Additive map) 区分。

- $\gcd(n, k)$  - 最大公因数，当  $k$  固定的情况
- 除数函数： $\sigma_k(n) = \sum_{d|n} d^k$ 
  - $\sigma_0(n)$  通常简记作  $d(n)$  或  $\tau(n)$ ，
  - $\sigma_1(n)$  通常简记作  $\sigma(n)$ 。
- $d(n)$  -  $n$  的正因子数目  $d(n) = \sum_{i|n} 1$
- $\sigma(n)$  -  $n$  的所有正因子之和  $\sigma(n) = \sum_{d|n} d$
- $1(n)$  - 不变函数，定义为  $1(n) = 1$ （完全积性）
- $id(n)$  - 单位函数（恒等函数），定义为  $id(n) = n$ （完全积性）
- $idk(n)$  - 幂函数，对于任何复数、实数  $k$ ，定义为  $idk(n) = n^k$ （完全积性）
- $\varepsilon(n)$  - 定义为： $\varepsilon(n) = [n = 1]$ ，若  $n = 1$ ， $\varepsilon(n) = 1$ ；若  $n > 1$ ， $\varepsilon(n) = 0$ 。即：**对于狄利克雷卷积的乘法单位 / 狄利克雷特卷积的单位元**，（完全积性）
- $\lambda(n)$  - 刘维尔函数，关于能整除  $n$  的质因子的数目
- $\gamma(n)$ ，定义为  $\gamma(n) = (-1)^{\omega(n)}$ ，在此加性函数  $\omega(n)$  是不同能整除  $n$  的质数的数目
- 另外，所有狄利克雷特征均是完全积性的。

积性函数 **欧拉函数** 已在 **0x14.1 欧拉函数** 中讲解过，这里不再赘述。

## 0x32 莫比乌斯函数

### 定义

$$\mu(n) = \begin{cases} 0 & \exists i \in [1, m], C_i > 1 \\ (-1)^m & \forall i \in [1, m], C_i = 1 \end{cases}$$

### 构造莫比乌斯函数



设  $N$  分解因数  $N = p_1^{c_1} p_2^{c_2} \dots p_m^{c_m}$  (是积性函数)

$$\mu(N) = \begin{cases} 0 & \exists i \in [1, m], c_i > 1 \\ (-1)^m & \forall i \in [1, m], c_i = 1 \end{cases}$$

当  $N$  含相同质因子时,  $\mu(N) = 0$   
 当  $N$  的质因子各不相同  
 若  $N$  有奇数个质因子  $\mu(N) = 1$   
 若  $N$  有偶数个质因子  $\mu(N) = -1$

Möbius 函数由来: 算术函数  $f$ , 和函数  $F = \sum_{d|n} f(d)$  很明显  $F$  由  $f$  推出.

我们可以反过来吗? 也就是由  $F$  简便地推出  $f$ ? 我们展开  $F(n)$  可以发现一些规律.

$$F(1) = f(1), F(2) = f(1) + f(2), F(3) = f(1) + f(3), F(4) = f(1) + f(2) + f(4) \dots$$

$$\text{解得: } f(1) = F(1), f(2) = F(2) - F(1), f(3) = F(3) - F(1), f(4) = F(4) - F(2) - F(1) + F(1)$$

发现  $f(n)$  等于形式为  $\pm F(\frac{n}{d})$  的一些项之和. 其中  $d|n$ , 猜测存在一个等式:

$$f(n) = \sum_{d|n} \mu(d) F(\frac{n}{d}) \quad (\mu(d) \text{ 为我们当前未知待构造的函数 也就是 } F \text{ 的系数})$$

发现  $F(p) = f(1) + f(p) \Rightarrow f(p) = F(p) - F(1)$ , 其中  $p$  为质数, 构造出  $\mu(p) = -1$

$$F(p^2) = f(1) + f(p) + f(p^2) \Rightarrow f(p^2) = F(p^2) - (F(p) - F(1)) - F(1) = F(p^2) - F(p)$$

$$\text{即令 } \mu(p^2) = 0 \Rightarrow \text{规定对任意质数 } p, p^k, k > 1, \text{ 令 } \mu(p^k) = 0$$

若  $\mu$  为积性函数, 则  $\mu$  的值仅由质数  $p$  的幂的值决定

综上所述, 我们可以构造出一个全新的函数: Möbius 函数  $\mu$ .

$$\text{令 } \mu(N) = \begin{cases} 0, & \exists i \in [1, m], c_i > 1 \\ 1, & \forall i \in [1, m], c_i = 1 \end{cases} \quad \text{其中 } N = p_1^{c_1} p_2^{c_2} \dots p_m^{c_m}$$

我们构造出来的 Möbius 函数  $\mu$  即为莫比乌斯函数!

我们利用构造出来的 Möbius 函数  $\mu$  实现由和函数  $F$  求

得  $f$  的过程即为莫比乌斯反演!

我们猜测的公式  $f(n) = \sum_{d|n} \mu(d) F(\frac{n}{d})$  即为反演公式!!!

[https://blog.csdn.net/weixin\\_45697774](https://blog.csdn.net/weixin_45697774)

莫比乌斯函数可以理解为一个容斥原理的映射

性质 32.1:

$$\sum_{d|n} \mu(d) = \begin{cases} 1 & n = 1 \\ 0 & n \neq 1 \end{cases}$$

即  $\sum_{d|n} \mu(d) = \varepsilon(n)$ ,  $\mu * 1 = \varepsilon$ , 其中  $*$  是指 Dirichlet 卷积.

证明

$$\text{设 } n = \prod_{i=1}^k p_i^{c_i}, n' = \prod_{i=1}^k p_i$$

那么  $\sum_{d|n} \mu(d) = \sum_{d|n'} \mu(d) = \sum_{i=0}^k C_k^i \cdot (-1)^i = (1 + (-1))^k$

根据二项式定理，易知该式子的值在  $k = 0$  即  $n = 1$  时值为 1 否则为 0，这也同时证明了  $\sum_{d|n} \mu(d) = [n = 1] = \varepsilon(n)$  以及  $\mu * 1 = \varepsilon$

性质32.2:

$[gcd(i, j) = 1] = \sum_{d|gcd(i, j)} \mu(d)$

**直接证明：**如果看懂了上一个结论，这个结论稍加思考便可以推出：如果  $gcd(i, j) = 1$  的话，那么代表着我们按上个结论中枚举的那个  $n$  是 1，也就是式子的值是 1，反之，有一个与  $[gcd(i, j) = 1]$  相同的值：0

**利用  $\varepsilon$  函数：** $[gcd(i, j) = 1] = \varepsilon[gcd(i, j)] = \mu * I = \sum_{d|gcd(i, j)} \mu(d)$

证明  $\mu(x)$  是积性函数

设  $a, b$ :

$a = p_1^{\alpha_1} \times p_2^{\alpha_2} \times \cdots \times p_n^{\alpha_k} b = p_1^{\beta_1} \times p_2^{\beta_2} \times \cdots \times p_n^{\beta_t} a \times b = p_1^{\alpha_1} \times p_2^{\alpha_2} \times \cdots \times p_n^{\alpha_k} \times p_1^{\beta_1} \times p_2^{\beta_2} \times \cdots \times p_n^{\beta_t}$

若  $\mu(a) = 0$  或  $\mu(b) = 0$ , 则  $\mu(a \times b) = 0$  (因为均含有  $\alpha$  or  $\beta > 1$ )

即  $\mu(a \times b) = \mu(a) \times \mu(b)$

若  $\mu(a) \neq 0$  或  $\mu(b) \neq 0$ , 则  $\mu(a \times b) = (-1)^{k+t} = (-1)^k \times (-1)^t = \mu(a) \times \mu(b)$

## 0x33 狄利克雷卷积

- 定义：**若函数  $f, g$  为积性函数，则定义  $f, g$  的狄利克雷卷积为：

$f * g(n) = \sum_{d|n} f(d)g(\frac{n}{d})$

计算的时候可以把枚举约数转换成枚举倍数，以调和级数  $O(n \log n)$  的复杂度求出  $f * g$  的前  $n$  项。详见本文 **0x33.2  $O(n \log n)$  的卷积预处理**

- 满足性质：**

**交换律：**

$f * g(n) = \sum_{d|n} f(d)g(\frac{n}{d}) = \sum_{d|n} g(d)f(\frac{n}{d}) = g * f(n)$

**结合律：**

$f * g * h(n) = \sum_{d|n} f(d) \sum_{t|\frac{n}{d}} g(t)h(\frac{n}{dt}) = \sum_{d_1 d_2 d_3 = n} f(d_1)g(d_2)h(d_3) = f * (g * h)(n)$

**分配律：** $f * (g + h) = f * g + f * h$

**等式的性质：** $f = g$  的充要条件是  $f * h = g * h$ ，其中数论函数  $h(x)$  要满足  $h(1) \neq 0$ 。

**证明：**充分性是显然的。

证明必要性，我们先假设存在  $x$ ，使得  $f(x) \neq g(y)$ 。那么我们找到最小的  $y \in \mathbb{N}$ ，满足  $f(y) \neq g(y)$ ，并设  $r = f * h - g * h = (f - g) * h$ 。

则有：

$$\begin{aligned} r(y) &= \sum_{d|y} (f(d) - g(d))h\left(\frac{y}{d}\right) \\ &= (f(y) - g(y))h(1) \\ &\neq 0 \end{aligned}$$

则  $f * h$  和  $g * h$  在  $y$  处的取值不一样，即有  $f * h \neq g * h$ 。矛盾，所以必要性成立。

**证毕**

**单位元：** $f * \varepsilon = f$

**逆元：**对于任何一个满足  $f(x) \neq 0$  的数论函数，如果有另一个数论函数  $g(x)$  满足  $f * g = \varepsilon$ ，则称  $g(x)$  是  $f(x)$  的逆元。由 **等式的性质** 可知，逆元是唯一的。

根据定义显然有


$$\begin{aligned} \epsilon &= g * f \\ [n = 1] &= \sum_{i|n} f(i) g\left(\frac{n}{i}\right) \end{aligned}$$

$$[n = 1] = f(1) g(n) + \sum_{i|n, i \neq 1} f(i) g\left(\frac{n}{i}\right)$$

化简即可得到：



$$g(n) = \frac{[n = 1] - \sum_{i|n, i \neq 1} f(i) g(\frac{n}{i})}{f(1)}$$

**重要性质：** 若函数 $f, g$ 为积性函数，则 $f * g$ 也为积性函数（为狄利克雷特卷积）

**两个积性函数的 Dirichlet 卷积也是积性函数**

**证明：** 设两个积性函数为  $f(x)$  和  $g(x)$ ，再记  $h = f * g$ 。

设  $\gcd(a, b) = 1$ ，则：

$$h(a) = \sum_{d_1|a} f(d_1)g\left(\frac{a}{d_1}\right), h(b) = \sum_{d_2|b} f(d_2)g\left(\frac{b}{d_2}\right),$$

所以：

$$\begin{aligned} h(a)h(b) &= \sum_{d_1|a} f(d_1)g\left(\frac{a}{d_1}\right) \sum_{d_2|b} f(d_2)g\left(\frac{b}{d_2}\right) \\ &= \sum_{d|ab} f(d)g\left(\frac{ab}{d}\right) \\ &= h(ab) \end{aligned}$$

所以结论成立。

**证毕**

**积性函数的逆元也是积性函数**

**证明：** 我们设  $f * g = \varepsilon$ ，并且不妨设  $f(1) = 1$ 。考虑归纳法：

- 若  $nm = 1$ ，则  $g(nm) = g(1) = 1$ ，结论显然成立；
- 若  $nm > 1(\gcd(n, m) = 1)$ ，假设现在对于所有的  $xy < nm(\gcd(x, y) = 1)$ ，都有  $g(xy) = g(x)g(y)$ ，所以有：

$$g(nm) = - \sum_{d|nm, d \neq 1} f(d)g\left(\frac{nm}{d}\right) = - \sum_{a|n, b|m, ab \neq 1} f(ab)g\left(\frac{nm}{ab}\right)$$

又因为  $\frac{nm}{ab} < nm$ ，所以有：

$$\begin{aligned} g(nm) &= - \sum_{a|n, b|m, ab \neq 1} f(ab)g\left(\frac{nm}{ab}\right) \\ &= - \sum_{a|n, b|m, ab \neq 1} f(a)f(b)g\left(\frac{n}{a}\right)g\left(\frac{m}{b}\right) \\ &= f(1)f(1)g(n)g(m) - \sum_{a|n, b|m} f(a)f(b)g\left(\frac{n}{a}\right)g\left(\frac{m}{b}\right) \\ &= g(n)g(m) - \sum_{a|n} f(a)g\left(\frac{n}{a}\right) \sum_{b|m} f(b)g\left(\frac{m}{b}\right) \\ &= g(n)g(m) - \varepsilon(n) - \varepsilon(m) \\ &= g(n)g(m) \end{aligned}$$

综合以上两点，结论成立。

**证毕**

**0x33.1 常见积性函数的卷积**

**性质0x33.1：**  $\forall f(n), e * f(n) = f(n)$

**性质0x33.2：**  $1 * 1(n) = \sum_{d|n} 1 = d(n)$

**性质0x33.3：**  $id * 1(n) = \sum_{d|n} d = \sigma(n)$

**性质0x33.4：**  $\mu * 1(n) = \sum_{d|n} \mu(d) \times 1(\frac{n}{d}) = [n = 1] = \varepsilon(n)$

**性质0x33.4证明：**

由唯一分解定理：  $n = p_1^{k_1} p_2^{k_2} \dots p_m^{k_m}$

代入上式得：

$$\begin{aligned}\mu * 1(n) &= \sum_{d|n} \mu(d) \times 1\left(\frac{n}{d}\right) \\ &= \sum_{C_1=0}^{k_1} \sum_{C_2=0}^{k_2} \cdots \sum_{C_m=0}^{k_m} \mu(p_1^{C_1} p_2^{C_2} \cdots p_m^{C_m})\end{aligned}$$

我们分析该和式的实际意义，显然当某个质因子数量  $c_i > 1$ ，则  $\mu$  的值为 0。这样的话，我们就可以去掉无用的枚举（ $\mu = 0$ ），利用二项式定理证明：

$$\begin{aligned}\mu * 1(n) &= \sum_{C_1=0}^1 \sum_{C_2=0}^1 \cdots \sum_{C_m=0}^1 \mu(p_1^{C_1} p_2^{C_2} \cdots p_m^{C_m}) \\ &= \sum_{C_1=0}^1 \sum_{C_2=0}^1 \cdots \sum_{C_m=0}^1 (-1)^{\sum_{i=1}^m C_i} \\ &= \sum_{i=0}^m (-1)^i \binom{m}{i} \\ &= [m = 0] = [n = 1] = \varepsilon(n)\end{aligned}$$

**性质0x33.5：**  $\varphi * 1(n) = \sum_{d|n} \varphi(d) = n = id(n)$

**性质0x33.5证明：**

将  $n$  分解质因数：  $n = \prod_{i=1}^k p_i^{c_i}$

首先，因为  $\varphi$  是积性函数，故只要证明当  $n' = p^c$  时  $\varphi * 1 = \sum_{d|n'} \varphi(\frac{n'}{d}) = id$  成立即可。

因为  $p$  是质数，于是  $d = p^0, p^1, p^2, \cdots, p^c$

易知如下过程：

$$\begin{aligned}\varphi * 1 &= \sum_{d|n} \varphi\left(\frac{n}{d}\right) \\ &= \sum_{i=0}^c \varphi(p^i) \\ &= 1 + p^0 \cdot (p - 1) + p^1 \cdot (p - 1) + \cdots + p^{c-1} \cdot (p - 1) \\ &= p^c \\ &= id\end{aligned}$$

该式子两侧同时卷  $\mu$  可得  $\varphi(n) = \sum_{d|n} d \cdot \mu(\frac{n}{d})$

$$\begin{aligned}\because \varphi &= \mu * id, \epsilon = \mu * 1 \\ \therefore 1 * \varphi &= 1 * \mu * id \\ \therefore 1 * \varphi &= \epsilon * id = id(n) \\ \therefore \sum_{d|n} \varphi(d) &= n\end{aligned}$$

**积性函数的转换关系：**

$$\mu \Rightarrow *1 \Rightarrow \varepsilon \Rightarrow *1 \Rightarrow 1 \Rightarrow *1 \Rightarrow d$$

$$\varphi \Rightarrow *1 \Rightarrow id \Rightarrow *1 \Rightarrow \sigma$$

**反方向的转换关系：**

$$\mu \Leftarrow * \mu \Leftarrow \varepsilon \Leftarrow * \mu \Leftarrow 1 \Leftarrow * \mu \Leftarrow d$$

$$\varphi \Leftarrow * \mu \Leftarrow id \Leftarrow * \mu \Leftarrow \sigma$$

### 0x33.2 $O(n \log n)$ 的卷积预处理

若已知数论函数  $f, g$ ，我么可以将枚举约数转换成枚举倍数，以调和级数  $O(n \log n)$  的复杂度求出  $f * g$  的前  $n$  项：

$$h(n) = f * g(n) = \sum_{d|n} f(d)g\left(\frac{n}{d}\right)$$

推荐这个写法：

```
1 void Dirichlet(ll *f, ll *g)
2 {
3     int h[N] = {0};
4     for(int i = 1; i <= n; ++ i) {
5         for(int j = i; j <= n; j += i) {
6             h[j] = (h[j] + f[i] * g[j / i]) % mod;
7         }
8     }
9     for(int i = 1; i <= n; ++ i)
10         f[i] = h[i];
11 }
```

另一种写法:

```
1 int f[N], g[N], h[N];
2 inline void init(int n) {
3     for (int i = 1; i * i <= n; i++) {
4         for (int j = i; i * j <= n; j++) {
5             if (j == i) h[i * j] += f[i] * g[i];
6             else h[i * j] += f[i] * g[j] + f[j] * g[i];
7         }
8     }
9 }
```

• 竞赛例题选讲

Problem A. Longge的问题 ([P2303 [SDOI2012]](<https://www.luogu.com.cn/problem/P2303>))

Problem

现在问题来了: 给定一个整数  $n$ , 你需要求出  $\sum_{i=1}^n \gcd(i, n)$ , 其中  $\gcd(i, n)$  表示  $i$  和  $n$  的最大公因数。

$$1 \leq n \leq 2^{32}$$

Solution

设  $\gcd(i, n) = d$ , 则  $\gcd(\frac{i}{d}, \frac{n}{d}) = 1$

$d$  显然就是  $n$  的因数

我们对于每个  $d$ , 要求有多少  $i$  使得  $\gcd(\frac{i}{d}, \frac{n}{d}) = 1$ , 设求出有  $x$  个  $i$ , 那么对答案的贡献就是  $d \times x$ 。为什么是这些贡献? 很显然, 这些  $i$  与  $n$  的 gcd 就是  $d$ , 共  $x$  个这样的  $i$ , 所以是  $d \times x$ 。

因为满足  $\gcd(\frac{i}{d}, \frac{n}{d}) = 1$  的  $\frac{i}{d}$  的个数就是与  $\frac{n}{d}$  互质的数, 每个  $\frac{i}{d}$  又对应一个  $i$ , 个数就是  $\varphi(\frac{n}{d})$ 。前面说了  $d$  是  $n$  的因数, 我们枚举所有因数, 累加  $\varphi(\frac{n}{d})$  即可。

对于每个  $\varphi(\frac{n}{d})$ ,  $\sqrt{n}$  求即可

当然利用欧拉反演可以直接得到一个一模一样的公式:

$$\begin{aligned} \sum_{i=1}^n \gcd(i, n) &= \sum_{i=1}^n \sum_{d|i} \sum_{d|n} \varphi(d) \\ &= \sum_{d|n} \sum_{i=1}^n \sum_{d|i} \varphi(d) \\ &= \sum_{d|n} \left\lfloor \frac{n}{d} \right\rfloor \varphi(d) \end{aligned}$$

Code

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4 #define int long long
5 const int N = 50007;
6
7 int n, m, t;
8 int ans;
9
10 int get_phi(int n)
```

```

11 {
12     int ans = n;
13     for(int i = 2; i * i <= n; ++ i) {
14         if(n % i == 0) {
15             ans = ans / i * (i - 1);
16             while(n % i == 0) n /= i;
17         }
18     }
19     if(n > 1) ans = ans / n * (n - 1);
20     return ans;
21 }
22
23 signed main()
24 {
25     scanf("%lld", &n);
26     int ans = 0;
27     for(int i = 1; i * i <= n; ++ i) {
28         if(n % i == 0) {
29             ans += n / i * get_phi(i);
30             if(i * i != n) {
31                 ans += i * get_phi(n / i);
32             }
33         }
34     }
35     printf("%lld\n", ans);
36     return 0;
37 }

```

### Problem B. Clarke and math (HDU 5628)

给定  $f(i)$ ,  $(i = 1, 2, \dots, n)$

求：

$$g(i) = \sum_{i_1|i} \sum_{i_2|i_1} \sum_{i_3|i_2} \cdots \sum_{i_k|i_{k-1}} f(i_k) \quad \text{mod } 1000000007 (1 \leq i \leq n)$$

### Solution

$f$  和  $g$  进行一次Dirichlet卷积是  $h(n) = f * g(n) = \sum_{d|n} f(d)g(\frac{n}{d})$ , 显然可以发现题目中的式子实际上就是  $f * 1^k$ , 其中  $1$  是不变函数, 即  $1(n) = 1$ . 使用快速幂加速即可。

### Time

$$O(n \log^2 n)$$

### Code

```

1 // Problem: Clarke and math
2 // Contest: HDUJ
3 // URL: http://acm.hdu.edu.cn/showproblem.php?pid=5628
4 // Memory Limit: 65 MB
5 // Time Limit: 5000 ms
6 //
7 // Powered by CP Editor (https://cpeditor.org)
8
9 #include <bits/stdc++.h>
10
11 using namespace std;
12 using ll = long long;
13 const int N = 5e5 + 7, mod = 1e9 + 7;
14
15 int n, m, k;
16 ll f[N], yi[N], res[N];
17
18 void Dirichlet(ll *f, ll *g)
19 {
20     int h[N] = {0};
21     for(int i = 1; i <= n; ++ i) {
22         for(int j = i; j <= n; j += i) {
23             h[j] = (h[j] + f[i] * g[j / i]) % mod;

```

```
24     }
25 }
26 for(int i = 1; i <= n; ++ i)
27     f[i] = h[i];
28 }
29
30 void fpow(ll *res, ll *yi, int k)
31 {
32     while(k) {
33         if(k & 1) Dirichlet(res, yi);
34         Dirichlet(yi, yi);
35         k >>= 1;
36     }
37 }
38
39 void solve()
40 {
41     scanf("%d%d", &n, &k);
42     for(int i = 1; i <= n; ++ i) {
43         scanf("%lld", &f[i]);
44         res[i] = 0;
45         yi[i] = 1;
46     }
47     res[1] = 1;
48     fpow(res, yi, k);
49     Dirichlet(f, res);
50
51     for(int i = 1; i <= n; ++ i) {
52         printf("%lld%s", f[i], i == n ? "\n" : " ");
53     }
54 }
55
56 int main()
57 {
58     int t;
59     scanf("%d", &t);
60     while(t -- ) {
61         solve();
62     }
63     return 0;
64 }
```

## ox40 反演

### Ox41 整除分块

#### ox41.1 前置知识

##### 引理 1

$$\forall a,b,c \in \mathbb{Z}, \left\lfloor \frac{a}{bc} \right\rfloor = \left\lfloor \frac{\left\lfloor \frac{a}{b} \right\rfloor}{c} \right\rfloor$$

##### 证明

$$\begin{aligned} \frac{a}{b} &= \left\lfloor \frac{a}{b} \right\rfloor + r (0 \leq r < 1) \\ \Rightarrow \left\lfloor \frac{a}{bc} \right\rfloor &= \left\lfloor \frac{a}{b} \cdot \frac{1}{c} \right\rfloor = \left\lfloor \frac{1}{c} \left( \left\lfloor \frac{a}{b} \right\rfloor + r \right) \right\rfloor = \left\lfloor \frac{\left\lfloor \frac{a}{b} \right\rfloor}{c} + \frac{r}{c} \right\rfloor = \left\lfloor \frac{\left\lfloor \frac{a}{b} \right\rfloor}{c} \right\rfloor \end{aligned}$$

□

##### 引理 2

$$\forall n \in \mathbb{N}_+, \left| \left\{ \left\lfloor \frac{n}{d} \right\rfloor \mid d \in \mathbb{N}_+, d \leq n \right\} \right| \leq \lfloor 2\sqrt{n} \rfloor$$

$|V|$  表示集合  $V$  的元素个数

##### 证明

对于  $0 \leq d \leq \lfloor \sqrt{n} \rfloor$ , 显然一共最多只有  $\sqrt{n}$  种  $d$ , 显然  $\left\lfloor \frac{n}{d} \right\rfloor$  有  $\lfloor \sqrt{n} \rfloor$  种取值。

对于  $d > \lfloor \sqrt{n} \rfloor$ , 有  $0 \leq \lfloor \frac{n}{d} \rfloor \leq \lfloor \sqrt{n} \rfloor$ , 显然也只有  $\lfloor \sqrt{n} \rfloor$  种取值

综上, 得证  $\square$

## 41.2 整除分块

我们在计算  $\sum_{i=1}^n \lfloor \frac{n}{i} \rfloor$  的时候, 如果直接枚举的话, 复杂度是  $O(n)$  的, 在  $n$  很大的时候是很难计算的。但是当我们打表找规律的时候, 发现序列  $\lfloor \frac{n}{i} \rfloor$  中有很多是相同的! (因为我们这里是下取整的形式, 会丢掉所有小数), 我们考虑对于元素相同的段  $O(1)$  计算, 即求出该段的长度, 然后乘上该段的值显然就是该段的和。

因此我们考虑对于含有  $\lfloor \frac{n}{i} \rfloor$  的求和式子 ( $n$  为给定的常数)

对于任意一个  $i (i \leq n)$ , 我们需要找到一个**最大的**  $j (i \leq j \leq n)$ , 使得  $\lfloor \frac{n}{i} \rfloor = \lfloor \frac{n}{j} \rfloor$ , 显然此时的  $i, j$  就是这段值相等的序列。

由引理1显然可以得到  $j = \lfloor \frac{n}{\lfloor \frac{n}{i} \rfloor} \rfloor$  (这里的除法都是下取整)

显然  $j \leq n$ , 考虑证明  $j \geq i$ :

$$\begin{aligned} \lfloor \frac{n}{i} \rfloor &\leq \frac{n}{i} \\ \Rightarrow \left\lfloor \frac{n}{\lfloor \frac{n}{i} \rfloor} \right\rfloor &\geq \left\lfloor \frac{n}{\frac{n}{i}} \right\rfloor = \lfloor i \rfloor = i \\ \Rightarrow i &\leq \left\lfloor \frac{n}{\lfloor \frac{n}{i} \rfloor} \right\rfloor = j \end{aligned}$$

$\square$

不妨设  $k = \lfloor \frac{n}{i} \rfloor$ , 考虑证明当  $\lfloor \frac{n}{j} \rfloor = k$  时,  $j$  的最大值为  $\lfloor \frac{n}{k} \rfloor$ :

$$\left\lfloor \frac{n}{j} \right\rfloor = k \iff k \leq \frac{n}{j} < k+1 \iff \frac{1}{k+1} < \frac{j}{n} \leq \frac{1}{k} \iff \frac{n}{k+1} < j \leq \frac{n}{k}$$

又因为  $j$  为整数 所以  $j_{\max} = \lfloor \frac{n}{k} \rfloor = \left\lfloor \frac{n}{\lfloor \frac{n}{i} \rfloor} \right\rfloor$

综上所述, 值相等的区间为  $[i, j] = [i, \left\lfloor \frac{n}{\lfloor \frac{n}{i} \rfloor} \right\rfloor]$

综上所述, 我们每次以  $[i, j]$  为一块, 分块求和即可。

### Code

```
1 for(ll l = 1, r; l <= n; l = r + 1){
2     r = n / (n / l);
3     ans += (r - l + 1) * (n / l);
4 }
```

根据 **引理2**, 显然该过程的时间复杂度为  $O(\sqrt{n})$

### 二维数论分块

$$\sum_{i=1}^{\min(n,m)} \left\lfloor \frac{n}{i} \right\rfloor \left\lfloor \frac{m}{i} \right\rfloor$$

每次右边界取  $\min(\left\lfloor \frac{n}{\lfloor \frac{n}{i} \rfloor} \right\rfloor, \left\lfloor \frac{m}{\lfloor \frac{m}{i} \rfloor} \right\rfloor)$ , 保证了两者分别相等, 所以每两项的乘积相等。复杂度并没有变化。

### Code

```
1 for(ll l = 1, r; l <= min(n, m); l = r + 1){
2     r = min(n / (n / l), m / (m / l));
3     ans += (r - l + 1) * (n / l) * (m / l);
4 }
```

### Example A

计算



$$\sum_{i=1}^n i \left\lfloor \frac{n}{i} \right\rfloor$$

**Solution**

对于每个块均有：

$$\begin{aligned} \sum_{i=l}^r i \left\lfloor \frac{n}{i} \right\rfloor &= \sum_{i=l}^r i \left\lfloor \frac{n}{l} \right\rfloor \\ &= \left\lfloor \frac{n}{l} \right\rfloor \sum_{i=l}^r i \\ &= \left\lfloor \frac{n}{l} \right\rfloor \times \frac{(r-l+1)(l+r)}{2} \end{aligned}$$

**Example B**

计算

$$\sum_{i=1}^n i^2 \left\lfloor \frac{n}{i} \right\rfloor$$

**Solution**

同样可以整除分块，只不过把等差数列求和公式换成平方数列求和的公式

**Code**

```
1 int cal(int x){
2     return x * (x + 1) * (2 * x + 1) / 6;
3 }
4 int n, ans;
5 signed main(){
6     cin >> n;
7     for(int l = 1, r; l <= n; l = r + 1){
8         r = n / (n / l);
9         ans += (cal(r) - cal(l - 1)) * (n / l);
10    }
11    printf("%lld\n", ans);
12 }
```

**Example C**

计算

$$\sum_{k=1}^n \sum_{k|x}^n x$$

**Solution**

显然它的实际意义就是求  $k$  在  $[1, n]$  内的所有倍数和。

即对于每个  $k, n$  里会存在  $\left\lfloor \frac{n}{k} \right\rfloor$  个  $k$  的倍数，而且是一个等差数列，公差为  $k$ 。所以式可以写成

$$\sum_{k=1}^n \frac{k(\left\lfloor \frac{n}{k} \right\rfloor + 1) \left\lfloor \frac{n}{k} \right\rfloor}{2}$$

显然可以直接整除分块：

```
1 int n, ans1, ans2;
2 signed main(){
3     cin >> n;
4     for(int k = 1; k <= n; ++ k)
5         for(int j = 1; j * k <= n; ++ j)
6             ans1 += j * k;
7     for(int l = 1, r; l <= n; l = r + 1){
8         r = n / (n / l);
9         ans2 += (1 + r) * (r - l + 1) / 2 * (1 + n / l) * (n / l) / 2;
10    }
11    printf("%lld %lld\n", ans1, ans2);
12 }
```

**Example E**

计算

$$\sum_{i=1}^n \left\lceil \frac{n}{i} \right\rceil$$

**Solution**

$$\left\lceil \frac{n}{i} \right\rceil = \begin{cases} \left\lfloor \frac{n}{i} \right\rfloor & i \mid n \\ \left\lfloor \frac{n}{i} \right\rfloor + 1 & i \nmid n \end{cases}$$

一共有  $d(n)$  (表示  $n$  的因数个数) 个数满足  $i \mid n$  , 我们计算全部都不能整除时的结果, 最后减去因数个数。

$$\begin{aligned} \sum_{i=1}^n \left\lceil \frac{n}{i} \right\rceil &= \sum_{i=1}^n \left( \left\lfloor \frac{n}{i} \right\rfloor + 1 \right) - d(n) \\ &= n - d(n) + \sum_{i=1}^n \left\lfloor \frac{n}{i} \right\rfloor \end{aligned}$$

Example F

计算

$$\sum_{i=1}^n \left\lfloor \frac{n}{i^2} \right\rfloor$$

Solution

令  $l' = l^2$  ,  $r' = r^2$  , 由引理1可得  $r' = \left\lfloor \frac{n}{\lfloor \frac{n}{l'} \rfloor} \right\rfloor$  。

故

$$r = \left\lfloor \sqrt{\left\lfloor \frac{n}{\lfloor \frac{n}{l^2} \rfloor} \right\rfloor} \right\rfloor$$

Example G

已知  $n, a, b$  为正整数, 求  $\sum_{i=1}^n \left\lfloor \frac{n}{ai+b} \right\rfloor$

Solution

令  $l' = al + b$  ,  $r' = ar + b$  , 由引理1可得:

$$r' = \left\lfloor \frac{n}{\lfloor \frac{n}{l'} \rfloor} \right\rfloor$$

故

$$\begin{aligned} ar + b &= \left\lfloor \frac{n}{\lfloor \frac{n}{l'} \rfloor} \right\rfloor \\ r &= \left\lfloor \frac{\left\lfloor \frac{n}{\lfloor \frac{n}{l'} \rfloor} \right\rfloor - b}{a} \right\rfloor \end{aligned}$$

Example H

一般地, 对于和式:  $\sum_{i=1}^n \left\lfloor \frac{k}{f(i)} \right\rfloor \bmod p$

其中  $n, p, k$  是正整数,  $1 \leq n \leq 10^{12}$ , 分块边界推导方法是:

$$\left\lfloor \frac{k}{f(l)} \right\rfloor = \left\lfloor \frac{k}{f(r)} \right\rfloor$$

由 引理1 可得

$$f(r) = \left\lfloor \frac{n}{\left\lfloor \frac{n}{f(l)} \right\rfloor} \right\rfloor$$

我们只要求出  $f(l)$  , 然后再解这个关于  $r$  的方程, 最后向下取整, 就得到了该块的右边界。

• 竞赛例题选讲

Problem A. 余数求和 ([Luogu P2261[CQOI2007]](<https://www.luogu.com.cn/problem/P2261>))

计算

$$ans = \sum_{i=1}^n (k \bmod i)$$

Solution

显然有:

$$ans = \sum_{i=1}^n (k \bmod i) = \sum_{i=1}^n k - i \left\lfloor \frac{k}{i} \right\rfloor$$

整除分块即可。

Code

```

1  typedef long long ll;
2  const int N = 20007, M = 50007, INF = 0x3f3f3f3f;
3  ll n, k;
4  int main()
5  {
6      scanf("%lld%lld", &n, &k);
7      ll ans = n * k;
8      for(ll l = 1, r; l <= n; l = r + 1){
9          if(k / l != 0)r = min((k / (k / l)) , n);
10         else r = n;
11         ans -= (r + 1) * (k / l) * (r - l + 1) / 2; //先乘后除，因为中间除2可能不能整除
12         //区间内i的平均值 * (k / l) * 区间长度
13     }
14     printf("%lld\n", ans);
15     return 0;
16 }
```

Problem B 小G的约数 (牛客练习赛77C)

小G定义了两个函数  $F(n)$  为  $n$  的约数和,  $G(n)$  为  $F(1) + F(2) + \dots + F(n - 1) + F(n)$

小G想知道  $G(G(n))$  等于多少。

$$n \leq 5 \times 10^4$$

Solution

我们知道对于  $i \in [1, n]$ , 约数为  $i$  的整数的个数为  $\left\lfloor \frac{n}{i} \right\rfloor$

显然  $1 \sim n$  的约数和为:

$$G(n) = \sum_{i=1}^n \left\lfloor \frac{n}{i} \right\rfloor \times i$$

显然前面的  $\left\lfloor \frac{n}{i} \right\rfloor$  可以直接整除分块, 后面的  $i$  相当于对于区间  $[l, r]$ ,  $\left\lfloor \frac{n}{i} \right\rfloor \times (l + (l + 1) + \dots + (r - 1) + r) = \left\lfloor \frac{n}{i} \right\rfloor \times \frac{l+r}{2}$ ,  $O(\sqrt{n})$  的复杂度实现。

Code

```

1  #define int long long
2  const int N = 5e5 + 7, mod = 1e9 + 7;
3  const double PI = acos(-1.0);
4
5  int n, m;
6
7  int G(int n)
8  {
9      int res = 0;
10     for(int l = 1, r; l <= n; l = r + 1) {
11         r = n / (n / l);
12         res += (r - l + 1) * (l + r) / 2 * (n / l);
13     }
14     return res;
15 }
16
17 signed main()
18 {
19     scanf("%lld", &n);
20     printf("%lld\n", G(G(n)));
21     return 0;
22 }
```

Problem C. Dividing (2020牛客多校 7 - H)

在  $1 \leq n \leq N, 1 \leq k \leq K$  范围内, 有多少对  $(n, k)$  是传奇对。传奇对的条件是:

$(1, k)$  一定是传奇对;

若  $(n, k)$  是传奇对,  $(n + k, k)$  和  $(nk, k)$  都是传奇对。

Solution

Problem D. Floor and Mod (CF1485C Floor and Mod)

Translation

定义一对数  $(a, b)$  是特殊的, 当且仅当  $\left\lfloor \frac{a}{b} \right\rfloor = a \bmod b$

给定  $x$  和  $y$ , 求一共有多少对特殊的数  $(a, b)$ , 当  $1 \leq a \leq x, 1 \leq b \leq y$ .

Solution

显然是一个结论题或者枚举题, 我们根据题目中给定的柿子:

$$\begin{aligned}\left\lfloor \frac{a}{b} \right\rfloor &= a \bmod b \\ \left\lfloor \frac{a}{b} \right\rfloor &= a - \left\lfloor \frac{a}{b} \right\rfloor \times b \\ a &= \left\lfloor \frac{a}{b} \right\rfloor \times (b + 1) \\ \frac{a}{b + 1} &= \left\lfloor \frac{a}{b} \right\rfloor\end{aligned}$$

因为  $\left\lfloor \frac{a}{b} \right\rfloor$  一定是整数, 所以  $\frac{a}{b + 1}$  一定是整数, 所以  $b + 1 \mid a$ .

所以一个很直观的想法就是枚举  $b$ , 对于每一个  $b$ ,  $b + 1$  的倍数  $a$  显然有  $\frac{x}{b + 1}$  个, 显然可以整除分块。

但是需要注意题目中的给定的条件为  $\left\lfloor \frac{a}{b} \right\rfloor = a \bmod b$

即:  $\left\lfloor \frac{a}{b} \right\rfloor < b$

即:  $\frac{a}{b + 1} < b \Rightarrow a < b^2 + b$

即  $a \in [0, b^2 + b - 1]$ 。那么符合题意的  $a$  的个数就是:  $\frac{b^2 + b - 1}{b + 1}$

并且  $a$  的上限是  $x$ , 故最终的答案为:

$$\sum_{b=1}^y ans = \frac{\min\{x, b^2 + b - 1\}}{b + 1}$$

我们可以先枚举  $b < \sqrt{x}$ , 计算  $\frac{b^2 + b - 1}{b + 1}$

剩余的  $b = \sqrt{x} \sim y$ , 显然  $\min\{x, b^2 + b - 1\} = x$ , 直接整除分块即可, 其中整除分块的时候, 我们需要计算的是  $\left\lfloor \frac{x}{b + 1} \right\rfloor$ , 我们令  $l, r$  代表  $b + 1$  即可。

时间复杂度  $O(\sqrt{x})$

Code

```
1 using ll = long long;
2 const int N = 1e5 + 7;
3
4 int n, m, t;
5 int x, y;
6
7 int main()
8 {
9     scanf("%d", &t);
10    while(t -- ) {
11        scanf("%d%d", &x, &y);
12        int a = 1, b = 1;
13        ll ans = 0;
14        for(; b * b + b - 1 < x && b <= y; ++ b)
15            ans += (b * b + b - 1) / (b + 1);
16
17        int l = b + 1, r;
18        for(; l <= x && l <= y + 1; l = r + 1) {
19            r = min(x / (x / l), y + 1);
20            ans += x / l * (r - l + 1);
21            if(r == y + 1) break;
22        }
23        cout << ans << endl;
24    }
25    return 0;
26 }
```

## Ox42 莫比乌斯反演

### 莫比乌斯函数

莫比乌斯函数已在 **Ox30 积性函数** 一章做过详细讲解和性质证明，这里罗略一些常用的性质，在讲解莫比乌斯反演的时候将会用到它们。

#### 定义

$$\mu(n) = \begin{cases} 0 & \exists i \in [1, m], C_i > 1 \\ (-1)^m & \forall i \in [1, m], C_i = 1 \end{cases}$$

- 基本性质定理

#### 性质32.1:

$$\sum_{d|n} \mu(d) = \begin{cases} 1 & n = 1 \\ 0 & n \neq 1 \end{cases}$$

即  $\sum_{d|n} \mu(d) = \varepsilon(n)$ ,  $\mu * 1 = \varepsilon$ , 其中  $*$  是指 *Dirichlet* 卷积。

#### 性质32.2:

$$[\gcd(i, j) = 1] = \sum_{d|\gcd(i, j)} \mu(d)$$

### Ox42.1 莫比乌斯反演公式

设  $F(n), f(n)$  为两个数论函数。

#### 形式一:

$$\text{如果有 } F(n) = \sum_{d|n} f(d), \text{ 那么有 } f(n) = \sum_{d|n} \mu(d) F\left(\frac{n}{d}\right).$$

#### 形式二:

$$\text{如果有 } F(n) = \sum_{n|d} f(d), \text{ 那么有 } f(n) = \sum_{n|d} \mu\left(\frac{d}{n}\right) F(d).$$

一般在求解数论函数  $f$  无法直接求解的时候，就可以构造  $F(n) = \sum_{d|n} f(d)$  或  $F(n) = \sum_{n|d} f(d)$ ，求出更好求解的  $F(n)$ ，此时利用枚举倍数的技巧在

$O(n \log n)$  的复杂度下利用反演公式求解出待求的  $f$  即可。

### Ox42.2 莫比乌斯反演证明

建议掌握反演的证明，因为我们在写莫比乌斯反演题目的时候，大多数都是直接推式子而不是使用反演公式。

#### 形式一证明

**证明一：** 对原式做数论变换。

$$\begin{aligned} \sum_{d|n} \mu(d) F\left(\frac{n}{d}\right) &= \sum_{d|n} \mu(d) \sum_{k|\frac{n}{d}} f(k) \\ &= \sum_{k|n} f(k) \sum_{d|\frac{n}{k}} \mu(d) \\ &= f(n) \end{aligned}$$

**解释：** 根据定义  $F(n) = \sum_{d|n} f(d)$ ，我们使用  $\sum_{d|n} f(d)$  来替换  $F\left(\frac{n}{d}\right)$ ，再交换求和顺序（ $\frac{n}{d}$  意味着  $n$  的所有因子， $k|\frac{n}{d}$  则  $k$  也是  $n$  的所有因子，即

单独枚举  $k|n$ ，此时  $d|n = d|\frac{n}{k}$ ，我们可以将  $d|n$  换成  $\frac{n}{k}$ ，是完全等价的）。最后根据莫比乌斯函数的 **性质32.1**： $\sum_{d|n} \mu(d) = [n = 1]$ ，因此在

$\frac{n}{k} = 1$  时第二个和式的值才为 1，否则全是 0。此时  $n = k$ ，故原式等价于

$$\sum_{k|n} [n = k] \cdot f(k) = f(n)$$

反演公式得证  $\square$

**证明二：** 利用 *Dirichlet* 卷积。

反演公式

设  $F(n), f(n)$  为两个数论函数。

$$\text{如果有 } F(n) = \sum_{d|n} f(d), \text{ 那么有 } f(n) = \sum_{d|n} \mu(d) F\left(\frac{n}{d}\right).$$

显然可以转化为 *Dirichlet* 卷积下的，已知  $F = f * 1$ ，证明  $f = F * \mu$

显然有：

$$\begin{aligned} F * \mu &= f * 1 * \mu \\ &= f * \varepsilon \\ &= f \end{aligned}$$

$$(1 * \mu = \varepsilon, \varepsilon * f = f)$$

反演公式得证  $\square$

形式二证明

证明一： 我们考虑逆推这个式子。

$$\begin{aligned} \sum_{n|d} \mu(\frac{d}{n}) F(d) &= \sum_{k=1}^{+\infty} \mu(k) F(kn) \\ &= \sum_{k=1}^{+\infty} \mu(k) \sum_{kn|d} f(d) \\ &= \sum_{n|d} f(d) \sum_{k|\frac{d}{n}} \mu(k) \\ &= \sum_{n|d} f(d) \epsilon(\frac{d}{n}) \\ &= \sum_{n|d} f(d) [\frac{d}{n} = 1] \\ &= \sum_{n|d} f(d) [n = d] \\ &= f(n) \end{aligned}$$

我们把  $d$  表示为  $kn$  的形式，然后把  $F(n) = \sum_{n|d} f(d)$  代入式子。

发现枚举  $k$  再枚举  $kn$  的倍数可以转换为直接枚举  $n$  的倍数再求出  $k$ ，发现后面那一块其实就是  $\epsilon$ ，式子只有在  $d = n$  的时候才不为 0。

0x42.3 推导结论

一些常用结论

(1) GCD与LCM

结论42.1.1:  $[\gcd(i, j) = 1] = \sum_{d|i, d|j} \mu(d)$

结论42.1.2:  $\sum_{i=1}^n \sum_{j=1}^m [\gcd(i, j) = k] = \sum_{d=1}^{\lfloor \frac{n}{k} \rfloor} \mu(d) \lfloor \frac{n}{dk} \rfloor \lfloor \frac{m}{dk} \rfloor$  (例题)

结论42.1.3:  $\sum_{i=1}^n \sum_{j=1}^m [\gcd(i, j) \in \{\text{Prime}\}] = \sum_{d=1}^n \left( \lfloor \frac{n}{d} \rfloor \lfloor \frac{m}{d} \rfloor \sum_{p|d \ \& \ p \in \{\text{Prime}\}} \mu(\frac{d}{p}) \right)$  (例题)

结论42.1.4:  $\sum_{i=1}^n \sum_{j=1}^m \text{lcm}(i, j) = \sum_{d=1}^n d \left( \sum_{x=1}^{\lfloor \frac{n}{d} \rfloor} x^2 \mu(x) \sum_{i=1}^{\lfloor \frac{n}{dx} \rfloor} i \sum_{j=1}^{\lfloor \frac{m}{dx} \rfloor} j \right)$  (例题)

(2) 除数函数

结论42.2.1:  $\sigma_k = \sum_{d|n} id^k$ , 即 $\sigma_k = \text{id}_k * 1$

$\sigma_0(x)$  表示  $x$  的约数个数

结论42.2.2:  $\sigma_0(xy) = \sum_{i|x} \sum_{j|y} [\gcd(i, j) = 1]$

结论42.2.3:  $\sum_{i=1}^n \sigma_0(i) =$  (例题)

结论42.2.4:  $\sum_{i=1}^n \sum_{j=1}^m \sigma_0(ij) = \sum_{k=1}^n \mu(k) \left( \sum_{i=1}^{\lfloor \frac{n}{k} \rfloor} \lfloor \frac{n}{ik} \rfloor \right) \left( \sum_{i=1}^{\lfloor \frac{m}{k} \rfloor} \lfloor \frac{m}{ik} \rfloor \right)$  (例题)

$\sigma_1(x)$  表示  $x$  的约数和

结论42.2.5:  $d(xy) = \sum_{i|x} \sum_{j|y} \frac{iy}{j} [\gcd(i, j) = 1]$

结论42.2.6:  $\sum_{i=1}^n \sum_{j=1}^n \sigma_1(ij) = \sum_{d=1}^n \mu(d) d \left( \sum_{i=1}^{\lfloor \frac{n}{d} \rfloor} \sigma_1(i) \right)^2$  (例题)



**结论42.2.7:**  $\sum_{i=1}^n \sum_{j=1}^m \sigma_1(\gcd(i, j)) = \sum_{d=1}^n \lfloor \frac{n}{d} \rfloor \lfloor \frac{m}{d} \rfloor \left( \sum_{i|d} \mu(\frac{d}{i}) \sigma_1(i) \right)$  (例题)

(3) 莫比乌斯函数

**结论42.3.1:**  $\sum_{k=1}^n \mu^2(k) = \sum_{d=1}^{\sqrt{n}} \mu(d) \lfloor \frac{n}{d^2} \rfloor$  (例题)

**结论42.3.2:**  $\sum_{i=1}^n \mu^2(i) \sqrt{\frac{n}{i}} = n$  (例题)

Ox42.4 竞赛例题选讲

**Problem A. problem b** ([P2522 [HAOI2011]](https://www.luogu.com.cn/problem/P2522))

对于给出的  $n$  个询问，每次求有多少个数对  $(x, y)$ ，满足  $a \leq x \leq b$ ， $c \leq y \leq d$ ，且  $\gcd(x, y) = k$ ， $\gcd(x, y)$  函数为  $x$  和  $y$  的最大公约数。

$1 \leq n, k \leq 5 \times 10^4$ ， $1 \leq a \leq b \leq 5 \times 10^4$ ， $1 \leq c \leq d \leq 5 \times 10^4$

**Solution**

**方法一：直接推导公式**

显然问题可以抽象成等式：

$$\sum_{i=a}^b \sum_{j=c}^d [\gcd(i, j) = k]$$

枚举的范围从  $a, c$  出发，不好处理，我们考虑转换成从 1 出发的形式。

根据容斥原理，原式可以分成 4 块来处理：

$ans = \text{solve}(1 \sim b, 1 \sim d) - \text{solve}(1 \sim a - 1, 1 \sim d) - \text{solve}(1 \sim b, 1 \sim c - 1) + \text{solve}(1 \sim a - 1, 1 \sim c - 1)$

(减去一个不合法的部分，加上两个都不合法的部分)

其中 solve 中要处理的式子都为：

$$\sum_{i=1}^n \sum_{j=1}^m [\gcd(i, j) = k] = \sum_{i=1}^n \sum_{j=1}^m [\gcd(\frac{i}{k}, \frac{j}{k}) = 1]$$

只有  $k$  的倍数才有贡献，而  $[1, n]$  中  $k$  的倍数显然有  $\lfloor \frac{n}{k} \rfloor$  个

所以令  $i = ik$  即仅枚举  $k$  的倍数即可

$$= \sum_{i=1}^{\lfloor \frac{n}{k} \rfloor} \sum_{j=1}^{\lfloor \frac{m}{k} \rfloor} [\gcd(i, j) = 1]$$

(所有  $k$  的倍数，除去  $k$  后互质才有贡献)

$$= \sum_{i=1}^{\lfloor \frac{n}{k} \rfloor} \sum_{j=1}^{\lfloor \frac{m}{k} \rfloor} \varepsilon(\gcd(i, j))$$

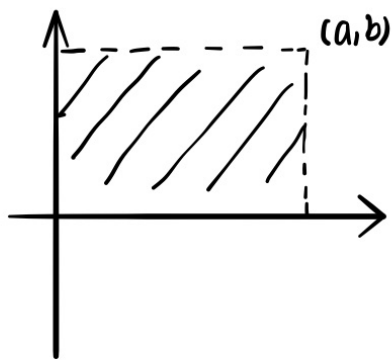
$$= \sum_{i=1}^{\lfloor \frac{n}{k} \rfloor} \sum_{j=1}^{\lfloor \frac{m}{k} \rfloor} \sum_{d|\gcd(i, j)} \mu(d)$$

$$= \sum_{d=1}^{\min\{\lfloor \frac{n}{k} \rfloor, \lfloor \frac{m}{k} \rfloor\}} \mu(d) \sum_{i=1}^{\lfloor \frac{n}{k} \rfloor} [d | i] \sum_{j=1}^{\lfloor \frac{m}{k} \rfloor} [d | j]$$

$$= \sum_{d=1}^{\min\{\lfloor \frac{n}{k} \rfloor, \lfloor \frac{m}{k} \rfloor\}} \mu(d) \left\lfloor \frac{n}{kd} \right\rfloor \left\lfloor \frac{m}{kd} \right\rfloor$$

显然可以整除分块，时间复杂度为  $O(N + T\sqrt{n})$

方法二：利用反演公式：

构造  $F(n)$  和  $f(n)$  满足  $F(n) = \sum_{d|n} f(d)$  $f(n) = \sum_{x=1}^a \sum_{y=1}^b [\gcd(x,y)=n]$  题目中的答案即在范围内  $\gcd(x,y)=n$  的点的个数 $F(n) = \sum_{x=1}^a \sum_{y=1}^b [n | \gcd(x,y)]$  在  $x \in [1,a], y \in [1,b]$  中  $\gcd(x,y)$  是  $n$  的倍数的点的个数 $f(n)$  为恰好等于  $n$  的点的个数,  $F(n)$  为  $n$  的倍数的个数故  $F(n) = \sum_{n|d} f(d)$  字面意思,  $n$  的倍数的个数, 包括 1故  $f(n) = \sum_{n|d} \mu\left(\frac{d}{n}\right) F(d)$  答案为  $f(k)$  令  $n=k$  $F(d)$  指范围内  $\gcd(x,y)=d$  的倍数的点  $(x,y)$  个数即  $d | \gcd(x,y) \Leftrightarrow d|x$  且  $d|y$ ,  $x \in [1,a]$  有  $\lfloor \frac{a}{d} \rfloor$  个  $d$ ,  $y \in [1,b]$  有  $\lfloor \frac{b}{d} \rfloor$  个  $d$  满足故  $F(d)$  共有  $\lfloor \frac{a}{d} \rfloor \cdot \lfloor \frac{b}{d} \rfloor$  个点

$$f(n) = \sum_{n|d} \mu\left(\frac{d}{n}\right) \left\lfloor \frac{a}{d} \right\rfloor \left\lfloor \frac{b}{d} \right\rfloor$$

$$= \sum_{d'} \mu(d') \left\lfloor \frac{a}{d'n} \right\rfloor \left\lfloor \frac{b}{d'n} \right\rfloor$$

$$= \sum_{d'} \mu(d') \left\lfloor \frac{a'}{d'} \right\rfloor \left\lfloor \frac{b'}{d'} \right\rfloor$$

 $d' = \frac{d}{n}$ , 表示  $d$  是  $n$  的多少倍 故  $d' = 1, 2, 3, \dots$ 

$$d = d'n$$

所有自然数

$$\text{令 } a' = \frac{a}{n}, b' = \frac{b}{n}$$

[https://blog.csdn.net/weixin\\_45697774](https://blog.csdn.net/weixin_45697774)

## Code

```

1 const int N = 500007, M = 500007, INF = 0x3f3f3f3f;
2 typedef long long ll;
3 int read()
4 {
5     int x = 0, f = 1;
6     char ch = getchar();
7     while(ch > '9' || ch < '0'){if(ch == '-')f = -1; ch = getchar();}
8     while(ch <= '9' && ch >= '0'){x = x * 10 + ch - '0'; ch = getchar();}
9     return x * f;
10 }
11 int a, b, c, d, k;
12 int mu[N];
13 int primes[N], cnt;
14 bool vis[N];
15
16 void get_mu(int n)

```

```

17 {
18     memset(vis, 0, sizeof vis);
19     memset(mu, 0, sizeof mu);
20     cnt = 0, mu[1] = 1;
21     for(int i = 2; i <= n; ++ i)
22     {
23         if(vis[i] == 0){
24             primes[ ++ cnt] = i;
25             mu[i] = -1;
26         }
27         for(int j = 1; j <= cnt && i * primes[j] <= n; ++ j){
28             vis[primes[j] * i] = 1;
29             if(i % primes[j] == 0) break;
30             mu[i * primes[j]] -= mu[i];
31         }
32     }
33     for(int i = 1; i <= n; ++ i)
34         mu[i] += mu[i - 1];
35 }
36
37 int solve(int n, int m)
38 {
39     n /= k, m /= k;
40     int res = 0;
41     for(int i = 1, j; i <= min(n, m); i = j + 1){
42         j = min(n / (n / i), m / (m / i)); //j是右边界，这里的值都是i
43         res += (mu[j] - mu[i - 1]) * (n / i) * (m / i);
44     }
45     return res;
46 }
47 int n, m;
48
49 int main()
50 {
51     get_mu(N - 1);
52     scanf("%d", &n);
53     for(int i = 1; i <= n; ++ i){
54         a = read(), b = read(), c = read(), d = read(), k = read();
55         int ans = solve(b, d) - solve(b, c - 1) - solve(a - 1, d) + solve(a - 1, c - 1);
56         printf("%d\n", ans);
57     }
58     return 0;
59 }
60
61

```

### Problem B. LCMSUM (Luogu SP5971)

计算

$$\sum_{i=1}^n \text{lcm}(i, n)$$

多组数据

$$1 \leq T \leq 3 \times 10^5, 1 \leq n \leq 10^6$$

### Solution

方法一：

$$\begin{aligned}ans &= \sum_{i=1}^n \text{lcm}(i, n) \\&= \sum_{i=1}^n \frac{i \cdot n}{\text{gcd}(i, n)} \\&= n \times \sum_{i=1}^n \frac{i}{\text{gcd}(i, n)} \\&= n \times \sum_{d|n} \sum_{i=1}^n \frac{i}{d} [\text{gcd}(i, n) = d] \\&= n \times \sum_{d|n} \sum_{i=1}^n \frac{i}{d} [\text{gcd}(\frac{i}{d}, \frac{n}{d})] \\&= n \times \sum_{d|n} \sum_{i=1}^{\lfloor \frac{n}{d} \rfloor} i \cdot [\text{gcd}(i, \frac{n}{d})] \quad (i = id)\end{aligned}$$

设  $f(n) = \sum_{i=1}^{\lfloor \frac{n}{d} \rfloor} i \cdot [\text{gcd}(i, \frac{n}{d})]$

显然它的实际含义为：  $1 \sim n$  中所有与  $n$  互质的数的和。

显然和为  $\frac{\varphi(n) \cdot n}{2}$

即：

$$f(x) = \frac{\varphi(n) \cdot n}{2}$$

即：

$$\begin{aligned}ans &= n \times \sum_{d|n} \sum_{i=1}^{\lfloor \frac{n}{d} \rfloor} i \cdot [\text{gcd}(i, \frac{n}{d})] \\&= n \times \sum_{d|n} \frac{\varphi(d) \cdot d}{2} \\&= n \times \sum_{d|n} f(d)\end{aligned}$$

显然可以先预处理出所有的  $f(n)$  , 然后利用 Dirichlet 前缀和计算  $\sum_{d|n} f(d)$  即可。

时间复杂度  $O(n \log n + T)$

**方法二：**

当然本题还有  $O(n)$  的做法。

我们重新推一遍：

$$\sum_{i=1}^n \frac{i \cdot n}{\text{gcd}(i, n)}$$

将原式复制一份并且颠倒顺序，然后将  $n$  一项单独提出，可得

$$\frac{1}{2} \cdot \left( \sum_{i=1}^{n-1} \frac{i \cdot n}{\text{gcd}(i, n)} + \sum_{i=n-1}^1 \frac{i \cdot n}{\text{gcd}(i, n)} \right) + n$$

根据  $\text{gcd}(i, n) = \text{gcd}(n - i, n)$  , 可将原式化为

$$\frac{1}{2} \cdot \left( \sum_{i=1}^{n-1} \frac{i \cdot n}{\text{gcd}(i, n)} + \sum_{i=n-1}^1 \frac{i \cdot n}{\text{gcd}(n - i, n)} \right) + n$$

两个求和式中分母相同的项可以合并。

$$\frac{1}{2} \cdot \sum_{i=1}^{n-1} \frac{n^2}{\text{gcd}(i, n)} + n$$

即

$$\frac{1}{2} \cdot \sum_{i=1}^n \frac{n^2}{\text{gcd}(i, n)} + \frac{n}{2}$$

可以将相同的  $\text{gcd}(i, n)$  合并在一起计算，故只需要统计  $\text{gcd}(i, n) = d$  的个数。当  $\text{gcd}(i, n) = d$  时，  $\text{gcd}(\frac{i}{d}, \frac{n}{d}) = 1$  , 所以  $\text{gcd}(i, n) = d$  的个数有  $\varphi(\frac{n}{d})$  个。

故答案为

$$\frac{1}{2} \cdot \sum_{d|n} \frac{n^2 \cdot \varphi(\frac{n}{d})}{d} + \frac{n}{2}$$

变换求和顺序，设  $d' = \frac{n}{d}$ ，合并公因式，式子化为

$$\frac{1}{2}n \cdot \left( \sum_{d'|n} d' \cdot \varphi(d') + 1 \right)$$

我们可以设  $g(n) = \sum_{d|n} d \times \varphi(d)$

显然函数  $g$  是一个积性函数，我们可以利用线性筛  $O(n)$  筛出  $g$ ：

1. 考虑  $g(p_j^k)$

显然它的约数只有  $p_j^0, p_j^1, \dots, p_j^k$ ，因此

$$g(p_j^k) = \sum_{c=0}^k p_j^c \times \varphi(p_j^c)$$

又有  $\varphi(p_j^c) = p_j^{c-1} \cdot (p_j - 1)$ ，则原式可化为

$$\sum_{c=0}^k p_j^{2c-1} \times (p_j - 1)$$

于是有

$$g(p_j^{k+1}) = g(p_j^k) + p_j^{2k+1} \times (p_j - 1)$$

2. 考虑  $g(i \times p_j)$ ，当  $p_j \mid i$

令  $i = a \times p_j^c (\gcd(a, p_j) = 1)$ ，可得

$$g(i \times p_j) = g(a) \times g(p_j^{c+1})$$

$$g(i) = g(a) \times g(p_j^c)$$

显然有：

$$\begin{aligned} g(i \times p_j) - g(i) &= g(a) \times g(p_j^{c+1}) - g(a) \times g(p_j^c) \\ &= g(a) \times (g(p_j^c) + p_j^{2c+1} \times (p_j - 1)) - g(a) \times g(p_j^c) \\ &= g(a) \times p_j^{2c+1} \times (p_j - 1) \end{aligned}$$

同理有

$$g(i) - g(\frac{i}{p_j}) = g(a) \times p_j^{2c-1} \times (p_j - 1)$$

因此

$$g(i \times p_j) = g(i) + \left( g(i) - g(\frac{i}{p_j}) \right) \times p_j^2$$

3. 考虑  $g(i \times p_j)$ ，当  $p_j \nmid i$

积性函数，显然  $g(i \times p_j) = g(i) \times g(p_j)$

时间复杂度  $O(n + T)$

Code1

$O(n \log n + T)$

```
1 typedef long long ll;
2 typedef pair<int , int> PII;
3 const int N = 5000007, M = 500007, T = 1000, Mod = 998244353, INF = 0x3f3f3f3f;
4
5 int phi[N];
6 bool vis[N];
7 ll f[N], g[N];
8 int primes[N], cnt;
9
10 void get_phi(int n)
11 {
12     phi[1] = 1;
13     f[1] = 1;
```

```

14     for(int i = 2; i <= n; ++ i) {
15         if(vis[i] == 0) {
16             primes[ ++ cnt] = i;
17             phi[i] = i - 1;
18         }
19         f[i] = 1ll * phi[i] * i / 2;
20         for(int j = 1; j <= cnt && i * primes[j] <= n; ++ j) {
21             vis[i * primes[j]] = true;
22             if(i % primes[j] == 0) {
23                 phi[i * primes[j]] = phi[i] * primes[j];
24                 break;
25             }
26             else phi[i * primes[j]] = phi[i] * (primes[j] - 1);
27         }
28     }
29     for(int j = 1; j <= cnt; ++ j) {
30         for(int i = 1; i * primes[j] <= n; ++ i) {
31             f[i * primes[j]] += f[i];
32         }
33     }
34 }
35
36 int n, t;
37
38 void solve()
39 {
40     scanf("%d", &n);
41     printf("%lld\n", f[n] * n);
42 }
43
44 int main()
45 {
46     scanf("%d", &t);
47     get_phi(N - 1);
48     while(t -- ) {
49         solve();
50     }
51 }

```

## Code2

$O(n + T)$

```

1  using ll = long long;
2  const int N = 1e6 + 7;
3
4  int n, m, s, t, k, ans, a[N];
5  int primes[N], cnt;
6  ll g[N];
7  bool vis[N];
8
9  void pre_work(int n)
10 {
11     vis[1] = 1;
12     g[1] = 1;
13     for (int i = 2; i <= n; ++ i) {
14         if(vis[i] == 0) {
15             primes[ ++ cnt] = i;
16             g[i] = 1ll * i * (i - 1) + 1;
17         }
18         for (int j = 1; j <= cnt && i * primes[j] <= n; ++ j) {
19             vis[i * primes[j]] = 1;
20             if(i % primes[j] == 0) {
21                 g[i * primes[j]] = g[i] + (g[i] - g[i / primes[j]]) * primes[j] * primes[j];
22                 break;
23             }
24             g[i * primes[j]] = g[i] * g[primes[j]];
25         }
26     }
27 }
28
29 void solve()

```



```
30 {
31     scanf("%d", &n);
32     printf("%lld\n", (g[n] + 1) * n / 211);
33 }
34
35 int main()
36 {
37     pre_work(N - 7);
38     scanf("%d", &t);
39     while(t -- ) {
40         solve();
41     }
42     return 0;
43 }
```

**Problem C. 约数个数和 (P3327 [SDOI2015])**

设  $d(x)$  为  $x$  的约数个数，给定  $N, M$ ，求

$$\sum_{i=1}^N \sum_{j=1}^M d(ij)$$

**Solution**

# SDOI 2015 最大约数和

设  $d(x)$  为  $x$  的约数个数, 给定  $N, M$ , 求  $\sum_{i=1}^N \sum_{j=1}^M d(i \cdot j)$

1和1互质 ( $\gcd(1,1)=1$ )

$$\text{公式: } d(i \cdot j) = \sum_{x|i} \sum_{y|j} [\gcd(x,y)=1]$$

证明: 设  $i = p_1^{d_1} \cdot p_2^{d_2} \cdot p_3^{d_3} \cdots p_k^{d_k}$

$$j = p_1^{\beta_1} \cdot p_2^{\beta_2} \cdot p_3^{\beta_3} \cdots p_k^{\beta_k} \quad (\beta_i \text{ 可以为 } 0 \text{ 嘛})$$

$$i \cdot j = p_1^{d_1+\beta_1} \cdot p_2^{d_2+\beta_2} \cdots p_k^{d_k+\beta_k}$$

对于  $p_1$ :

仅  $p_1$  约数个数

$$(d_1 + \beta_1 + 1)$$

$$p_2 = (d_2 + \beta_2 + 1)$$

$$\vdots$$

$$\vdots$$

$$\vdots$$

$$\vdots$$

$$\vdots$$

$$\vdots$$

$$\vdots$$

$$\vdots$$

$$\vdots$$

$$\vdots$$

$$\vdots$$

$$\vdots$$

$$\vdots$$

$$\vdots$$

$$\vdots$$

$$\vdots$$

$$\vdots$$

$$\vdots$$

$$\vdots$$

$$\vdots$$

$$\vdots$$

$$\vdots$$

$$\vdots$$

$$\vdots$$

$$\vdots$$

$$\vdots$$

$$\vdots$$

$$\vdots$$

$$\vdots$$

$$\vdots$$

$$\vdots$$

$$\vdots$$

$$\vdots$$

$$\vdots$$

$$\vdots$$

$$\vdots$$

$$\vdots$$

$$\vdots$$

$$\vdots$$

$$\vdots$$

$$\vdots$$

$i \cdot j$  约数个数:  $(d_1 + \beta_1 + 1)(d_2 + \beta_2 + 1) \cdots (d_k + \beta_k + 1)$  即为总约数个数

即  $i$  和  $j$  的所有因子  $x, y$ , 互质则答案+1

$$\text{即为 } d(i \cdot j) = \sum_{x|i} \sum_{y|j} [\gcd(x,y)=1]$$

$$\sum_{i=1}^N \sum_{j=1}^M d(i \cdot j) = \sum_{i=1}^N \sum_{j=1}^M \sum_{x|i} \sum_{y|j} [\gcd(x,y)=1]$$

$$f(n) = \sum_{i=1}^N \sum_{j=1}^M \sum_{x|i} \sum_{y|j} [\gcd(x,y)=n] \quad (\text{答案为 } f(1))$$

$$F(n) = \sum_{i=1}^N \sum_{j=1}^M \sum_{x|i} \sum_{y|j} [n | \gcd(x,y)] \quad (\text{经典})$$

$f(n)$  为恰好等于  $n$  的点的个数,  $F(n)$  为  $n$  的倍数的个数

$$\text{故 } F(n) = \sum_{n|d} f(d) \quad \text{字面意思, } n \text{ 的倍数的个数, 包括 } 1$$

$$\text{故 } f(n) = \sum_{n|d} \mu\left(\frac{d}{n}\right) F(d) \quad \text{答案为 } f(1) \text{ 令 } n=1$$

$$F(n) = \sum_{i=1}^N \sum_{j=1}^M \sum_{x|i} \sum_{y|j} [n | \gcd(x,y)]$$

## 开始推柿子

$$F(n) = \sum_{i=1}^N \sum_{j=1}^M \sum_{x|i} \sum_{y|j} [n | \gcd(x, y)]$$

$$= \sum_{x=1}^N \sum_{y=1}^M \left\lfloor \frac{N}{x} \right\rfloor \left\lfloor \frac{M}{y} \right\rfloor [n | \gcd(x, y)] \quad (1)$$

$$= \sum_{x'=1}^{\frac{N}{n}} \sum_{y'=1}^{\frac{M}{n}} \left\lfloor \frac{N}{nx'} \right\rfloor \left\lfloor \frac{M}{ny'} \right\rfloor \quad (2)$$

$$= \sum_{i=1}^{N'} \sum_{j=1}^{M'} \left\lfloor \frac{N'}{i} \right\rfloor \left\lfloor \frac{M'}{j} \right\rfloor \quad (3)$$

$$= \left( \sum_{i=1}^{N'} \left\lfloor \frac{N'}{i} \right\rfloor \right) \left( \sum_{j=1}^{M'} \left\lfloor \frac{M'}{j} \right\rfloor \right)$$

$$= H(N') H(M') \quad (\text{别忘了 } N' = \frac{N}{n}, M' = \frac{M}{n})$$

其中  $H(k)$  为调和数

$$H(k) = \sum_{i=1}^k \frac{1}{i} \quad \text{预处理 } H(n) \text{ 即可 } (n \leq 50000) \quad \text{整除分块 } O(\sqrt{n})$$

则答案为:  $f(n) = \sum_{n|d} \mu(d) F(d) \quad (n=1, n|d \text{ 则 } d \text{ 为自然数集})$

$$\begin{aligned} N = \min\{n, m\} &= \sum_{d=1}^N \mu(d) F(d) \\ &= \sum_{d=1}^N \mu(d) H\left(\left\lfloor \frac{N}{d} \right\rfloor\right) \cdot H\left(\left\lfloor \frac{M}{d} \right\rfloor\right) \quad \text{整除分块 } O(\sqrt{n}) \end{aligned}$$

我们发现  $H\left(\left\lfloor \frac{N}{d} \right\rfloor\right), H\left(\left\lfloor \frac{M}{d} \right\rfloor\right)$  中,

$H(x)$  仅与变量  $x$  有关

故可以整除分块



$$\frac{k}{l} = \frac{k}{y}$$

$l \sim r$  间  $\frac{k}{x}$  均相同, 故取  $\frac{k}{l}$  即可

$$y = k / (k / l)$$

总时间复杂度  $O(T\sqrt{n})$   $T$  是询问组数

## Code

```
1 const int N = 50007;
2 typedef long long ll;
3 int n, m, t;
4 int primes[N];
5 ll res;
6 bool vis[N];
7 int mu[N], sum[N], cnt, H[N];
8 int g(int k, int x) {return k / (k / x);}
9 void init(int n)
10 {
```

```

11     mu[1] = 1;
12     for(int i = 2; i <= n; ++ i) {
13         if(vis[i] == 0) {
14             primes[ ++ cnt] = i;
15             mu[i] = -1;
16         }
17         for(int j = 1; j <= cnt && primes[j] * i <= n; ++ j) {
18             vis[i * primes[j]] = true;
19             if(i % primes[j] == 0) break;
20             mu[i * primes[j]] = -mu[i];
21         }
22     }
23     for(int i = 1; i <= n; ++ i) sum[i] = sum[i - 1] + mu[i];
24     for(int i = 1; i <= n; ++ i) {
25         for(int l = 1, r; l <= i; l = r + 1) {
26             r = min(i, g(i, l));
27             H[i] += (r - l + 1) * (i / l);
28         }
29     }
30 }
31
32 int main()
33 {
34     cin >> t;
35     init(N - 1);
36     while(t -- ) {
37         res = 0;
38         scanf("%d%d", &n, &m);
39         int k = min(n, m);
40         for(int l = 1, r; l <= k; l = r + 1) {
41             r = min(k, min(g(n, l), g(m, l)));
42             res += ((ll)sum[r] - sum[l - 1]) * H[n / l] * H[m / l];
43         }
44         printf("%lld\n", res);
45     }
46     return 0;
47 }

```

#### Problem D.Crash 的数字表格 (Luogu P1829)

计算 (对 20101009 取模)

$$\sum_{i=1}^n \sum_{j=1}^m \text{lcm}(i, j) \quad (n, m \leqslant 10^7)$$



## Solution

计算  $\sum_{i=1}^n \sum_{j=1}^m \text{lcm}(i, j)$  设  $n \leq m$

$$\sum_{i=1}^n \sum_{j=1}^m \text{lcm}(i, j)$$

$$= \sum_{i=1}^n \sum_{j=1}^m \frac{i \times j}{\gcd(i, j)}$$

$$= \sum_{i=1}^n \sum_{j=1}^m \frac{i \times j}{d} [\gcd(i, j) = d] [d|i] [d|j]$$

$$= \sum_{i=1}^n \sum_{j=1}^m \sum_{d|i, d|j} [\gcd(\frac{i}{d}, \frac{j}{d}) = 1] \frac{i \times j}{d}$$

$$= \sum_{d=1}^n \sum_{i=1}^{\lfloor \frac{n}{d} \rfloor} \sum_{j=1}^{\lfloor \frac{m}{d} \rfloor} [\gcd(i, j) = 1] \frac{i \times d \times j \times d}{d}$$

$$= \sum_{d=1}^n d \cdot \sum_{i=1}^{\lfloor \frac{n}{d} \rfloor} \sum_{j=1}^{\lfloor \frac{m}{d} \rfloor} [\gcd(i, j) = 1] i \times j$$

$$= \sum_{d=1}^n d \sum_{i=1}^{\lfloor \frac{n}{d} \rfloor} \sum_{j=1}^{\lfloor \frac{m}{d} \rfloor} \sum_{k|\gcd(i, j)} \mu(k) i \cdot j \quad \varepsilon = \mu * 1$$

$$= \sum_{d=1}^n d \sum_{k=1}^{\lfloor \frac{n}{d} \rfloor} \mu(k) \sum_{i=1}^{\lfloor \frac{n}{kd} \rfloor} i [k|i] \sum_{j=1}^{\lfloor \frac{m}{kd} \rfloor} j [k|j]$$

$$= \sum_{d=1}^n d \sum_{k=1}^{\lfloor \frac{n}{d} \rfloor} \mu(k) \sum_{i=1}^{\lfloor \frac{n}{kd} \rfloor} ik \sum_{j=1}^{\lfloor \frac{m}{kd} \rfloor} jk$$

$$= \sum_{d=1}^n d \sum_{k=1}^{\lfloor \frac{n}{d} \rfloor} k^2 \mu(k) \sum_{i=1}^{\lfloor \frac{n}{kd} \rfloor} i \sum_{j=1}^{\lfloor \frac{m}{kd} \rfloor} j$$

$$= \sum_{d=1}^n d \sum_{k=1}^{\lfloor \frac{n}{d} \rfloor} k^2 \mu(k) \frac{\lfloor \frac{n}{kd} \rfloor (\lfloor \frac{n}{kd} \rfloor + 1)}{2} \cdot \frac{\lfloor \frac{m}{kd} \rfloor (\lfloor \frac{m}{kd} \rfloor + 1)}{2}$$

将  $d, k$  联系起来, 设  $T = dk$

$$\text{原式} = \sum_{T=1}^n \frac{\lfloor \frac{n}{T} \rfloor (\lfloor \frac{n}{T} \rfloor + 1)}{2} \cdot \frac{\lfloor \frac{m}{T} \rfloor (\lfloor \frac{m}{T} \rfloor + 1)}{2} \sum_{k|T} T \cdot k \mu(k)$$

设  $f(T) = \sum_{k|T} k \mu(k)$  可用 Dirichlet 前缀和预处理  $O(n \log \log n)$

多组数据需要优化至  $O(n)$

$$\text{设 } S(T) = \frac{T(T+1)}{2} \quad g(T) = T \times f(T)$$

显然  $f(T)$  是积性函数

$$\text{原式} = \sum_{T=1}^n S(\frac{n}{T}) \times S(\frac{m}{T}) \times g(T)$$

则有  $T = \prod_{i=1}^t p_i^{d_i}$

数论分块  $S$ ,  $O(n)$  预处理  $g$ , 时间复杂度  $O(\max\{n, T\sqrt{n}\})$

$$f(T) = f(\prod_{i=1}^t p_i^{d_i})$$

$$= \prod_{i=1}^t f(p_i^{d_i})$$

$$= \prod_{i=1}^t (p_i^0 \times \mu(1) + p_i^1 \times \mu(p_i) + p_i^2 \times \mu(p_i^2) + p_i^3 \times \mu(p_i^3) + \dots)$$

$$= \prod_{i=1}^t (1 - 1 \times p_i) \quad (\mu(p_i^2) = 0)$$

$$= \prod_{i=1}^t (1 - p_i)$$

则对于线性筛  
三种情况:

$$\textcircled{1} \text{ if } (vis == 0) \quad f(p) = \sum_{k|p} p \times \mu(k) = 1 - (-1) \times p = 1 - p$$

$$\textcircled{2} \text{ if } (i \% p == 0) \quad f(p^k) = f(p^{k-1}) = f(p)$$

$$\textcircled{3} \text{ else } (i, p \text{ 互质}) \quad f(i \times p) = f(i) \times f(p)$$

https://blog.csdn.net/weixin\_45697774

## Code

```
1 #define int long long
2 const int N = 10000007, mod = 20101009;
3
4 int n, m;
5 int primes[N], cnt, f[N], g[N];
6 bool vis[N];
7
8 void init(int n)
9 {
```

```

10  g[1] = f[1] = 1;
11  for(int i = 2; i <= n; ++ i) {
12      if(vis[i] == 0) {
13          primes[ ++ cnt] = i;
14          f[i] = (1 - i % mod + mod) % mod;
15      }
16      for(int j = 1; j <= cnt && i * primes[j] <= n; ++ j) {
17          vis[i * primes[j]] = true;
18          if(i % primes[j] == 0) {
19              f[i * primes[j]] = f[i];
20              break;
21          }
22          f[i * primes[j]] = (f[i] * f[primes[j]]) % mod;
23      }
24  }
25  for(int i = 1; i <= n; ++ i) {
26      g[i] = (f[i] * i % mod + g[i - 1]) % mod;
27  }
28 }
29
30 int s(int x)
31 {
32     return (x * (x + 1) / 2) % mod;
33 }
34
35 int ans;
36
37 signed main()
38 {
39     init(N - 7);
40     scanf("%lld%lld", &n, &m);
41     if(n > m) swap(n, m);
42     ans = 0;
43     for(int l = 1, r; l <= n; l = r + 1) {
44         r = min(n / (n / l), m / (m / l));
45         ans = (ans + (s(n / l) * s(m / l) % mod) * (g[r] - g[l - 1] + mod) + mod % mod) % mod;
46     }
47     printf("%lld\n", ans);
48     return 0;
49 }

```

### Problem E. 简单的数学题 (luogu P3768)

#### 前置知识：0x52 杜教筛

输入一个整数  $n$  和一个整数  $p$ , 你需要求出:

$$\left( \sum_{i=1}^n \sum_{j=1}^n ij \gcd(i, j) \right) \bmod p$$

其中  $\gcd(a, b)$  表示  $a$  与  $b$  的最大公约数。

$n \leq 10^{10}$ ,  $5 \times 10^8 \leq p \leq 1.1 \times 10^9$  且  $p$  为质数。

#### Solution



$$\begin{aligned}
 & \text{计算: } \left( \sum_{i=1}^n \sum_{j=1}^n i \cdot j \cdot \gcd(i, j) \right) \bmod p \\
 &= \sum_{i=1}^n \sum_{j=1}^n i \cdot j \cdot \gcd(i, j) \\
 &= \sum_{d=1}^n d \sum_{i=1}^{\frac{n}{d}} \sum_{j=1}^{\frac{n}{d}} i \cdot j \cdot [\gcd(i, j) = 1] \\
 &= \sum_{d=1}^n d \sum_{i=1}^{\frac{n}{d}} \sum_{j=1}^{\frac{n}{d}} i \cdot j \cdot [\gcd(\frac{i}{d}, \frac{j}{d}) = 1] \quad \left( i=id, j=jd \text{ (} d=\gcd(i, j) \text{ 是 } i, j \text{ 的倍数)} \right) \\
 &= \sum_{d=1}^n d \sum_{i=1}^{\frac{n}{d}} \sum_{j=1}^{\frac{n}{d}} i \cdot d \cdot j \cdot d \cdot [\gcd(i, j) = 1] \\
 &= \sum_{d=1}^n d^3 \sum_{i=1}^{\frac{n}{d}} \sum_{j=1}^{\frac{n}{d}} i \cdot j \cdot [\gcd(i, j) = 1] \\
 &= \sum_{d=1}^n d^3 \sum_{i=1}^{\frac{n}{d}} \sum_{j=1}^{\frac{n}{d}} i \cdot j \sum_{k|\gcd(i, j)} \mu(k) \\
 &= \sum_{d=1}^n d^3 \sum_{i=1}^{\frac{n}{d}} \sum_{j=1}^{\frac{n}{d}} i \cdot j \sum_{k=1}^{\frac{n}{d}} \mu(k) [k|i] [k|j] \\
 &= \sum_{d=1}^n d^3 \sum_{k=1}^{\frac{n}{d}} \mu(k) \sum_{i=1}^{\frac{n}{dk}} \sum_{j=1}^{\frac{n}{dk}} i \cdot j [k|i] [k|j] \\
 &= \sum_{d=1}^n d^3 \sum_{k=1}^{\frac{n}{d}} \mu(k) \sum_{i=1}^{\frac{n}{dk}} \sum_{j=1}^{\frac{n}{dk}} i \cdot k \cdot j \cdot k \quad \left( i=ik, j=jk \text{ (} i, j \text{ 是 } k \text{ 的倍数)} \right) \\
 &= \sum_{d=1}^n d^3 \sum_{k=1}^{\frac{n}{d}} \mu(k) k^2 \sum_{i=1}^{\frac{n}{dk}} \sum_{j=1}^{\frac{n}{dk}} i \cdot j \\
 &= \sum_{d=1}^n d^3 \sum_{k=1}^{\frac{n}{d}} \mu(k) k^2 \left( \sum_{i=1}^{\frac{n}{dk}} i \right)^2
 \end{aligned}$$

设  $T=kd$

$$\begin{aligned}
 &= \sum_{T=1}^n T^2 \sum_{d|T} \mu\left(\frac{T}{d}\right) d \left( \sum_{i=1}^{\frac{n}{T}} i \right)^2 \\
 &= \sum_{T=1}^n T^2 \varphi\left(\frac{T}{d}\right) \left( \sum_{i=1}^{\frac{n}{T}} i \right)^2 \\
 &= \sum_{T=1}^n T^2 \varphi(T) \left( \sum_{i=1}^{\frac{n}{T}} i \right)^2 \\
 &= \sum_{T=1}^n \varphi(T) T^2 \sum_{i=1}^{\frac{n}{T}} i^2
 \end{aligned}$$

$n \leq 10^6$  考虑杜教筛

$$\text{公式: } g(1) S(n) = \sum_{i=1}^n h(i) - \sum_{i=2}^n g(i) S\left(\left\lfloor \frac{n}{i} \right\rfloor\right)$$

$$\begin{aligned}
 & \text{设 } f(x) = x^2 \varphi(x) \quad S(n) = \sum_{i=1}^n f(i) \\
 & h(i) = f * g(i)
 \end{aligned}$$

$$= \sum_{d|i} f(d) g\left(\frac{i}{d}\right)$$

$$= \sum_{d|i} d^2 \varphi(d) \times g\left(\frac{i}{d}\right)$$

$$\text{显然 } \sum_{d|i} \varphi(d) = i \quad (\varphi * 1 = id)$$

$$\text{故设 } g(x) = i^2 \quad \text{则: } h(i) = \sum_{d|i} d^2 \varphi(d) \cdot \frac{i^2}{d^2} = \sum_{d|i} \varphi(d) i^2 = i^3$$

故根据杜教筛公式得:

$$S(n) = \frac{\sum_{i=1}^n h(i) - \sum_{i=2}^n g(i) S\left(\left\lfloor \frac{n}{i} \right\rfloor\right)}{g(1)}$$

$$= \sum_{i=1}^n i^3 - \sum_{i=2}^n i^2 S\left(\left\lfloor \frac{n}{i} \right\rfloor\right)$$

数论分块 ( $i^2$  的区间前缀和)

$$\text{显然 } \sum_{i=1}^n i^3 = 1^3 + 2^3 + \dots + n^3 = (1+2+3+\dots+n)^2 = \frac{n^2(n+1)^2}{4}$$

$$\sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6}$$

Hint

注意一点: 这种需要输入模数mod的题, 一定要等输入模数之后再初始化 init ...不然会RE...膜0可不就RE了嘛...

Time

$$O(n^{\frac{2}{3}})$$

Code

```

1 #define int long long
2 typedef long long ll;
3 const int N = 4641589; // [(10^10)^(2/3)]

```

```

4
5 int primes[N + 7], cnt;
6 ll phi[N + 7];
7 bool vis[N + 7];
8 unordered_map<ll, ll> sum_f;
9 int mod, n;
10 int inv4, inv6;
11
12 int qpow(int a, int b)
13 {
14     int res = 1;
15     while(b) {
16         if(b & 1) res = 1ll * res * a % mod;
17         a = 1ll * a * a % mod;
18         b >>= 1;
19     }
20     return res % mod;
21 }
22
23 void init(int n)
24 {
25     phi[1] = 1;
26     for(int i = 2; i <= n; ++ i) {
27         if(vis[i] == 0) {
28             primes[ ++ cnt] = i;
29             phi[i] = i - 1;
30         }
31         for(int j = 1; j <= cnt && i * primes[j] <= n; ++ j) {
32             vis[i * primes[j]] = true;
33             if(i % primes[j] == 0) {
34                 phi[i * primes[j]] = 1ll * phi[i] * primes[j] % mod;
35                 break;
36             }
37             phi[i * primes[j]] = 1ll * phi[i] * phi[primes[j]] % mod;
38         }
39     }
40     for(int i = 1; i <= n; ++ i) {
41         phi[i] = (1ll * phi[i - 1] + (1ll * phi[i] * i % mod * i % mod)) % mod;
42     }
43 }
44
45
46
47 inline int g_sum(int n) //i^2
48 {
49     n %= mod;
50     return 1ll * n * inv6 % mod * (n + 1) % mod * (2 * n + 1) % mod;
51 }
52
53 inline int h_sum(int n) //i^3
54 {
55     n %= mod;
56     return 1ll * n * inv4 % mod * n % mod * (n + 1) % mod * (n + 1) % mod;
57 }
58
59 inline int get_s_sum(int x)
60 {
61     if(x <= N - 7) return phi[x];
62     if(sum_f.find(x) != sum_f.end()) return sum_f[x];
63
64     ll ans = h_sum(x);
65     for(ll l = 2, r; l <= x; l = r + 1) {
66         r = x / (x / l);
67         ans = (ans - 1ll * (g_sum(r) - g_sum(l - 1) + mod) % mod * get_s_sum(x / l) % mod) % mod ;
68     }
69     return sum_f[x] = ans / g_sum(1);
70 }
71
72 void solve(int n)
73 {
74     ll ans = 0;

```

```
75     for(int l = 1, r; l <= n; l = r + 1) {
76         r = n / (n / l);
77         ans = (ans + (111 * h_sum(n / l) * (get_s_sum(r) - get_s_sum(l - 1) + mod) % mod)) % mod;
78     }
79     printf("%lld\n", ans);
80     return ;
81 }
82
83 signed main()
84 {
85
86     scanf("%lld%lld", &mod, &n);
87     init(N - 7);
88     inv4 = qpow(4, mod - 2);
89     inv6 = qpow(6, mod - 2);
90     solve(n);
91     //cout << "ok" << endl;
92     return 0;
93 }
```

趣味题目：

# Problem F. Gcd Product (2020 ICPC 济南 F)

Give you  $n, A_{1..n}, B_{1..n}$ , you need to calculate:

$$C_k = \sum_{i=1}^k A_{gcd(i,k)} B_{gcd(k+1-i,k)}$$

Because the output may be too large, let  $Ans_i$  denote  $C_i \bmod 998244353$ , you only need to output

$Ans_1 \text{ xor } Ans_2 \text{ xor } \dots \text{ xor } Ans_n$

## Solution

$$\begin{aligned} C_k &= \sum_{i=1}^k A_{gcd(i,k)} B_{gcd(k+1-i,k)} \\ &= \sum_{d_1|k} A_{d_1} \sum_{d_2|k} B_{d_2} \sum_{i=1}^k [gcd(i,k) = d_1] [gcd(k+1-i,k) = d_2] \\ &= \sum_{d_1|k} A_{d_1} \sum_{d_2|k} B_{d_2} \sum_{i=1}^k [gcd(\frac{i}{d_1}, \frac{k}{d_1}) = 1] [gcd(\frac{k+1-i}{d_2}, \frac{k}{d_2}) = 1] [d_1|i] [d_2|k+1-i] \\ &= \sum_{d_1|k} A_{d_1} \sum_{d_2|k} B_{d_2} \sum_{t_1|gcd(\frac{i}{d_1}, \frac{k}{d_1})} \mu(t_1) \sum_{t_2|gcd(\frac{k+1-i}{d_2}, \frac{k}{d_2})} \mu(t_2) \sum_{i=1}^k [d_1|i] [d_2|k+1-i] \quad \varepsilon = \mu * 1 \rightarrow \sum_{t_1|gcd(\frac{i}{d_1}, \frac{k}{d_1})} \mu(t_1) \\ &= \sum_{d_1|k} A_{d_1} \sum_{d_2|k} B_{d_2} \sum_{t_1|\frac{k}{d_1}} \mu(t_1) \sum_{t_2|\frac{k}{d_2}} \mu(t_2) \sum_{i=1}^k [d_1|i] [t_1|\frac{i}{d_1}] [d_2|k+1-i] [t_2|\frac{k+1-i}{d_2}] \end{aligned}$$

把  $t_1$  和  $d_1$  联系起来:  $d_1|k, t_1|\frac{k}{d_1} \rightarrow T_1 = d_1 \times t_1 \rightarrow d_1|T_1, t_1 = \frac{T_1}{d_1}, T_1|k$  可得  $t_1, d_1$  合并  
 $t_2, d_2$  同理  $T_2 = d_2 \times t_2 \rightarrow t_2 = \frac{T_2}{d_2}$

$$\text{则原式} = \sum_{T_1|k} \sum_{d_1|T_1} A_{d_1} \mu(\frac{T_1}{d_1}) \sum_{T_2|k} \sum_{d_2|T_2} B_{d_2} \mu(\frac{T_2}{d_2}) \sum_{i=1}^k [T_1|i] [T_2|k+1-i]$$

$$\text{设 } f(T_1) = \sum_{d_1|T_1} A_{d_1} \mu(\frac{T_1}{d_1}) \quad g(T_2) = \sum_{d_2|T_2} B_{d_2} \mu(\frac{T_2}{d_2})$$

$$\therefore C_k = \sum_{T_1|k} f(T_1) \sum_{T_2|k} g(T_2) \sum_{i=1}^k [T_1|i] [T_2|k+1-i]$$

前面  $f, g$  卷积  $O(n \log n)$  即可

后面  $\sum_{i=1}^k [T_1|i] [T_2|k+1-i]$

整除问题  $\rightarrow$  经典列方程 设  $i = k_1 T_1, k+1-i = k_2 T_2$  且  $k_1|i, k_2|k+1-i, T_1|k, T_2|k$

$$\text{则 } k_1 T_1 + k_2 T_2 = i + k + 1 - i = k + 1$$

$$k_1 T_1 + k_2 T_2 = k + 1$$

$$\therefore T_1|k, T_2|k$$

$$\therefore (k_1 T_1 + k_2 T_2) \% T_1 = (k+1) \% T_1 \Rightarrow k_2 T_2 \equiv 1 \pmod{T_1}$$

$$(k_1 T_1 + k_2 T_2) \% T_2 = (k+1) \% T_2 \Rightarrow k_1 T_1 \equiv 1 \pmod{T_2}$$

$$\text{设 } gcd(T_1, T_2) = d. \text{ 则 } d|i, d|k+1-i \Rightarrow d|i+k+1-i \Rightarrow d|k+1$$

$$\text{而 } T_1|k, T_2|k \Rightarrow gcd(T_1, T_2) = d|k \Rightarrow d|k, \text{ 而 } d|k+1 \Rightarrow d=1 \Rightarrow T_1, T_2 \text{ 互质}$$

$$T_1, T_2 \text{ 互质} \Rightarrow \text{存在一组解 } (x_1, x_2) \begin{cases} k_2 T_2 \equiv 1 \pmod{T_1} \quad \xrightarrow{T_1, T_2 \text{ 互质所以}} T_1 T_2 \text{ 互质所以} \\ k_1 T_1 \equiv 1 \pmod{T_2} \quad \xrightarrow{\text{只是 } k \% T_1 \equiv 1} \end{cases} \begin{cases} k_2 \equiv 1 \pmod{T_1} \Rightarrow k_2 = \gamma_2 T_1 + a_2 + 1 = \gamma_2 T_1 + x_2 \\ k_1 \equiv 1 \pmod{T_2} \Rightarrow k_1 = \gamma_1 T_2 + a_1 + 1 = \gamma_1 T_2 + x_1 \end{cases}$$

$$\therefore k_1 T_1 + k_2 T_2 = k + 1 \Rightarrow x_1 T_1 + x_2 T_2 = k + 1 - (\gamma_1 + \gamma_2) T_1 T_2$$

$$\therefore T_1|k_1, T_2|k_2, gcd(T_1, T_2) = 1 \Rightarrow T_1 T_2|k$$

$$\therefore (x_1 T_1 + x_2 T_2) \bmod (T_1 T_2) = (k + 1 - (\gamma_1 + \gamma_2) T_1 T_2) \bmod T_1 T_2$$

$$x_1 T_1 + x_2 T_2 \equiv 1 \pmod{T_1 T_2}$$

$$\therefore 1 \leq x_1 < T_2, 1 \leq x_2 < T_1 \Rightarrow 1 < x_1 T_1 + x_2 T_2 < 2 T_1 T_2$$

$$\therefore x_1 T_1 + x_2 T_2 = T_1 T_2 + 1 \xrightarrow{\text{代入}} x_1 T_1 + x_2 T_2 = k + 1 - (\gamma_1 + \gamma_2) T_1 T_2 \Rightarrow \gamma_1 + \gamma_2 = \frac{k}{T_1 T_2} - 1$$

$$\begin{aligned} 0 \leq a_1 < T_2 \quad x_1 = a_1 + 1 \quad 1 \leq x_1 < T_2 \\ 0 \leq a_2 < T_1 \quad x_2 = a_2 + 1 \quad 1 \leq x_2 < T_1 \end{aligned}$$



显然对于  $x_1 + x_2 = n$ , 共有  $n+1$  组非负整数解

故  $yr_1 + r_2 = \frac{k}{T_1 T_2} - 1$ ,  $r_1, r_2$  共有  $\frac{k}{T_1 T_2} - 1 + 1 = \frac{k}{T_1 T_2}$  组非负整数解.

而  $x_1, x_2$  仅有一组解

$k_2 = r_2 T_1 + x_1 \Rightarrow k_1, k_2$  共有  $\frac{k}{T_1 T_2}$  组解  $\xrightarrow{T_1 T_2 \text{ 约值}}$   $k_1 T_1, k_2 T_2$  共有  $\frac{k}{T_1 T_2}$  组解  
 $k_1 = r_1 T_2 + x_2$

$$\begin{aligned} \text{故 } & \sum_{i=1}^k [T_1 | i] [T_2 | k+1-i] \\ &= \frac{k}{T_1 T_2} [gcd(T_1, T_2) = 1] \end{aligned}$$

合并:  $T = T_1 T_2$

$$\text{故 } C_k = \sum_{T|k} \frac{k}{T} \sum_{T_1|T} f(T_1) g\left(\frac{T}{T_1}\right) [gcd(T_1, \frac{T}{T_1}) = 1]$$

$$\text{令 } h(T) = \sum_{T_1|T} f(T_1) g\left(\frac{T}{T_1}\right) [gcd(T_1, \frac{T}{T_1}) = 1]$$

$$\text{则 } C_k = \sum_{T|k} h(T) \frac{k}{T}$$

[https://blog.csdn.net/weixin\\_45897774](https://blog.csdn.net/weixin_45897774)

## Code

```
1 #define int long long
2 const int N = 5e5 + 7, mod = 998244353;
3
4 int n, m;
5 int a[N], b[N], c[N];
6 int primes[N], cnt, mu[N], phi[N];
7 bool vis[N];
8 int f[N], g[N], h[N];
9
10 void init(int n)
11 {
12     mu[1] = phi[1] = 1;
13     for(int i = 2; i <= n; ++ i) {
14         if(vis[i] == 0) {
15             primes[ ++ cnt] = i;
16             phi[i] = i - 1;
17             mu[i] = -1;
18         }
19         for(int j = 1; j <= cnt && i * primes[j] <= n; ++ j) {
20             vis[i * primes[j]] = true;
21             if(i % primes[j] == 0) {
22                 phi[i * primes[j]] = phi[i] * primes[j];
23                 break;
24             }
25             phi[i * primes[j]] = phi[i] * (primes[j] - 1);
26             mu[i * primes[j]] = -mu[i];
27         }
28     }
29 }
30
31 int ans = 0;
32
33 signed main()
34 {
35
36     init(N - 7);
37     scanf("%lld", &n);
38     for(int i = 1; i <= n; ++ i) {
39         scanf("%lld", &a[i]);
40     }
41
42     for(int i = 1; i <= n; ++ i) {
43         scanf("%lld", &b[i]);
44     }
```

```
45
46     for(int i = 1; i <= n; ++ i) {
47         for(int j = i; j <= n; j += i) {
48             if(mu[i] == 0) continue;
49             f[j] = ((f[j] + mu[i] * a[j / i]) % mod + mod) % mod;
50             g[j] = ((g[j] + mu[i] * b[j / i]) % mod + mod) % mod;
51         }
52     }
53
54     for(int i = 1; i <= n; ++ i) {
55         for(int j = i; j <= n; j += i) {
56             if(phi[i] * phi[j / i] == phi[j]) {
57                 //if(__gcd(i, j / i) == 1) {
58                     h[j] = (h[j] + f[i] * g[j / i] % mod) % mod;
59                 }
60             }
61         }
62
63         for(int i = 1; i <= n; ++ i) {
64             for(int j = i; j <= n; j += i) {
65                 c[j] = (c[j] + h[i] * j / i % mod) % mod;
66             }
67         }
68         for(int i = 1; i <= n; ++ i) {
69             ans = (ans ^ c[i]);
70         }
71         printf("%lld\n", ans);
72         return 0;
73     }
```

0x42.5 莫比乌斯反演扩展

莫比乌斯反演的非卷积形式

对于数论函数  $f, g$  和完全积性函数  $t$  且  $t(1) = 1$ :

$$f(n) = \sum_{i=1}^n t(i)g\left(\left\lfloor \frac{n}{i} \right\rfloor\right) \\ \iff \\ g(n) = \sum_{i=1}^n \mu(i)t(i)f\left(\left\lfloor \frac{n}{i} \right\rfloor\right)$$

证明

$$\begin{aligned} g(n) &= \sum_{i=1}^n \mu(i)t(i)f\left(\left\lfloor \frac{n}{i} \right\rfloor\right) \\ &= \sum_{i=1}^n \mu(i)t(i) \sum_{j=1}^{\lfloor \frac{n}{i} \rfloor} t(j)g\left(\left\lfloor \frac{\lfloor \frac{n}{i} \rfloor}{j} \right\rfloor\right) \\ &= \sum_{i=1}^n \mu(i)t(i) \sum_{j=1}^{\lfloor \frac{n}{i} \rfloor} t(j)g\left(\left\lfloor \frac{n}{ij} \right\rfloor\right) \\ &= \sum_{T=1}^n \sum_{i=1}^n \mu(i)t(i) \sum_{j=1}^{\lfloor \frac{n}{i} \rfloor} [ij = T]t(j)g\left(\left\lfloor \frac{n}{T} \right\rfloor\right) && \text{【先枚举 } ij \text{ 乘积】} \\ &= \sum_{T=1}^n \sum_{i|T} \mu(i)t(i)t\left(\frac{T}{i}\right)g\left(\left\lfloor \frac{n}{T} \right\rfloor\right) && \text{【} \sum_{j=1}^{\lfloor \frac{n}{i} \rfloor} [ij = T] \text{对答案的贡献为 1，于是省略】} \\ &= \sum_{T=1}^n g\left(\left\lfloor \frac{n}{T} \right\rfloor\right) \sum_{i|T} \mu(i)t(i)t\left(\frac{T}{i}\right) \\ &= \sum_{T=1}^n g\left(\left\lfloor \frac{n}{T} \right\rfloor\right) \sum_{i|T} \mu(i)t(T) && \text{【} t \text{ 是完全积性函数】} \\ &= \sum_{T=1}^n g\left(\left\lfloor \frac{n}{T} \right\rfloor\right) t(T) \sum_{i|T} \mu(i) \\ &= \sum_{T=1}^n g\left(\left\lfloor \frac{n}{T} \right\rfloor\right) t(T)\varepsilon(T) && \text{【} \mu * 1 = \varepsilon \text{】} \\ &= g(n)t(1) && \text{【当且仅当 } T=1, \varepsilon(T) = 1 \text{ 时】} \\ &= g(n) \quad \square \end{aligned}$$

## 0x43 欧拉反演

本身是没有欧拉反演这个东西的，在刘汝佳的蓝书上讲过一道利用欧拉函数的性质简便求解答案的题目，由于和莫比乌斯反演雷同，都是由一个基本性质推出一个反演公式，慢慢大家也将这种方法叫做欧拉反演。

我们知道欧拉函数的基本性质：

$$\varphi * 1 = id \Rightarrow n = \sum_{d|n} \varphi(d)$$

(性质的证明详见 0x33.1 常见积性函数的卷积的 **性质0x33.5** )

我们就可以利用这个性质，将  $n$  换成我们想要的东西，来直接求解答案（反演）。

例如我们可以令  $n = \gcd(i, j)$ ，显然有  $(d | \gcd(i, j) \rightarrow \gcd(i, j) | i, j \rightarrow d | i, j)$ ：

$$\begin{aligned} \gcd(i, j) &= \sum_{d|\gcd(i, j)} \varphi(d) \\ &= \sum_{d|i} \sum_{d|j} \varphi(d) \end{aligned}$$

我们可以尝试使用一下这个式子：

$$\begin{aligned} \sum_{i=1}^n \gcd(i, n) &= \sum_{i=1}^n \sum_{d|i} \sum_{d|n} \varphi(d) \\ &= \sum_{d|n} \sum_{i=1}^n \sum_{d|i} \varphi(d) \\ &= \sum_{d|n} \lfloor \frac{n}{d} \rfloor \varphi(d) \end{aligned}$$

一些结论：

**结论43.1:**  $\sum_{i=1}^n \sum_{j=1}^n \gcd(i, j) = \sum_{d=1}^n d \left( 2 \sum_{i=1}^{\lfloor \frac{n}{d} \rfloor} \varphi(i) - 1 \right)$

**结论43.2:**  $\sum_{i=1}^n \sum_{j=1}^m \gcd(i, j) = \sum_{d=1}^n \varphi(d) \lfloor \frac{n}{d} \rfloor \lfloor \frac{m}{d} \rfloor$  (例题)

**结论43.3:**  $\prod_{i=1}^n \prod_{j=1}^n \left( \frac{\text{lcm}(i, j)}{\gcd(i, j)} \right) = \frac{(n!)^{2n}}{\left( \prod_{d=1}^n d^{(2S_\varphi(\lfloor \frac{n}{d} \rfloor) - 1)} \right)^2}$  (例题)

### ● 竞赛例题选讲

**Problem A. 能量采集** ([P1447 [NOI2010]](<https://www.luogu.com.cn/problem/P1447>) )



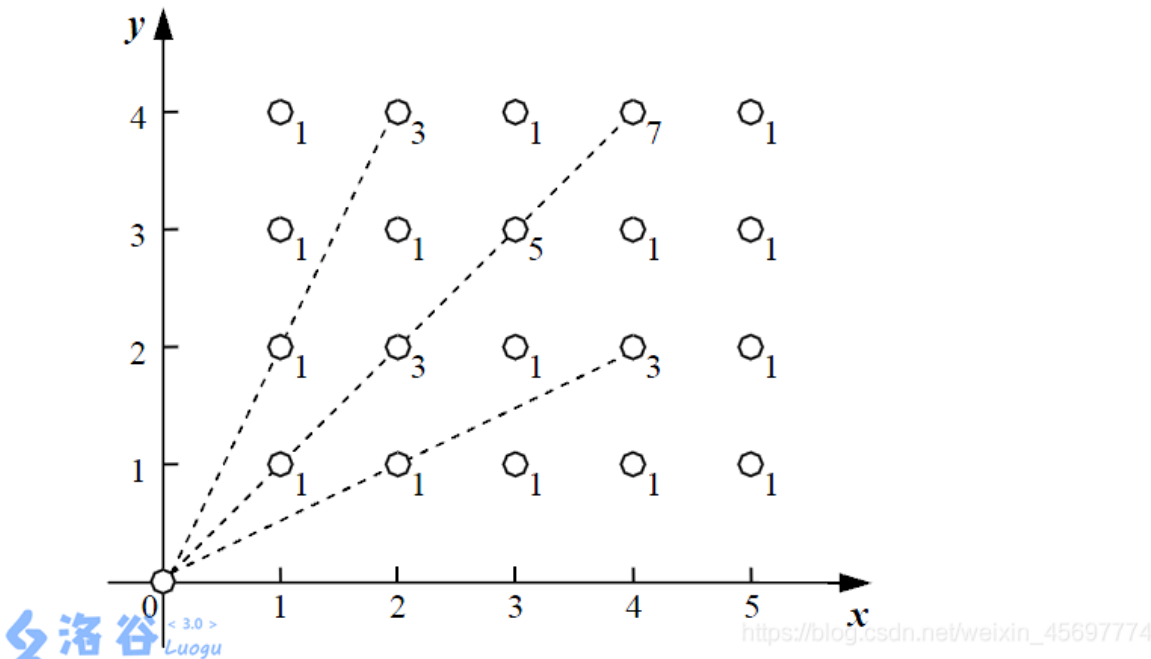
栋栋有一块长方形的地，他在地上种了一种能量植物，这种植物可以采集太阳光的能量。在这些植物采集能量后，栋栋再使用一个能量汇集机器把这些植物采集到的能量汇集到一起。

栋栋的植物种得非常整齐，一共有  $n$  列，每列有  $m$  棵，植物的横竖间距都一样，因此对于每一棵植物，栋栋可以用一个坐标  $(x, y)$  来表示，其中  $x$  的范围是 1 至  $n$ ， $y$  的范围是 1 至  $m$ ，表示是在第  $x$  列的第  $y$  棵。

由于能量汇集机器较大，不便移动，栋栋将它放在了一个角上，坐标正好是  $(0, 0)$ 。

能量汇集机器在汇集的过程中有一定的能量损失。如果一棵植物与能量汇集机器连接而成的线段上有  $k$  棵植物，则能量的损失为  $2k + 1$ 。例如，当能量汇集机器收集坐标为  $(2, 4)$  的植物时，由于连接线段上存在一棵植物  $(1, 2)$ ，会产生 3 的能量损失。注意，如果一棵植物与能量汇集机器连接的线段上没有植物，则能量损失为 1。现在要计算总的能量损失。

下面给出了一个能量采集的例子，其中  $n = 5$ ， $m = 4$ ，一共有 20 棵植物，在每棵植物上标明了能量汇集机器收集它的能量时产生的能量损失。



Solution

方法一：莫比乌斯反演 + 欧拉反演

先简单解释一下本题解题第一步用到的显然的结论。

数轴上任意一点  $(x_1, y_1)$  到原点之间线段上的经过的点  $(x_2, y_2)$ ， $\gcd(x_2, y_2)$  一定是  $\gcd(x_1, y_1)$  的因子。例如仪仗队那道题，从原点出发能看到的点一定都是  $\gcd(x, y) = 1$  的点。从原点到  $(x_1, y_1)$  所以经过的点的个数就是  $\gcd(x_1, y_1) - 1$ （去掉自身）（ $(2, 4)$  会有因子  $(1, 2)$ ）

证明来源：

设点  $(x, y)$  与  $(0, 0)$  之间有且仅有  $k$  个整数点，它们分别为  $(\frac{x}{\lambda_1}, \frac{y}{\lambda_1}) \sim (\frac{x}{\lambda_k}, \frac{y}{\lambda_k})$

$(\forall i \in [1, k], \lambda_i > 1)$

$\therefore \forall i \in [1, k]$  都有  $(\frac{x}{\lambda_i}, \frac{y}{\lambda_i})$  为整数点

即  $\frac{x}{\lambda_i}, \frac{y}{\lambda_i} \in \mathbb{Z}$

因为不保证  $\lambda \in \mathbb{Z}$  所以不能写成  $\lambda_i \mid x \wedge \lambda_i \mid y$

那么，我们令  $p = \frac{x}{\lambda_i}, q = \frac{y}{\lambda_i}, p, q \in \mathbb{Z}$

$\therefore \gcd(x, y) = \gcd(p\lambda_i, q\lambda_i)$

$\therefore \frac{\gcd(x, y)}{\lambda_i} = \gcd(p, q)$

$\therefore \lambda_i$  有且仅有  $k$  个取值

$\therefore \gcd(p, q)$  有且仅有  $k$  个取值，且必定为整数

$\therefore$  对  $\forall n > 1, \frac{\gcd(x, y)}{n}$  有且仅有  $k$  个整数取值

注： $n$  可为分数

$\therefore$  对  $\forall n > 0, \frac{\gcd(x, y)}{n}$  有且仅有  $(k + 1)$  个整数取值

这不就说明了  $\gcd(x, y) = k + 1$  吗？

https://blog.csdn.net/weixin\_45697774

显然坐标  $(x, y)$  的点与原点  $(0, 0)$  之间有  $\gcd(x, y) - 1$  棵植物

则每一棵损失的能量为  $2 \times (\gcd(x, y) - 1) + 1 = 2 \gcd(x, y) - 1$

则本题的答案为  $\sum_{i=1}^n \sum_{j=1}^m 2 \gcd(i, j) - 1$

$$= 2 \sum_{i=1}^n \sum_{j=1}^m \gcd(i, j) - n \times m$$

设  $f = \sum_{i=1}^n \sum_{j=1}^m \gcd(i, j)$  设  $n \leq m$

$$= \sum_{d=1}^n \sum_{i=1}^n \sum_{j=1}^m d [\gcd(i, j) = d]$$

$$= \sum_{d=1}^n d \sum_{i=1}^n \sum_{j=1}^m [\gcd(\frac{i}{d}, \frac{j}{d}) = 1]$$

$$= \sum_{d=1}^n d \sum_{i=1}^{\frac{n}{d}} \sum_{j=1}^{\frac{m}{d}} [\gcd(i, j) = 1]$$

$$= \sum_{d=1}^n d \sum_{i=1}^{\frac{n}{d}} \sum_{j=1}^{\frac{m}{d}} \sum_{k|\gcd(i, j)} \mu(k)$$

$$= \sum_{d=1}^n d \sum_{k=1}^{\frac{n}{d}} \mu(k) \sum_{i=1}^{\frac{n}{dk}} \sum_{j=1}^{\frac{m}{dk}} [k|i][k|j]$$

$$= \sum_{d=1}^n d \sum_{k=1}^{\frac{n}{d}} \mu(k) \frac{n}{kd} \times \frac{m}{kd}$$

令  $T = kd$   
则  $f = \sum_{T=1}^n \sum_{k|T} \mu(k) \frac{T}{k} \times \frac{n}{T} \times \frac{m}{T}$

令  $d$  为  $k$  (好看)  
 $f = \sum_{T=1}^n \sum_{d|T} \mu(d) \times \frac{T}{d} \times \frac{n}{T} \times \frac{m}{T}$   
 $= \sum_{T=1}^n \frac{n}{T} \times \frac{m}{T} \times \sum_{d|T} \mu(d) \times \frac{T}{d}$

显然  $\mu * id = \varphi$ ,  $id(n) = n$ ,  $id(\frac{T}{d}) = \frac{T}{d}$

则  $f = \sum_{T=1}^n \frac{n}{T} \times \frac{m}{T} \varphi(T)$   
 $= \sum_{T=1}^n \frac{n}{T} \times \frac{m}{T} \varphi(T)$

$\therefore$  答案为  $2f - n \times m$

$$= 2 \sum_{T=1}^n \frac{n}{T} \times \frac{m}{T} \varphi(T) - n \times m$$

[https://blog.csdn.net/weixin\\_45697774](https://blog.csdn.net/weixin_45697774)

Time

$O(n + \sqrt{n})$

Code

```
1 const int N = 500007;
2 int primes[N], cnt, phi[N];
3 int n, m;
4 bool vis[N];
5 int sum[N];
```

```

6
7 void init(int n)
8 {
9     phi[1] = 1;
10    for(int i = 2; i <= n; ++ i) {
11        if(vis[i] == 0) {
12            primes[ ++ cnt] = i;
13            phi[i] = i - 1;
14        }
15        for(int j = 1; j <= cnt && i * primes[j] <= n; ++ j) {
16            vis[i * primes[j]] = true;
17            if(i % primes[j] == 0) {
18                phi[i * primes[j]] = phi[i] * primes[j];
19                break;
20            }
21            phi[i * primes[j]] = phi[i] * phi[primes[j]];
22        }
23    }
24    for(int i = 1; i <= n; ++ i) {
25        sum[i] = sum[i - 1] + phi[i];
26    }
27 }
28
29 void solve()
30 {
31     int res = 0;
32     for(int l = 1, r; l <= n; l = r + 1) {
33         r = min(n / (n / l), m / (m / l));
34         res += (n / l) * (m / l) * (sum[r] - sum[l - 1]); //(sum[r] - sum[l - 1])已经是区间长度了
35     }
36     res = 2 * res - n * m;
37     printf("%lld\n", res);
38 }
39
40 signed main()
41 {
42     init(N - 7);
43     scanf("%lld%lld", &n, &m);
44     if(n > m) swap(n, m);
45     solve();
46     return 0;
47 }

```

方法二：容斥原理

有趣的是，我们可以利用容斥原理解决这个问题。

时间复杂度  $O(n \log n)$

令 $f[x]$ 为 $\text{GCD}(i,j)=x$ 的数对 $(i,j)$ 的个数,这个不是很好求

我们令 $g[x]$ 为存在公因数 $=x$ 的数对 $(i,j)$ 的个数(注意不是最大公因数!),显然有 $g[x]=(n/x)*(m/x)$

但是这些数对中有一些的最大公因数为 $2d,3d,4d$ ,我们要把他们减掉

于是最终 $f[x]=(n/x)*(m/x)-\sum(2*x \leq i*x \leq \min(m,n))f[i*x]$

从后向前枚举 $x$ 即可

[https://blog.csdn.net/weixin\\_45697774](https://blog.csdn.net/weixin_45697774)

Code

```

1 #define int long long
2 using namespace std;
3
4 const int N = 500007;
5
6 int n, m;
7 int f[N];
8
9 signed main()
10 {
11     scanf("%lld%lld", &n, &m);
12     if(n > m) swap(n, m);
13     int ans = 0;
14     for(int i = n; i; -- i) {
15         f[i] = (n / i) * (m / i);

```

```
16     for(int j = i << 1; j <= n; j += i)
17         f[i] -= f[j];
18     ans += (2 * i - 1) * f[i];
19 }
20 printf("%lld\n", ans);
21 return 0;
22 }
```

## Ox44 二项式反演<sup>1</sup>

我们知道容斥原理的一般形式为

$$|\overline{A_1} \cap \dots \cap \overline{A_n}| = |A| - \sum_{i=1}^n |A_i| + \sum_{1 \leq i_1 < i_2 \leq n} |A_{i_1} \cap A_{i_2}| - \dots + (-1)^n \cdot |A_1 \cap \dots \cap A_n|$$

那么根据德摩根定律显然有

$$|\overline{A_1 \cup \dots \cup A_n}| + |\overline{A_1} \cap \dots \cap \overline{A_n}| = |A|$$

即

$$|A_1 \cup A_2 \cup \dots \cup A_n| = \sum_{1 \leq i \leq n} |A_i| - \sum_{1 \leq i < j \leq n} |A_i \cap A_j| + \dots + (-1)^{n-1} \times |A_1 \cap A_2 \cap \dots \cap A_n|$$

那么我们设  $A_i^c$  为  $A_i$  的补集，变形可得

$$|A_1^c \cap A_2^c \cap \dots \cap A_n^c| = |S| - \sum_{1 \leq i \leq n} |A_i| + \sum_{1 \leq i < j \leq n} |A_i \cap A_j| - \dots + (-1)^n \times |A_1 \cap A_2 \cap \dots \cap A_n|$$

显然补集的补集就是原集，则有

$$|A_1 \cap A_2 \cap \dots \cap A_n| = |S| - \sum_{1 \leq i \leq n} |A_i^c| + \sum_{1 \leq i < j \leq n} |A_i^c \cap A_j^c| - \dots + (-1)^n \times |A_1^c \cap A_2^c \cap \dots \cap A_n^c|$$

设  $f(n)$  表示  $n$  个补集的交集大小， $g(n)$  表示  $n$  个原集的大小

则

$$|A_1^c \cap A_2^c \cap \dots \cap A_n^c| = |S| - \sum_{1 \leq i \leq n} |A_i| + \sum_{1 \leq i < j \leq n} |A_i \cap A_j| - \dots + (-1)^n \times |A_1 \cap A_2 \cap \dots \cap A_n|$$

即可表示为

$$f(n) = \sum_{i=0}^n (-1)^i \binom{n}{i} g(i)$$

而第二个等式

$$|A_1 \cap A_2 \cap \dots \cap A_n| = |S| - \sum_{1 \leq i \leq n} |A_i^c| + \sum_{1 \leq i < j \leq n} |A_i^c \cap A_j^c| - \dots + (-1)^n \times |A_1^c \cap A_2^c \cap \dots \cap A_n^c|$$

即可表示为

$$g(n) = \sum_{i=0}^n (-1)^i \binom{n}{i} f(i)$$

即：

$$f(n) = \sum_{i=0}^n (-1)^i \binom{n}{i} g(i) \Leftrightarrow g(n) = \sum_{i=0}^n (-1)^i \binom{n}{i} f(i)$$

我们令  $h(n) = (-1)^n g(n)$  带入上述公式即可得到**二项式反演的第一种形式**

- 二项式反演的第一种形式

$$f(n) = \sum_{i=0}^n \binom{n}{i} h(i) \Leftrightarrow \frac{h(n)}{(-1)^n} = \sum_{i=0}^n (-1)^i \binom{n}{i} f(i)$$

更一般的形式为：

$$f(n) = \sum_{i=m}^n \binom{n}{i} g(i) \Leftrightarrow g(n) = \sum_{i=m}^n (-1)^{n-i} \binom{n}{i} f(i)$$

- 二项式反演的第二种形式

$$f(n) = \sum_{i=n}^m \binom{i}{n} g(i) \Leftrightarrow g(n) = \sum_{i=n}^m (-1)^{i-n} \binom{i}{n} f(i)$$

二项式反演的第二种形式中，我们设  $f(n)$  表示 “钦定选择  $n$  个” 的方案数， $g(n)$  表示 “恰好选  $n$  个” 的方案数， $f(n)$  表示先钦定  $n$  个，再统计在已经钦定的情况下的方案数，那么对于任意的  $i \geq n$ ，在  $f(n)$  中被计算了  $\binom{i}{n}$  次，故  $f(n) = \sum_{i=n}^m \binom{i}{n} g(i)$ ，其中  $m$  是数目上界。而对于恰好选择了  $i$  个，从  $i$  个中钦定  $n$  个的方案数显然为  $\binom{i}{n}$ ，即  $g(i)$  也在  $f(i)$  中被计算了  $\binom{i}{n}$  次，然后根据容斥原理，可得  $g(n) = \sum_{i=n}^m (-1)^{i-n} \binom{i}{n} f(i)$ 。

形式一证明

$$\begin{aligned} f(n) &= \sum_{i=0}^n \binom{n}{i} \sum_{j=0}^i (-1)^{i-j} \binom{i}{j} f(j) \\ &= \sum_{i=0}^n \sum_{j=0}^i (-1)^{i-j} \binom{n}{i} \binom{i}{j} f(j) \\ &= \sum_{j=0}^n f(j) \sum_{i=j}^n (-1)^{i-j} \binom{n}{i} \binom{i}{j} \\ &= \sum_{j=0}^n \binom{n}{j} f(j) \sum_{i=j}^n \binom{n-j}{i-j} (-1)^{i-j} \quad \text{当} \\ &= \sum_{j=0}^n \binom{n}{j} f(j) \sum_{t=0}^{n-j} \binom{n-j}{t} (-1)^t 1^{n-j-t} \\ &= \sum_{j=0}^n \binom{n}{j} f(j) (1-1)^{n-j} \end{aligned}$$

$n-j \neq 0$  时,  $(1-1)^{n-j} = 0$

当  $n-j = 0$  时, 由于出现  $0^0$ ，而  $0^0$  没有任何意义，所以我们从组合角度出发，有  $\binom{n-j}{t} (-1)^t = 1$

故：
$$\sum_{t=0}^{n-j} \binom{n-j}{t} (-1)^t = [j = n]$$

即： $f(n) = \binom{n}{n} f(n) = f(n)$   
二项式反演第一种形式得证  $\square$

形式二证明

$$\begin{aligned} f(n) &= \sum_{i=n}^m \binom{i}{n} \sum_{j=i}^m (-1)^{j-i} \binom{j}{i} f(j) \\ &= \sum_{i=n}^m \sum_{j=i}^m (-1)^{j-i} \binom{i}{n} \binom{j}{i} f(j) \\ &= \sum_{j=n}^m f(j) \sum_{i=n}^j (-1)^{j-i} \binom{i}{n} \binom{j}{i} \\ &= \sum_{j=n}^m \binom{j}{n} f(j) \sum_{i=n}^j \binom{j-n}{j-i} (-1)^{j-i} \\ &= \sum_{j=n}^m \binom{j}{n} f(j) \sum_{t=0}^{j-n} \binom{j-n}{t} (-1)^t 1^{j-n-t} \\ &= \sum_{j=n}^m \binom{j}{n} f(j) (1-1)^{j-n} \\ &= \sum_{j=n}^m \binom{j}{n} f(j) [j = n] \\ &= \binom{n}{n} f(n) \\ &= f(n) \end{aligned}$$

二项式反演第二种形式得证  $\square$

● 基本性质定理

性质44.1：
$$f(n) = \sum_{i=0}^n C_n^i g(i) \iff g(n) = \sum_{i=0}^n (-1)^{n-i} C_n^i f(i)$$

性质44.2：
$$f(n) = \sum_{i=0}^n (-1)^i C_n^i g(i) \iff g(n) = \sum_{i=0}^n (-1)^i C_n^i f(i)$$

性质44.3：
$$f(n) = \sum_{i=n}^? C_i^n g(i) \iff g(n) = \sum_{i=n}^? (-1)^{i-n} C_i^n f(i)$$
（例题、例题）

性质44.4：
$$f(n) = \sum_{i=n}^? (-1)^i C_i^n g(i) \iff g(n) = \sum_{i=n}^? (-1)^i C_i^n f(i)$$

性质44.5：
$$f(n, m) = \sum_{i=0}^n \sum_{j=0}^m C_n^i C_m^j g(i, j) \iff g(n, m) = \sum_{i=0}^n \sum_{j=0}^m (-1)^{n+m-i-j} C_n^i C_m^j f(i, j)$$

性质44.6:  $f(n, m) = \sum_{i=0}^n \sum_{j=0}^m (-1)^{i+j} C_n^i C_m^j g(i, j) \iff g(n, m) = \sum_{i=0}^n \sum_{j=0}^m (-1)^{i+j} C_n^i C_m^j f(i, j)$

性质44.7:  $f(n, m) = \sum_{i=n}^? \sum_{j=m}^? C_i^n C_j^m g(i, j) \iff g(n, m) = \sum_{i=n}^? \sum_{j=m}^? (-1)^{i+j-n-m} C_i^n C_j^m f(i, j)$  (例题、例题)

性质44.8:  $f(n, m) = \sum_{i=n}^? \sum_{j=m}^? (-1)^{i+j} C_i^n C_j^m g(i, j) \iff g(n, m) = \sum_{i=n}^? \sum_{j=m}^? (-1)^{i+j} C_i^n C_j^m f(i, j)$

● 竞赛例题选讲

Problem A. 集合计数 ([BZOJ 2839])

一个有N个元素的集合有  $2^N$  个不同子集（包含空集），现在要在这  $2^N$  个集合中取出若干集合（至少一个），使得它们的交集的元素个数为  $K$ ，求取法的方案数，答案模1000000007。（是质数喔~）

$1 \leq N \leq 10^6, 0 \leq K \leq N$

Solution

设  $f(i)$  表示钦定交集元素为某  $i$  个，子集的交集至少为这  $i$  个元素的子集的方案数

显然有  $f(i) = \binom{n}{i} 2^{2^{n-i}-1}$

即先钦定  $i$  个必选的元素，所有子集至少包含该  $i$  个元素， 剩余  $n - i$  个元素，这  $n - i$  个元素，每个元素对于每一个子集而言，可选可不选，故能组成的至少包含这  $i$  个元素的子集方案数为  $2^{n-i}$  个，不允许存在空集，故一共是  $2^{n-i} - 1$  个可用子集，选择若干子集求交集的时候，每个子集可选可不选，子集的选择方案数为  $2^{2^{n-i}-1}$ 。

我们再设  $g(i)$  表示交集元素恰好为  $i$  个的方案数，答案显然就是  $g(k)$ ，则显然有

$$\begin{aligned} f(k) &= \binom{n}{k} (2^{2^{n-k}} - 1) \\ &= \sum_{i=k}^n \binom{i}{k} g(i) \end{aligned}$$

进行二项式反演显然有：

$$\begin{aligned} g(k) &= \sum_{i=k}^n (-1)^{i-k} \binom{i}{k} f(i) \\ &= \sum_{i=k}^n (-1)^{i-k} \binom{i}{k} \binom{n}{i} (2^{2^{n-i}} - 1) \end{aligned}$$

预处理后计算即可。

时间复杂度  $O(n)$

Code

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4 const int N = 1e6 + 7, mod = 1e9 + 7;
5
6 int n, m, s, t, k, ans, a[N];
7 int fact[N], infact[N], inv[N], pow22[N];
8
9 void pre_work(int n)
10 {
11     inv[1] = fact[0] = infact[0] = 1;
12     pow22[0] = 2;
13     for (int i = 2; i <= n; ++ i)
14         inv[i] = 1ll * (mod - mod / i) * inv[mod % i] % mod;
15     for (int i = 1; i <= n; ++ i) {
16         fact[i] = 1ll * fact[i - 1] * i % mod;
17         infact[i] = 1ll * infact[i - 1] * inv[i] % mod;
18         pow22[i] = 1ll * pow22[i - 1] * pow22[i - 1] % mod;
19     }
20 }
21
22 int C(int n, int m)
23 {
```



```
24     if(n == m) return 1;
25     return 1ll * fact[n] * infact[n - m] % mod * infact[m] % mod;
26 }
27
28 int main()
29 {
30     pre_work(N - 7);
31     scanf("%d%d", &n, &k);
32     ans = 0;
33     for (int i = k; i <= n; ++ i) {
34         if((i - k) & 1)
35             ans = (1ll * ans - 1ll * C(i, k) * C(n, i) % mod * (pow22[n - i] - 1 + mod) % mod + mod) % mod;
36         else ans = (1ll * ans + 1ll * C(i, k) * C(n, i) % mod * (pow22[n - i] - 1 + mod) % mod) % mod;
37     }
38     cout << ans << endl;
39     return 0;
40 }
```

## Ox45 斯特林反演

### 前置知识

这里先复习一下两类斯特林数与上升幂，下降幂之间的转换公式：

$$x^n = \sum_{i=0}^n S(n, i) x^{\bar{i}}$$
$$x^n = \sum_{i=0}^n (-1)^{n-i} S(n, i) x^{\bar{i}}$$
$$x^n = \sum_{i=0}^n (-1)^{n-i} s(n, i) x^i$$
$$x^{\bar{n}} = \sum_{i=0}^n s(n, i) x^i$$

根据下降幂的公式：

$$x^n = \sum_{i=0}^n S(n, i) x^{\bar{i}}$$

我们根据上面的公式将其变成上升幂：

$$x^n = \sum_{i=0}^n S(n, i) (-1)^i (-x)^{\bar{i}}$$

代入上升幂转第一类斯特林数的公式：

$$x^n = \sum_{i=0}^n S(n, i) (-1)^i \sum_{j=0}^i s(i, j) (-x)^j$$

把  $x$  的幂提前，换一下求和符号：

$$x^n = \sum_{j=0}^n x^j \sum_{i=j}^n S(n, i) s(i, j) (-1)^{i-j}$$

由于这里我们把  $x$  看成未知量，其他的都是已知量，所以我们可以把左右当作多项式，那么对比系数可得 **反转公式**：

$$\sum_{i=m}^n S(n, i) s(i, m) (-1)^{i-m} = [m = n]$$

同理我们可以得出第二个**反转公式**：

$$\sum_{i=m}^n s(n, i) S(i, m) (-1)^{i-m} = [m = n]$$

注意：反转公式  $-1$  的指数也可以写成  $n - i$ ，稍加分析可以发现  $m = n$  时成立， $m \neq n$  时有两种情况，一种不变，另一种会将答案取相反数，但是由于结果为 0 所以不影响。

### 斯特林反演及其证明

了解了上面的前置知识以后，我们引出**斯特林反演的公式**：

$$f(n) = \sum_{i=0}^n S(n, i) g(i) \iff g(n) = \sum_{i=0}^n (-1)^{n-i} s(n, i) f(i)$$

考虑证明：



我们先写出一个  $[i = n]$  的形式：

$$g(n) = \sum_{i=0}^n [i = n]g(i)$$

我们再把斯特林数以及  $[m = n]$  的式子套进去，也就是上面的反转公式（注意  $-1$  的指数）：

$$\begin{aligned} g(n) &= \sum_{i=0}^n g(i) \sum_{j=i}^n (-1)^{n-j} s_1(n, j) s_2(j, i) \\ &= \sum_{j=0}^n (-1)^{n-j} s_1(n, j) \sum_{i=0}^j s_2(j, i) g(i) \\ &= \sum_{i=0}^n (-1)^{n-i} s_1(n, i) f(i) \end{aligned}$$

综上所述，斯特林反演公式得证。□

当然由于反转公式的对称性，所以互换  $s$  和  $S$  依然成立。

**性质45.1：**  $f(n) = \sum_{i=0}^n S_n^i g(i) \iff g(n) = \sum_{i=0}^n (-1)^{n-i} s_n^i g(i)$

**性质45.2：**  $f(n) = \sum_{i=0}^n s_n^i g(i) \iff g(n) = \sum_{i=0}^n (-1)^{n-i} S_n^i f(i)$

**性质45.3：**  $f(n) = \sum_{i=n}^? S_i^n g(i) \iff g(n) = \sum_{i=n}^? (-1)^{i-n} s_i^n g(i)$

**性质45.4：**  $f(n) = \sum_{i=n}^? s_i^n g(i) \iff g(n) = \sum_{i=n}^? (-1)^{i-n} S_i^n f(i)$

● 竞赛例题选讲

**Problem A. 方阵**（[[2018雅礼集训1-16]](<https://vjudge.net/problem/TopCoder-13444>))

给出一个  $n \times m$  大小的矩形，每个位置可以填上  $[1, c]$  中的任意一个数，要求填好后任意两行互不等价且任意两列互不等价，两行或两列等价当且仅当对应位置完全相同，求方案数。

$n, m \leq 5000$

**Solution**

## 0x46 单位根反演

**单位根反演**

$$\forall k, [n \mid k] = \frac{1}{n} \sum_{i=0}^{n-1} \omega_n^{ik}$$

若  $n \mid k$ ，显然有  $\omega_n^{ik} = \omega^0 = 1$ ，故  $\frac{1}{n} \sum_{i=0}^{n-1} \omega_n^{ik} = 1$ 。

若  $n \nmid k$ ， $\frac{1}{n} \sum_{i=0}^{n-1} \omega_n^{ik} = \frac{1}{n} \frac{\omega_n^{nk} - \omega_n^0}{\omega_n^k - 1} = 0$

故原式得证。

若需计算某个多项式的特定倍数的系数和，即  $\sum_{i=0}^{\lfloor \frac{n}{k} \rfloor} [x^{ik}] f(x)$

则有：

$$\begin{aligned} \sum_{i=0}^{\lfloor \frac{n}{k} \rfloor} [x^{ik}] f(x) &= \sum_{i=0}^n [k|i] [x^i] f(x) \\ &= \sum_{i=0}^n [x^i] f(x) \frac{1}{k} \sum_{j=0}^{k-1} \omega_k^{ji} \\ &= \frac{1}{k} \sum_{i=0}^n a_i \sum_{j=0}^{k-1} \omega_k^{ij} \\ &= \frac{1}{k} \sum_{j=0}^{k-1} \sum_{i=0}^n a_i (\omega_k^j)^i \\ &= \frac{1}{k} \sum_{j=0}^{k-1} f(\omega_k^j) \end{aligned}$$

- 基本性质定理

性质46.1.1:  $[n|k] = \frac{\sum_{i=0}^{n-1} w_n^{ik}}{n}$

性质46.1.2:  $[a=b] = \frac{\sum_{i=0}^{n-1} w_n^{ai} w_n^{-ib}}{n} (a,b < n)$

- 推导结论

结论46.2.1:  $\sum_{i=0}^n C_n^i m^i a_{(i \bmod 4)} = \frac{1}{4} \sum_{j=0}^3 a_j \sum_{k=0}^3 w_4^{-kj} (m w_4^k + 1)^n$  (例题)

结论46.2.1:  $\sum_{i=0}^n C_n^i m^i \lfloor \frac{i}{k} \rfloor = \frac{1}{k} \left( nm (m+1)^{n-1} - \frac{1}{k} \sum_{t=0}^k (m \omega_k^t + 1)^n f(t) \right)$  (例题)

- 竞赛例题选讲

## 0x47 子集反演

说是子集反演，其实就是裸的容斥

- 基本性质定理

性质47.1:  $f(S) = \sum_{T \subseteq S} g(T) \iff g(S) = \sum_{T \subseteq S} (-1)^{|S|-|T|} f(T)$  (模板)

性质47.2:  $f(S) = \sum_{T \supseteq S} g(T) \iff g(S) = \sum_{T \supseteq S} (-1)^{|T|-|S|} f(T)$

性质47.3:  $f(S) = \sum_{T \subseteq S} g(T) \iff g(S) = \sum_{T \subseteq S} \mu(|S|-|T|) f(T)$  ( $\mu(S)$  在  $S$  有重复元素时为 0，否则为  $(-1)^{|S|}$ )

性质47.4:  $f(S) = \sum_{T \supseteq S} g(T) \iff g(S) = \sum_{T \supseteq S} \mu(|T|-|S|) f(T)$  ( $\mu(S)$  在  $S$  有重复元素时为 0，否则为  $(-1)^{|S|}$ )

- 竞赛例题选讲

Problem A. 黑暗前的幻想乡 ([Luogu P4336 [SHOI2016]](https://www.luogu.com.cn/problem/P4336))

题目描述

幽香上台以后，第一项措施就是要修建幻想乡的公路。幻想乡一共有  $n$  个城市，之前原来没有任何路。幽香向选民承诺要减税，所以她打算只修  $n - 1$  条公路将这些城市连接起来。但是幻想乡有正好  $n - 1$  个建筑公司，每个建筑公司都想在修路过程中获得一些好处。虽然这些建筑公司在选举前没有给幽香钱，幽香还是打算和他们搞好关系，因为她还指望他们帮她建墙。所以她打算让每个建筑公司都负责一条路来修。

每个建筑公司都告诉了幽香自己有能力负责修建的路是哪些城市之间的。所以幽香打算  $n - 1$  条能够连接幻想乡所有城市的边，然后每条边都交给一个能够负责该边的建筑公司修建，并且每个建筑公司都恰好修建一条边。

幽香现在想要知道一共有多少种可能的方案呢？两个方案不同当且仅当它们要么修的边的集合不同，要么边的分配方式不同。

https://blog.csdn.net/weixin\_45697774

$n \leq 17, P = 10^9 + 7$

Solution

题目就是要求由  $n - 1$  个公司每个公司一条边建成的生成树的方案数。

显然求生成树的方案数，我们直接用矩阵树定理计算即可。

现在考虑满足 **每个公司都只负责一条边的方案数** 如何计算。

显然有： $n \leq 17$  + 计数问题 = 容斥

我们可以先用矩阵树定理计算  $n - 1$  个公司包含的所有边集的生成树的个数，显然这里算出来的生成树，不一定  $n - 1$  个公司都参与了建设，我们用容斥原理减去不合法的即可。

显然就是枚举有多少个公司没有参与建设，然后利用容斥原理奇加偶减，答案减去容斥原理计算出的结果，变成奇减偶加。

我们设  $g(S)$  为最终的答案，即全集  $S$  全部被覆盖的情况的方案数。

$f(S)$  为公司恰好仅为  $S$  的方案。

显然有：

$$g(S) = \sum_{T \subseteq S} f(T)$$

根据子集反演可得：

$$f(S) = \sum_{T \subseteq S} (-1)^{|S|-|T|} g(T)$$

即答案就是所有生成树的方案数，减去 1 个公司没有参与，由剩下的  $n - 2$  个公司建成的生成树的个数，加上 2 个公司没有参与，由剩下的  $n - 3$  个公司建成的生成树的个数，减去 3 个公司没有参与，由剩下的  $n - 4$  个公司建成的生成树的个数...

我们可以通过二进制枚举来枚举具体选择了那几个公司的边，处理出此时的基尔霍夫矩阵，然后直接利用矩阵树定理高斯消元  $O(n^3)$  计算代数余子式的行列式即可。

注意我们矩阵树定理计算的时候是可以处理重边的，所以如果一条边被多个公司覆盖，就把它当成重边即可，这样都当成重边算，最后减下来是没有重边的。

Time

$O(2^n \times n^3 \log P)$

Code

```
1 #include <bits/stdc++.h>
2 #define int long long
3 using namespace std;
4
5 const int N = 100 + 7, M = 1e5 + 7, mod = 1e9 + 7;
6
7 int n, m[M], s, t, k, ans, a[N][N];
8 int siz[M];
9 int u[N][N], v[N][N];
10
11 int qpow(int a, int b)
12 {
13     int res = 1;
14     while(b) {
15         if(b & 1) res = res * a % mod;
16         a = a * a % mod;
17         b >>= 1;
18     }
19     return res;
```

```

20 }
21
22 int guass2(int n)// 辗转相除 = TLE
23 {
24     int det = 1;
25     for (int i = 2; i <= n; ++ i) {
26         for (int j = i + 1; j <= n; ++ j) {
27             while(a[j][i]) {
28                 int t = a[i][i] / a[j][i];
29                 for (int k = i; k <= n; ++ k)
30                     a[i][k] = (a[i][k] - t * a[j][k] + mod) % mod;
31                 swap(a[i], a[j]);
32                 det = -det;
33             }
34         }
35         det = (det * a[i][i]) % mod;
36         if(det == 0) return 0;
37     }
38     return (det + mod) % mod;
39 }
40
41 int guass(int n)// 求逆元 = AC
42 {
43     int det = 1;
44     for (int i = 2; i <= n; ++ i) {
45         for (int j = i; j <= n; ++ j) {
46             if(a[i][j]) {
47                 swap(a[i], a[j]);
48                 if(i != j)
49                     det = mod - det;
50                 break;
51             }
52         }
53
54         int inv = qpow(a[i][i], mod - 2);
55
56         for (int j = i + 1; j <= n; ++ j) {
57             if(a[j][i]) {
58                 int tmp = a[j][i] * inv % mod;
59                 for (int k = i; k <= n; ++ k) {
60                     a[j][k] = (a[j][k] - a[i][k] * tmp % mod + mod) % mod;
61                 }
62             }
63         }
64     }
65     for (int i = 2; i <= n; ++ i)
66         det = det * a[i][i] % mod;
67     return det;
68 }
69
70 signed main()
71 {
72     scanf("%lld", &n);
73     for (int i = 1; i <= n - 1; ++ i) {
74         scanf("%lld", &m[i]);
75         for (int j = 1; j <= m[i]; ++ j) {
76             scanf("%lld%lld", &u[i][j], &v[i][j]);
77
78             int x = u[i][j], y = v[i][j];
79             a[x][x] ++ ;
80             a[y][y] ++ ;
81             a[x][y] = (a[x][y] + mod - 1) % mod;
82             a[y][x] = (a[y][x] + mod - 1) % mod;
83         }
84     }
85     ans = guass(n);
86
87     for (int i = 1; i <= (1 << (n - 1)) - 1; ++ i) {
88         for (int j = 1; j <= n; ++ j)
89             for (int k = 1; k <= n; ++ k)
90                 a[j][k] = 0;

```

```
91
92     int cnt = 0;
93     int tmp = i, j;
94
95     for (j = 1; tmp; tmp >>= 1, ++ j) {
96         if((tmp & 1) == 0) continue;
97         cnt ++ ;
98         for (int k = 1; k <= m[j]; ++ k) {
99             int x = u[j][k], y = v[j][k];
100             a[x][x] ++ ;
101             a[y][y] ++ ;
102             a[x][y] = (a[x][y] + mod - 1) % mod;
103             a[y][x] = (a[y][x] + mod - 1) % mod;
104         }
105     }
106     if(cnt == n - 1) continue;
107     ans = (ans - (((n - 1) - cnt) & 1) ? 1 : -1) * guass(n) + mod) % mod;
108 }
109 printf("%lld\n", ans);
110 return 0;
111 }
```

## 0x48 最值反演 ( Min - Max 容斥)

- 基本性质定理

### 最值反演 —— Min - Max 容斥

设  $\min$  为集合最小值,  $\max$  为集合最大值

$$\max(S) = \sum_{T \subseteq S} (-1)^{|T|+1} \min(T)$$

$$\min(S) = \sum_{T \subseteq S} (-1)^{|T|+1} \max(T)$$

证明:

记  $S = \{a_i\}$ , 其中对于  $i < j$  有  $a_i < a_j$   
那么我们计算每一个  $a_i$  的贡献, 有

$$\begin{aligned} \sum_{T \subseteq S} (-1)^{|T|+1} \min\{T\} &= \sum_{i=1}^n a_i \sum_{j=0}^{n-i} (-1)^j \binom{n-i}{j} \\ &= \sum_{i=1}^n a_i [n-i=0] \\ &= \sum_{i=1}^n a_i [n=i] \\ &= a_n \\ &= \max\{S\} \end{aligned}$$

Min - Max 容斥在期望意义下也是成立的:, 设  $E(x)$  表示元素  $x$  出现的期望操作次数, 则:

$$\begin{aligned} E(\max(S)) &= \sum_{T \subseteq S} (-1)^{|T|-1} E(\min(T)) \\ E(\min(S)) &= \sum_{T \subseteq S} (-1)^{|T|-1} E(\max(T)) \end{aligned}$$

### 拓展 Min - Max 容斥

设  $k^{th} \max(S)$  表示  $S$  的第  $k$  大元素, 则

$$k^{th} \max(S) = \sum_{T \subseteq S} (-1)^{|T|-k} \binom{|T|-1}{k-1} \min(T)$$

证明同上, 同样在期望意义下成立:

$$E(k^{th} \max(S)) = \sum_{T \subseteq S} (-1)^{|T|-k} \binom{|T|-1}{k-1} E(\min(T))$$

性质48.1:  $\max(S) = \sum_{T \subseteq S} (-1)^{|T|+1} \min(T)$

性质48.2:  $\min(S) = \sum_{T \subseteq S} (-1)^{|T|+1} \max(T)$

性质48.3:  $E(\max(S)) = \sum_{T \subseteq S} (-1)^{|T|-1} E(\min(T))$

性质48.4:  $E(\min(S)) = \sum_{T \subseteq S} (-1)^{|T|-1} E(\max(T))$

性质48.5:  $k^{th} \max(S) = \sum_{T \subseteq S} (-1)^{|T|-k} \binom{|T|-1}{k-1} \min(T)$

性质48.6:  $E(k^{th} \max(S)) = \sum_{T \subseteq S} (-1)^{|T|-k} \binom{|T|-1}{k-1} E(\min(T))$

Problem A

有  $N(1 \leq N \leq 20)$ 张卡片，每包中含有这些卡片的概率为  $p_1, p_2, \cdots p_N$ 。每包至多一张卡片，可能没有卡片。求需要买多少包才能拿到所有的  $N$  张卡片，求次数的期望。

Solution

对于此题，我们想要求出所有卡片集合  $S$  中最晚出现的卡片的期望，就转化为计算  $S$  的所有子集中最早出现的期望

点集中最早出现的元素的期望是  $\min$ ，最晚出现的元素的期望是  $\max$ ；全部出现的期望就是最晚出现的元素的期望。

对于一个集合T，显然出现一张卡片的期望为  $\frac{1}{\sum_{i \in T} p_i}$

Code

```
1  const int N = 23, M = 1048576 + 3;
2  int n, V, cnt[M];
3  double ans, p[N], Min[M];
4  int main() {
5      while (~scanf("%d", &n)) {
6          V = (1 << n) - 1, ans = 0;
7
8          for (int i = 1; i <= n; ++i)
9              scanf("%lf", &p[i]);
10
11         for (int s = 1; s <= V; ++s) {
12             Min[s] = 0, cnt[s] = cnt[s >> 1] + (s & 1);
13
14             for (Re i = 1; i <= n; ++i)
15                 if (s & (1 << i - 1))
16                     Min[s] += p[i];
17
18             Min[s] = 1.0 / Min[s];
19         }
20
21         for (int t = 1; t <= V; ++t)
22             ans += (cnt[t] & 1) ? Min[t] : -Min[t];
23
24         printf("%lf\n", ans);
25     }
26 }
```

Problem B. 按位或 (Luogu P3175 [HAOI2015])

刚开始你有一个数字 0，每一秒钟你会随机选择一个  $[0, 2^n - 1]$  的数字，与你手上的数字进行按位或。选择数字  $i$  的概率是  $p_i$ 。保证  $0 \leq p_i \leq 1, \sum p_i = 1$ 。问期望多少秒后，你手上的数字变成  $2^n - 1$ 。

Solution

分析题目，我们每次选择的数是在  $[0, 2^n - 1]$  的数，进行按位或运算，显然无论如何选择，手里的数都不会大于  $2^n - 1$ 。我们希望最后手里的数变为  $2^n - 1$ ，即一个  $n$  位二进制数  $x$  中所有  $n$  位全部变成 1，因此我们可以将问题抽象成，有一个  $n$  位二进制数  $x$ ，开始  $n$  位上全是 0，我们每次选择一个数，可以使得  $x$  发生变化，最后  $x$  的所有位全部变为 1 的期望选择次数（时间）。

然后因为选择数字  $i$  的概率为  $p_i$ ，我们每次选择数  $i$ ，使得该二进制数发生一些变化，直到所有二进制数全部变为 1，显然我们可以联想到一个类似的问题模型：有  $n \leq 20$  种牌，每种牌无限张，得到第  $i$  种牌的概率为  $p_i$ ，问每张牌均至少拥有一张的期望购买次数（时间）。即 **HDU4336 Card Collector**，我们发现抽象出来问题模型之后，两个问题的模型几乎一模一样！

我们回想 **HDU4336 Card Collector** 是怎么解决的，Min - Max 容斥！

我们设  $a_i$  为得到卡片  $i$  的期望时间，集合  $S = \{a_i\}$ 。显然  $\max\{S\}$  为得到最后一张卡片的期望时间（此时我们已经得到了所有的卡片，游戏结束），则  $\min\{S\}$  为得到第一张卡片的期望时间。

显然有 Min - Max 容斥：

$$\max(S) = \sum_{T \subseteq S} (-1)^{|T|+1} \min(T)$$
$$\min(S) = \sum_{T \subseteq S} (-1)^{|T|+1} \max(T)$$

我们只需要直接爆搜计算集合  $S$  的所有子集  $T$  的  $\min(T) = \frac{1}{\sum_{i \in T} p_i}$  即可在  $O(2^n)$  的时间复杂度下求得  $\max(S)$ ，即为答案。

显然两个题目的模型是一模一样的，我们考虑使用 Min - Max 容斥解决本题。

类似的，我们设  $a_k$  为二进制数  $x$  的第  $k$  位变为 1 的期望时间，集合  $S = \{a_i\}$ ，显然这里的  $S$  为全集，也就是二进制数  $x$  的全部  $n$  位， $S$  的子集  $T$ ，表示某几位，也就是二进制数  $x$  的  $y \leq n$  位。

显然  $\max\{S\}$  为二进制数  $x$  的所有位均为 1 所有的期望时间， $\min\{S\}$  为二进制数  $x$  第一次有位数为 1 的期望时间。

$$\max(S) = \sum_{T \subseteq S} (-1)^{|T|+1} \min(T)$$

同样的，我们只需要计算集合  $S$  的所有子集  $T$  的  $\min(T)$  即可。

显然根据期望公式有：

$$\min(T) = \sum_{i=1}^{+\infty} i \times P(\min(T) = i)$$

其中  $P(\min(T) = i)$  表示在第  $i$  秒，集合  $T$  中第一次出现了 1 的概率，这也就意味着前  $i - 1$  秒内，每一秒，集合  $T$  中均没有出现 1，即集合  $T$  在全集  $S$  的补集  $W$  中出现了 1 ( $W = \complement_S T, W \cap T = \emptyset, W + T = S$ ，显然  $W = S \text{ xor } T$ ，因为这里代表的是二进制数)。设每一秒为事件  $A$ ，显然有  $P(A) = \sum_{w \subset W} P(w)$ ，其中  $P(w)$  表示出现的 1 是在集合  $w$  中的概率。

第  $i$  秒，集合  $T$  第一次出现 1，显然概率为  $1 - P(A) = 1 - \sum_{w \subset W} P(w)$ 。

显然为几何概型，期望为：

$$E(\min(T)) = \frac{1}{P}$$
$$= \frac{1}{1 - \sum_{w \subset W} P(w)}, W = \complement_S T$$

也就是说我们只需要计算出每个集合的子集和概率之和  $\sum_{w \subset W} P(w)$  即可利用 Min - Max 容斥在  $O(2^n)$  的复杂度下计算出答案。

那么要怎么算呢？

显然如果直接枚举子集计算的话需要  $O(3^n)$ ， $n \leq 20$ ，肯定要炸，所以考虑优化。

我们知道 FMT 计算按位或卷积的时候我们执行了这样一个  $O(n \log n)$  的过程：

$$\text{定义 } \text{FMT}(f)_n = \sum_{i \subseteq n} f_i$$
$$\text{FMT}(a)_n \times \text{FMT}(b)_n$$
$$= \sum_{i \subseteq n} a_i \sum_{j \subseteq n} b_j$$
$$= \sum_{k \subseteq n} \sum_{i \cup j = k} a_i b_j$$
$$= \text{FMT}(c)_n$$

好嘛，FMT 就是枚举子集...

即：序列经历过一次 FMT 按位或变换之后得到的新的序列就是原序列的子集和形式！

因此我们可以直接利用一次 FMT 在  $O(m \log m)$ ， $m = 2^n$  的复杂度下计算出所有的  $P(w) = \sum_{w \subset W} P(w)$ ，然后利用 Min - Max 容斥在  $O(2^n)$  的复杂度下计算出答案即可。

Code

```
1 // Problem: P3175 [HAOI2015]按位或
2 // Contest: Luogu
3 // URL: https://www.luogu.com.cn/problem/P3175
4 // Memory Limit: 125 MB
5 // Time Limit: 1000 ms
6 //
7 // Powered by CP Editor (https://cpeditor.org)
8
9 #include <bits/stdc++.h>
10
11 using namespace std;
12 const int N = 3e6 + 7;
13
14 int n, m, s, t, k, a[N];
```



```
15 double ans;
16 double p[N];
17 int cnt[N];
18
19 void FMT_OR(double *f, int n, double x = 1.0)
20 {
21     for (int o = 2; o <= n; o <= 1) {
22         for (int i = 0, k = o >> 1; i <= n; i += o) {
23             for (int j = 0; j < k; ++ j) {
24                 f[i + j + k] = f[i + j] * x + f[i + j + k];
25             }
26         }
27     }
28 }
29
30 int main()
31 {
32     scanf("%d", &n);
33     int S = (1 << n) - 1;
34     int limit = 1;
35     while(limit <= S) limit <= 1;
36     for (int i = 0; i <= S; ++ i)
37         scanf("%lf", &p[i]);
38     FMT_OR(p, limit);
39
40     for (int i = 1; i <= S; ++ i)
41         cnt[i] = cnt[i >> 1] + (i & 1);
42
43     ans = 0;
44     for (int i = 1; i <= S; ++ i) {
45         int SminusT = S ^ i;
46         if(1 - p[SminusT] < 1e-8)
47             return puts("INF"), 0;
48         ans += 1.0 / (1.0 - p[SminusT]) * (cnt[i] & 1 ? 1 : -1);
49     }
50     printf("%.10f\n", ans);
51     return 0;
52 }
```

Problem C. 重返现世 (Luogu P4707)

为了打开返回现世的大门，Yopilla 需要制作开启大门的钥匙。Yopilla 所在的迷失大陆有  $n$  种原料，只需要集齐任意  $k$  种，就可以开始制作。

Yopilla 来到了迷失大陆的核心地域。每个单位时间，这片地域就会随机生成一种原料。每种原料被生成的概率是不同的，第  $i$  种原料被生成的概率是  $\frac{p_i}{m}$ 。如果 Yopilla 没有这种原料，那么就可以进行收集。

Yopilla 急于知道，他收集到任意  $k$  种原料的期望时间，答案对 998244353 取模。

Solution

<https://www.luogu.com.cn/blog/Sooke/solution-p4707>

Ox49 拉格朗日反演

基本性质定理

对于两个多式  $F(x), G(x)$ ，两个多项式都是常数项均为 0 且 1 次项不为 0，如果有  $F(G(x)) = x$ ，我们称  $F, G$  互为复合逆，同时一定有  $G(F(x)) = x$ ，即可称  $G(x) = F^{-1}(x), F(x) = G^{-1}(x)$ 。

在这种情况下，拉格朗日反演

$$[x^n]F(x) = \frac{1}{n}[x^{-1}](\frac{1}{G(x)})^n = \frac{1}{n}[x^{n-1}](\frac{x}{G(x)})^n$$

扩展拉格朗日反演

$$[x^n]H(F(x)) = \frac{1}{n}[x^{-1}]H'(x)(\frac{1}{G(x)})^n = \frac{1}{n}[x^{n-1}]H'(x)(\frac{x}{G(x)})^n$$

Problem A. 大朋友和多叉树 (BZOJ 3684)

我们的大朋友很喜欢计算机科学，而且尤其喜欢多叉树。对于一棵带有正整数点权的有根多叉树，如果它满足这样的性质，我们的大朋友就会将其称作神犇的：点权为 1 的结点是叶子结点；对于任一点权大于 1 的结点  $u$ ， $u$  的孩子数目 `deg[u]` 属于集合  $D$ ，且  $u$  的点权等于这些孩子结点的点权之和。

给出一个整数  $s$ ，你能求出根节点权值为  $s$  的神犇多叉树的个数吗？请参照样例以更好的理解什么样的两棵多叉树会被视为不同的。

我们只需要知道答案关于 950009857 ( $453 \times 2^{21} + 1$ ，一个质数) 取模后的值。

$$1 \leq m < s \leq 10^5, 2 \leq d[i] \leq s$$

Solution

显然计数问题，我们可以写出二叉树的个数的生成函数为：

$$\begin{aligned} H(x) &= x + \prod_{i \in D} H^i(x) \\ \Rightarrow H(x) - \prod_{i \in D} H^i(x) &= x \end{aligned}$$

令  $F(x) = H(x), G(x) = x - \prod_{i \in D} x^i$ ，显然有  $G(F(x)) = x$

因此我们就可以进行拉格朗日反演：

$$[x^n]F(x) = \frac{1}{n} [x^{-1}] \frac{1}{G(x)^n}$$

显然直接辅以多项式全家桶暴力计算即可在  $O(n \log n)$  的时间复杂度下求出  $T(s)$  即为答案。

## 0x4A 反演常用技巧

反演往往是通过一系列和式变换，将一个无法承受的时间复杂度优化到一个可以接收的时间复杂度下，

**技巧4A.1：无关项提前**

$$\sum_{i=1}^n \sum_{j=1}^m n a_i b_j = n \sum_{i=1}^n a_i \sum_{j=1}^m b_j$$

**技巧4A.2：交换枚举顺序**

$$\sum_{i=1}^n a_i \sum_{j=1}^m b_j = \sum_{j=1}^m b_j \sum_{i=1}^n a_i$$

$$\sum_{i=1}^n a_i \sum_{d|i} b_d = \sum_{d=1}^n b_d \sum_{i=1}^{\lfloor \frac{n}{d} \rfloor} a_{id}$$

这样可以将枚举的约数变成枚举倍数。

**技巧4A.3：暴力破解法**

无从下手的时候，多一个暴力枚举判断，多一重枚举，构造出判断式，尝试构造  $\varepsilon = [n = 1]$  进行反演。

$$\begin{aligned} & \sum_{i=1}^n \sum_{j=1}^m f(\gcd(i, j)) \\ &= \sum_{d=1}^{\min(n, m)} \sum_{i=1}^n \sum_{j=1}^m f(d) [\gcd(i, j) = d] \\ &= \sum_{d=1}^{\min(n, m)} f(d) \sum_{i=1}^n \sum_{j=1}^m [\gcd(i, j) = d] \\ & \quad \dots \end{aligned}$$

**技巧4A.4：减少未知数**

$$\sum_{p|k} \sum_{d=1}^{\lfloor \frac{n}{p} \rfloor} \mu(d) \lfloor \frac{n}{pd} \rfloor \lfloor \frac{m}{pd} \rfloor$$

可以令  $k = pd$ （尽量减少未知数，减少枚举）

$$\sum_{p|k} \sum_{d=1}^{\lfloor \frac{n}{p} \rfloor} \mu\left(\frac{k}{p}\right) \lfloor \frac{n}{k} \rfloor \lfloor \frac{m}{k} \rfloor$$

**技巧4A.5：狄利克雷卷积优化**

尽力拼凑成积性函数卷积的形式

$$\begin{aligned} & \sum_{d=1}^n d^3 \sum_{k=1}^{\lfloor \frac{n}{d} \rfloor} \mu(k) k^2 \left( \sum_{i=1}^{\lfloor \frac{n}{dk} \rfloor} i \right)^2 \\ &= \sum_{T=1}^n T^2 \mu\left(\frac{T}{d}\right) d \left( \sum_{i=1}^{\lfloor \frac{n}{dk} \rfloor} i \right)^2 \\ &= \sum_{T=1}^n T^2 \varphi\left(\frac{T}{d} \times d\right) \left( \sum_{i=1}^{\lfloor \frac{n}{dk} \rfloor} i \right)^2 \\ &= \sum_{T=1}^n T^2 \varphi(T) \sum_{i=1}^{\lfloor \frac{n}{dk} \rfloor} i^3 \end{aligned}$$

**技巧4A.6：前缀和公式**

$$\begin{aligned}1 + 2 + 3 + \cdots + (n - 1) &= \frac{1}{2}n^2 - \frac{1}{2}n = \frac{n(n - 1)}{2} \\1^2 + 2^2 + 3^2 + \cdots + (n - 1)^2 &= \frac{1}{3}n^3 - \frac{1}{2}n^2 + \frac{1}{6}n = \frac{n(n - 1)(2n - 1)}{6} \\1^3 + 2^3 + 3^3 + \cdots + (n - 1)^3 &= \frac{1}{4}n^4 - \frac{1}{2}n^3 + \frac{1}{4}n^2 = \frac{n^2(n - 1)^2}{4} \\1^4 + 2^4 + 3^4 + \cdots + (n - 1)^4 &= \frac{1}{5}n^5 - \frac{1}{2}n^4 + \frac{1}{3}n^3 - \frac{n}{30} = \frac{n(n - 1)(2n - 1)(3n^2 - 3n - 1)}{30} \\1^5 + 2^5 + 3^5 + \cdots + (n - 1)^5 &= \frac{1}{6}n^6 - \frac{1}{2}n^5 + \frac{5}{12}n^4 - \frac{1}{12}n^2 = \frac{n^2(n - 1)^2(2n^2 - 2n - 1)}{12}\end{aligned}$$

证明

规定  $C(n, k)$  表示从  $n$  个数中取  $k$  个的组合数。

$$\begin{aligned}2^{n+1} &= 1 + C(n + 1, 1)1^n + C(n + 1, 2)1^{n-1} + C(n + 1, 3)1^{n-2} + \cdots + C(n + 1, n + 1) \\3^{n+1} &= 2^{n+1} + C(n + 1, 1)2^n + C(n + 1, 2)2^{n-1} + c(n + 1, 3)2^{n-2} + \cdots + c(n + 1, n + 1)\end{aligned}$$

.....

$$(n + 1)^{n+1} = n^{n+1} + C(n + 1, 1)n^n + C(n + 2, 2)n^{n-1} + \cdots + C(n + 1, n + 1)$$

上面  $n$  个式子相加

$$(n + 1)^{n+1} = 1 + (n + 1)(1^n + 2^n + \ldots + n^n) + C(n + 2, 2)(1^{n-1} + 2^{n-1} + \ldots + n^{n-1}) + \ldots + C(n + 1, n)(1 + 2 + 3 + \ldots + n) + n$$

整理得

$$1^n + 2^n + \cdots + n^n = (n + 1)^n - \frac{1}{n+1} - (\frac{1}{n+1})(C(n + 2, 2)(1^{n-1} + 2^{n-1} + \ldots + n^{n-1}) + \ldots + C(n + 1, n)(1 + 2 + 3 + \cdots + n) + n)$$

我们知道 1 次的前缀和为:  $1 + 2 + \cdots + n = \frac{n(1 + n)}{2}$

因此我们就可以递推得到 2次, 3 次, 乃至  $n$  次前缀和公式

## ox4B. Dirichlet 前缀和

### ox4B.1 Dirichlet 前缀和

#### Template Problem Dirichlet 前缀和 (P5495)

给定一个长度为  $n$  的数列  $a_1, a_2, a_3, \dots, a_n$ 。

现在你要求出一个长度为  $n$  的数列  $b_1, b_2, b_3, \dots, b_n$ ，满足

$$b_d = \sum_{i|d} a_i$$

#### Solution

如果我们直接枚举倍数的话显然可以做到  $O(\sum_{i=1}^n \left\lfloor \frac{n}{i} \right\rfloor) = O(n \log n)$ ，考虑优化。

根据唯一分解定理，我们可以设  $i = \prod p_d^{\alpha_d}, j = \prod p_d^{\beta_d}$ ，则  $a_i$  贡献到  $b_j$  当且仅当  $\forall d, \alpha_d \leq \beta_d$ 。

发现这个实际上就是一个关于质因子分解后的指数的高维前缀和（例如FMT），即每个数字会被它除以所有质因子的那个数转移过来，所有的质因子构成了一组正交基，核心代码：

```
1 for(int i = 1; i <= cnt && primes[i] <= n; ++ i)
2     for(int j = 1; j * primes[i] <= n; ++ j)
3         a[j * primes[i]] += a[j];
```

显然时间复杂度同埃氏筛： $O(n \log \log n)$

#### Code

```
1 #define uint unsigned int
2 uint seed;
3 inline uint getNext() {
4     seed ^= seed << 13;
5     seed ^= seed >> 17;
6     seed ^= seed << 5;
7     return seed;
8 }
9 uint a[N], b[N];
10 uint ans;
11 int n, m;
12 bool vis[N];
13 int primes[N], cnt;
14 void get_primes(int n)
15 {
```

```

16     for(int i = 2; i <= n; ++ i) {
17         if(vis[i] == 0)
18             primes[ ++ cnt] = i;
19         for(int j = 1; j <= cnt && i * primes[j] <= n; ++ j) {
20             vis[i * primes[j]] = true;
21             if(i % primes[j] == 0) break;
22         }
23     }
24 }
25 int main()
26 {
27     get_primes(N - 1);
28     scanf("%d%u", &n, &seed);
29     for(int i = 1; i <= n; ++ i)
30         a[i] = getnext();
31     for(int i = 1; i <= cnt && primes[i] <= n; ++ i)
32         for(int j = 1; j * primes[i] <= n; ++ j)
33             a[j * primes[i]] += a[j];
34     ans = a[1];
35     for(int i = 2; i <= n; ++ i)
36         ans ^= a[i];
37     printf("%u\n", ans);
38     return 0;
39 }

```

## 0x4B.2 Dirichlet 后缀和

$$b[i] = \sum_{i|d} a[d]$$

我们这里是从一个数字本身 转移到 这个数字除以所有质因子的数。

```

1     for(int i = 1; i <= cnt && primes[i] <= n; ++ i)
2         for(int j = n / primes[i]; j ; -- j)
3             a[j] += a[j * primes[i]];

```

## 0x4B.2 倒推 Dirichlet 前缀和

$$b[i] = \sum_{d|i} a[d]$$

这里是我们知道数组  $b$  , 求数组  $a$

```

1     for(int i = cnt; i ; -- i)
2         for(int j = n / primes[i]; j ; -- j)
3             a[j * primes[i]] -= a[j];

```

## 0x4B.3 倒推 Dirichlet 后缀和

$$b[i] = \sum_{i|d} a[d]$$

同上, 我们知道数组  $b$  , 求数组  $a$

```

1     for(int i = cnt; i ; -- i)
2         for(int j = 1; j * primes[i] <= n; ++ j)
3             a[j] -= a[j * primes[i]];

```

# 0x50 筛法

## 0x51 线性筛法

在上面筛质数的时候, 我们讲解了如何使用线性筛线性地筛出  $1, \dots, n$  中的所有质数。

每次只用一个数用小于当前这个数最小质因子的质数去筛其他数, 即**保证每个合数  $i \times p_j$  只都被自己的最小质因子  $p_j$  筛掉一遍**, 所以复杂度保证是  $O(n)$  的。

我们知道, 凡是积性函数, 均可由线性筛法来求解。

0x51.1 线性筛法求欧拉函数

我们注意到在线性筛中，每一个合数都是被最小的质因子筛掉，筛法求素数的同时也得到了每个数的最小质因子，这是线性筛求欧拉函数的关键。

1. 考虑  $n = p_j^k, \varphi(n)$

显然有  $\varphi(p_j^k) = p_j^k - p_j^{k-1} = p_j^{k-1} \times (p_j - 1)$

2. 考虑  $n = i \times p_j, \varphi(n)$ , 当  $p_j \mid i$

即  $i$  含有因子  $p_j$ , 由于  $i = \frac{n}{p_j}$  即  $i$  拥有  $n$  的所有质因子，由唯一分解定理及欧拉函数计算式得：

$$\begin{aligned}\varphi(n) &= n \times \prod_{i=1}^s \frac{p_i - 1}{p_i} \\ &= p_1 \times n' \times \prod_{i=1}^s \frac{p_i - 1}{p_i} \\ &= p_1 \times \varphi(n')\end{aligned}$$

3. 考虑  $n = i \times p_j, \varphi(n)$ , 当  $p_j \nmid i$

即  $i$  与  $p_j$  互质，积性函数显然有性质：

$$\begin{aligned}\varphi(n) &= \varphi(i) \times \varphi(p_j) \\ &= \varphi(i) \times (p_j - 1)\end{aligned}$$

由于我们仅在在线性筛的框架上增加了一些细节，所以时间复杂度依然是  $O(n)$  的。

Code

```
1 void get_euler(int n)
2 {
3     phi[1] = 1;
4     for(int i = 2; i <= n; ++ i) {
5         if(!vis[i]) {
6             primes[ ++ cnt] = i;
7             phi[i] = i - 1;
8         }
9         for(int j = 1; j <= cnt && i * primes[j] <= n; ++ j) {
10             vis[i * primes[j]] = true;
11             if(i % primes[j] == 0) { //最小的质因子
12                 phi[i * primes[j]] = phi[i] * primes[j];
13                 break;
14             }
15             phi[i * primes[j]] = phi[i] * (primes[j] - 1);
16         }
17     }
18 }
```

0x51.2 线性筛求莫比乌斯函数

同样考虑三种情况即可。

1. 考虑  $n = p_j^k, \mu(n)$

显然有

$$\mu(p_j) = -1$$

$$\mu(p_j^k) = 0, k > 1$$

2. 考虑  $n = i \times p_j, \mu(n)$ , 当  $p_j \mid i$

$$\mu(n) = 0$$

3. 考虑  $n = i \times p_j, \mu(n)$ , 当  $p_j \nmid i$

$$\mu(n) = -\mu(i)$$

Code

```
1 void get_mu(int n)
2 {
3     cnt = 0, mu[1] = 1;
4     for(int i = 2; i <= n; ++ i) {
5         if(vis[i] == 0){
6             primes[ ++ cnt] = i;
```

```
7         mu[i] = -1;
8     }
9     for(int j = 1; j <= cnt && i * primes[j] <= n; ++ j) {
10         vis[primes[j] * i] = 1;
11         if(i % primes[j] == 0) break;
12         mu[i * primes[j]] -= mu[i];
13         //mi[i * primes[j]] = -mu[i];
14     }
15 }
16 for(int i = 1; i <= n; ++ i)
17     sum[i] += sum[i - 1] + mu[i];
18 }
```

0x51.3 线性筛求约数个数函数

**约数个数函数：** $d(n)$ ：表示  $n$  的正因子数目。

$$d(n) = \sum_{i|n} 1$$

**定理51.3.1：** 若  $n = \prod_{i=1}^m p_i^{c_i}$ ，则  $d_i = \prod_{i=1}^m c_i + 1 = (c_1 + 1) \times \cdots \times (c_m + 1)$

证明：我们知道  $p_i^{c_i}$  的约数一共有  $p_i^0, p_i^1, \cdots, p_i^{c_i}$  个，即对于每个质因子  $p_i$  都可以选择其出现 0 次，1 次， $\cdots, c_i$  次。我们根据乘法原理，可得  $n$  的约数个数为  $\prod_{i=1}^m c_i + 1$

这里我们规定  $d_i$  为  $i$  的约数个数， $num_i$  表示  $i$  的最小质因子出现次数。

随机数据下，约数个数的期望是  $O(\ln n)$

Code

```
1 void get_divisor_num(int n) {
2     d[1] = 1;
3     for (int i = 2; i <= n; ++i) {
4         if (vis[i] == 0) {
5             vis[i] = 1;
6             primes[ ++ cnt] = i;
7             d[i] = 2, num[i] = 1;
8         }
9         for (int j = 1; j <= cnt && i * primes[j] <= n; ++ j) {
10             vis[primes[j] * i] = 1;
11             if (i % primes[j] == 0) {
12                 num[i * primes[j]] = num[i] + 1;//这里的primes[j]一定是i的最小质因子
13                 d[i * primes[j]] = d[i] / num[i * primes[j]] * (num[i * primes[j]] + 1);
14                 //数i*primes[j]的质因子primes[j]的出现次数num增加,即公式中的ci增加,对答案的贡献增加,所以需要更新
15                 break;
16             } else num[i * primes[j]] = 1, d[i * primes[j]] = d[i] * 2;
17         }
18     }
19 }
20 }
```

0x51.4 线性筛求约数和函数

**约数和函数：** $\sigma(n)$ ：表示  $n$  的所有正因子之和。

$$\sigma(n) = \sum_{d|n} d$$

**定理51.3.2：** 若  $n = \prod_{i=1}^m p_i^{c_i}$ ，则  $\sigma_i = \prod_{i=1}^m (\sum_{j=0}^{c_i} (p_i)^j) = (1 + p_1 + p_1^2 + \cdots + p_1^{c_1}) \times \cdots \times (1 + p_m + p_m^2 + \cdots + p_m^{c_m})$

证明略，思路同约数个数函数  $d(n)$  的证明。

这里我们规定  $f_i$  表示  $i$  的约数和， $g_i$  表示  $i$  的最小质因子  $p + p^1 + p^2 + \cdots p^k$ 。

```
1 void get_divisor_sum() {
2     g[1] = f[1] = 1;
3     for (int i = 2; i <= n; ++ i) {
4         if (vis[i] == 0) {
5             vis[i] = 1;
6             primes[ ++ cnt] = i;
7             g[i] = i + 1;
8             f[i] = i + 1;
9         }
10        for (int j = 1; j <= cnt && i * primes[j] <= n; ++ j) {
11            v[i * primes[j]] = 1;
```



```
12         if (i % primes[j] == 0) {
13             g[i * primes[j]] = g[i] * primes[j] + 1;
14             f[i * primes[j]] = f[i] / g[i] * g[i * primes[j]]; //更新
15             break;
16         } else {
17             f[i * primes[j]] = f[i] * f[primes[j]];
18             g[i * primes[j]] = primes[j] + 1;
19         }
20     }
21 }
22 for (int i = 1; i <= n; ++ i)
23     f[i] = (f[i - 1] + f[i]) % mod;
24 }
```

我们可以利用线性筛求解其他的**所有**积性函数的值，仅需根据具体的性质自行推导两种情况的处理办法即可。

$n$  的正约数之积....  $n^{\left\lfloor \frac{d(x)}{2} \right\rfloor}$

## 0x52 杜教筛

### 0x52.1 杜教筛

杜教筛被用来处理数论函数的前缀和问题。对于一个前缀和，杜教筛可以在低于线性时间的复杂度 ( $O(n^{\frac{2}{3}})$ ) 内求解。

由  $f(n)$ , 计算  $S(n) = \sum_{i=1}^n f(i)$  要求低于线性, 构造出形如  $\lfloor \frac{n}{d} \rfloor$   $\longrightarrow$  整除分块优化

即: 由  $S(n)$  构造关于  $S(\lfloor \frac{n}{d} \rfloor)$  的递推式.

构造两个积性函数  $h, g$ . 满足卷积:  $h = g * f$

$$\therefore h(i) = \sum_{d|i} g(d) f(\frac{i}{d})$$

$$\therefore \sum_{i=1}^n h(i) = \sum_{i=1}^n \sum_{d|i} g(d) f(\frac{i}{d})$$

$$= \sum_{d=1}^n g(d) \sum_{i=1}^{\lfloor \frac{n}{d} \rfloor} f(i)$$

$$= \sum_{d=1}^n g(d) S(\lfloor \frac{n}{d} \rfloor)$$

$$= \sum_{d=1}^n g(d) S(\lfloor \frac{n}{d} \rfloor)$$

$$= \sum_{d=1}^n g(d) S(\lfloor \frac{n}{d} \rfloor)$$

$$\text{即 } \sum_{i=1}^n h(i) = g(1) \cdot S(n) + \sum_{d=2}^n g(d) S(\lfloor \frac{n}{d} \rfloor)$$

$$g(1) \cdot S(n) = \sum_{i=1}^n h(i) - \sum_{d=2}^n g(d) S(\lfloor \frac{n}{d} \rfloor)$$

此外实际上是对所有  $i \leq n$  做贡献, 交换枚举顺序  
先枚举  $d$  提出  $g$ .

$\because d|i \therefore i$  为  $d$  的倍数.  
则  $d$  会对  $d$  的倍数  $k \cdot d$  做贡献 (即  $i$ )

贡献为  $g(d) \cdot f(\frac{k \cdot d}{d}) = g(d) \cdot f(k)$

而在  $n$  中  $d$  的倍数为  $d \sim \lfloor \frac{n}{d} \rfloor d$ , 有  $\lfloor \frac{n}{d} \rfloor$  个.

故得到下式

$$\sum_{i=1}^n \sum_{d|i} g(d) \cdot f(\frac{i}{d}) = \sum_{d=1}^n g(d) \sum_{i=1}^{\lfloor \frac{n}{d} \rfloor} f(i)$$

$$= \sum_{d=1}^n g(d) S(\lfloor \frac{n}{d} \rfloor) \quad (S(n) = \sum_{i=1}^n f(i))$$

此外  $i$  与  $d$  无关, 可替换

$$g(1) \cdot S(n) = \sum_{i=1}^n h(i) - \sum_{i=2}^n g(i) S(\lfloor \frac{n}{i} \rfloor) \longrightarrow \text{杜教筛公式.}$$

$$\sum_{i=1}^n \sum_{d|i} g(d) f(\frac{i}{d})$$

int sum = 0;

for (i = 1 to n)

for (d | i)

sum += g(d) \* f(i/d)

杜教筛变换  
等价

$$\sum_{d=1}^n g(d) \sum_{d|i} f(\frac{i}{d})$$

int sum = 0;

for (d = 1 to n)

int tmp = 0;

for (i = 1 to n & d | i)

tmp += f(i/d)

sum += g(d) \* tmp

杜教筛其实特别简单, 就是一个构造 + 和式转换, 利用 Dirichlet 卷积构造两个积性函数卷起来, 将要求的前缀和  $s(n)$  构造成为  $s(\lfloor \frac{n}{i} \rfloor)$  的形式, 这样我们就可以用整除分块来优化复杂度, 可以快速解决一类积性函数的前缀和,  $n$  可以达到  $10^9 \sim 10^{10}$ , 积性函数比如莫比乌斯函数  $\mu(x)$ , 欧拉函数  $\phi(x)$ , 约数和函数  $\sigma_k(x)$ , 约数个数函数  $\sigma(x)$  等等。

#### • 基本性质定理

性质 52.1.1:  $g(1)S(n) = \sum_{i=1}^n (f * g)(i) - \sum_{d=2}^n g(d)S(\lfloor \frac{n}{d} \rfloor)$  (其中  $S(n) = \sum_{i=1}^n f(i)$ )

#### • 推导结论:

https://blog.csdn.net/weixin\_45697774

性质52.2.1:  $S_{\mu(x)}(n) = 1 - \sum_{d=2}^n S\left(\left\lfloor \frac{n}{d} \right\rfloor\right)$  (模板)

性质52.2.2:  $S_{\varphi(x)}(n) = \sum_{i=1}^n i - \sum_{d=2}^n S\left(\left\lfloor \frac{n}{d} \right\rfloor\right)$  (模板)

性质52.2.3:  $S_{(n^2\varphi(n))}(n) = \sum_{i=1}^n i^3 - \sum_{d=2}^n d^2 S\left(\left\lfloor \frac{n}{d} \right\rfloor\right)$  (例题)

## 0x52.2 求欧拉函数前缀和

### 欧拉函数前缀和

$$S(n) = \sum_{i=1}^n \varphi(i) \quad \text{利用欧拉函数性质: } n = \sum_{d|n} \varphi(d)$$

#### ① 直接套用杜教筛公式

$$h = f * g = \sum_{d|n} f(d) g\left(\frac{n}{d}\right)$$

$$n = \sum_{d|n} \varphi(d) \Rightarrow id = \varphi * I$$

$$\text{则 } h = id, g = I$$

$$\text{杜教筛公式: } g(1) S(n) = \sum_{i=1}^n h(i) - \sum_{i=2}^n g(i) S\left(\left\lfloor \frac{n}{i} \right\rfloor\right)$$

$$\text{得: } S(n) = \sum_{i=1}^n i - \sum_{i=2}^n S\left(\left\lfloor \frac{n}{i} \right\rfloor\right)$$

#### ② 自行从头推导★

$$= \left( \frac{n(n+1)}{2} - \sum_{i=2}^n S\left(\left\lfloor \frac{n}{i} \right\rfloor\right) \right) / g(1)$$

由欧拉函数重要性质  $n = \sum_{d|n} \varphi(d)$  推导出来

$$n = \sum_{d|n} \varphi(d)$$

$$n = \varphi(n) + \sum_{d|n, d < n} \varphi(d)$$

$$\varphi(n) = n - \sum_{d|n, d < n} \varphi(d)$$

$$\sum_{i=1}^n \varphi(i) = \sum_{i=1}^n \left( i - \sum_{d|i, d < i} \varphi(d) \right)$$

$$\sum_{i=1}^n \varphi(i) = \frac{n(n+1)}{2} - \sum_{i=1}^n \sum_{d|i, d < i} \varphi(d)$$

改变枚举变量: 枚举  $\frac{i}{d}$ , 即枚举  $d$  的倍数  $i$ , 且  $i \neq d$ . 故从2开始

$$\sum_{i=1}^n \varphi(i) = \frac{n(n+1)}{2} - \sum_{\frac{i}{d}=2}^n \sum_{d=1}^{\left\lfloor \frac{n}{i} \right\rfloor} \varphi(d)$$

$$\text{设 } S(n) = \sum_{i=1}^n \varphi(i), \text{ 将 } \frac{i}{d} \text{ 写为 } i$$

$$\text{得 } S(n) = \left( \frac{n(n+1)}{2} - \sum_{i=2}^n S\left(\left\lfloor \frac{n}{i} \right\rfloor\right) \right) / g(1)$$



## 0x52.3 求莫比乌斯函数前缀和

## 莫比乌斯前缀和

$$S(n) = \sum_{i=1}^n \mu(i), \text{ 利用莫比乌斯函数性质: } \sum_{d|n} \mu(d) = \begin{cases} 1, n=1 \\ 0, n>1 \end{cases}$$

## ① 直接套用杜教筛公式

$$\Rightarrow \sum_{d|n} \mu(d) = [n=1]$$

$$h = f * g = \sum_{d|n} f(d) g(\frac{n}{d})$$

$$\sum_{d|n} \mu(d) = [n=1] \Rightarrow \varepsilon = \mu * I$$

$$\text{故 } h = \varepsilon, g = I$$

$$g(1) S(n) = \sum_{i=1}^n h(i) - \sum_{i=2}^n g(i) S(\lfloor \frac{n}{i} \rfloor)$$

$$\begin{aligned} \text{得: } S(n) &= \sum_{i=1}^n \varepsilon(i) - \sum_{i=2}^n S(\lfloor \frac{n}{i} \rfloor) \\ &= (1 - \sum_{i=2}^n S(\lfloor \frac{n}{i} \rfloor)) / g(1) \end{aligned}$$

## ② 自行推导(套路)

$$\sum_{d|n} \mu(d) = [n=1]$$

$$[n=1] = \mu(n) + \sum_{d|n, d < n} \mu(d)$$

$$\mu(n) = [n=1] - \sum_{d|n, d < n} \mu(d)$$

$$\sum_{i=1}^n \mu(i) = 1 - \sum_{i=1}^n \sum_{d|i, d < i} \mu(d)$$

$$\sum_{i=1}^n \mu(i) = 1 - \sum_{\frac{i}{d}=2}^n \sum_{d=2}^{\lfloor \frac{i}{d} \rfloor} \mu(d)$$

$$\sum_{i=1}^n \mu(i) = \sum_{i=1}^n \mu(i) \text{ 将 } \frac{i}{d} \text{ 写成 } i,$$

$$\text{得 } S(n) = (1 - \sum_{i=2}^n S(\lfloor \frac{n}{i} \rfloor)) / g(1)$$

https://blog.csdn.net/weixin\_45697774

## Template A. 【模板】杜教筛 (Sum) (P4213)

给定一个正整数, 求

$$ans1 = \sum_{i=1}^n \varphi(i)$$

$$ans2 = \sum_{i=1}^n \mu(i)$$

Code

```
1 //const int N = 5e6 + 10; // n^2/3
2 const int N = 1665703; // n^2/3
3 inline int read()
```

```

4 {
5     register int x=0,f=1;
6     char c=getchar();
7     while(c<'0' || c>'9') {if(c=='-') f=-1;c=getchar();}
8     while(c>='0'&&c<='9') x=x*10+c-48,c=getchar();
9     return x*f;
10 }
11
12 int cnt;
13 int primes[N + 7];
14 ll mu[N + 7];
15 bool vis[N + 7];
16 ll phi[N + 7];
17 unordered_map<ll, ll> sum_mu; //数组开不了那么大，所以用哈希表
18 unordered_map<ll, ll> sum_phi;
19
20 inline void init(int n)
21 {
22     vis[0] = vis[1] = 1;
23     mu[1] = phi[1] = 1;
24     for (int i = 2; i <= n; ++ i) {
25         if (!vis[i]) {
26             primes[ ++ primes[0]] = i;
27             mu[i] = -1;
28             phi[i] = i - 1;
29         }
30         for (int j = 1; j <= primes[0] && i * primes[j] <= n; ++ j) {
31             vis[i * primes[j]] = 1;
32             if (i % primes[j] == 0) {
33                 mu[i * primes[j]] = 0;
34                 phi[i * primes[j]] = phi[i] * primes[j];
35                 break;
36             }
37             else {
38                 mu[i * primes[j]] = - mu[i];
39                 phi[i * primes[j]] = phi[i] * phi[primes[j]];
40             }
41         }
42     }
43     for (int i = 1; i <= n; ++ i) {
44         mu[i] += mu[i - 1];
45         phi[i] += phi[i - 1];
46     }
47 }
48
49 inline int g_sum(int x) //g的前缀和，这里的g = I(x)//常数函数
50 {
51     return x;
52 }
53
54 inline int get_sum_mu(int x) // 记忆化搜索
55 {
56     if (x <= N) return mu[x]; //预处理
57     //if (sum_mu.find(x) != sum_mu.end()) return sum_mu[x]; //记忆化
58     if(sum_mu[x]) return sum_mu[x];
59     int ans = 1; // 杜教筛中推出的1
60     for (ll l = 2, r; l <= x;l = r + 1) { // 整除分块
61         r = x / (x / l);
62         //Σ_{i=2}^x {g(i)*S(⌊n/i⌋)} g不一样，S一样，然后整除分块
63         ans -= (g_sum(r) - g_sum(l - 1)) * get_sum_mu(x / l);
64     }
65     return sum_mu[x] = ans / g_sum(1); // 最后除以g(1)
66 }
67
68 inline ll get_sum_phi(int x)
69 {
70     if(x <= N) return phi[x];
71     //if(sum_phi.find(x) != sum_phi.end()) return sum_phi[x];
72     if(sum_phi[x]) return sum_phi[x];
73
74     ll ans = x * ((ll)x + 1) / 2; //杜教筛中的 n(n + 1) / 2

```

```
75     for (ll l = 2, r; l <= x; l = r + 1) {
76         r = x / (x / l);
77         ans -= 1ll * (g_sum(r) - g_sum(l - 1)) * get_sum_phi(x / l);
78     }
79     return sum_phi[x] = ans / g_sum(1);
80 }
81
82 int main()
83 {
84     init(N);
85     int t;
86     scanf("%d", &t);
87     while(t -- ) {
88         int n;
89         scanf("%d", &n);
90         printf("%lld %d\n", get_sum_phi(n), get_sum_mu(n));
91     }
92     return 0;
93 }
```

当我们的题目中所给出的积性函数前缀和不太好找到性质使用线性筛  $O(n)$  求解，或者数据过大需要更加优秀的时间复杂度时，我们可以用狄利克雷卷上一个积性函数，使卷完后的式子的前缀和变得简单，卷完后的前缀和算出来基本上题目就完成了。

例如：我们遇见 $\varphi(x)$ ，就想办法推式子往 $\sum_{d|n} \varphi(d)$  上凑。

- 做题时想办法凑出  $\sum_{i=1}^n \sum_{d|i} f(d)$ ，且要求这个式子可以直接求出来。
- 我们转化这个式子 $\sum_{i=1}^n \sum_{j=1}^{\lfloor \frac{n}{i} \rfloor} f(d)$ ，再拆出要求的答案： $\sum_{i=1}^n \sum_{j=1}^{\lfloor \frac{n}{i} \rfloor} f(d) = \sum_{i=2}^n \sum_{j=1}^{\lfloor \frac{n}{i} \rfloor} f(d) + \sum_{j=1}^n f(d)$
- $\sum_{i=2}^n \sum_{j=1}^{\lfloor \frac{n}{i} \rfloor} f(d)$ 这部分往下递归做即可。
- 当  $f(i) = \varphi(i)$  或者  $\mu(i)$  时， $\sum_{d|i} f(i)$  可以直接求出来，但是当  $f(i) = i * u(i)$  时就不行了，我们需要卷积一个 $g(i) = i$ ，此时
$$\sum_{i=1}^n f * g = \sum_{i=1}^n \sum_{d|i} f(d)g(\frac{i}{d}) = \sum_{i=1}^n \sum_{d|i} u(d) = 1。$$

竞赛例题选讲

Problem A. 简单数学题 （Luogu P3768）

由于出题人懒得写背景了，题目还是简单一点好。

输入一个整数  $n$  和一个整数  $p$ ，你需要求出：

$$\left( \sum_{i=1}^n \sum_{j=1}^n ij \gcd(i, j) \right) \bmod p$$

其中  $\gcd(a, b)$  表示  $a$  与  $b$  的最大公约数。

[https://blog.csdn.net/weixin\\_45697774](https://blog.csdn.net/weixin_45697774)

$n \leq 10^{10}$  ,  $5 \times 10^8 \leq p \leq 1.1 \times 10^9$  且  $p$  为质数。

Solution



$$\begin{aligned}
 & \text{计算: } \left( \sum_{i=1}^n \sum_{j=1}^n i \cdot j \cdot \gcd(i, j) \right) \bmod p \\
 &= \sum_{i=1}^n \sum_{j=1}^n i \cdot j \cdot \gcd(i, j) \\
 &= \sum_{d=1}^n d \sum_{i=1}^n \sum_{j=1}^n i \cdot j \cdot [\gcd(i, j) = d] \\
 &= \sum_{d=1}^n d \sum_{i=1}^n \sum_{j=1}^n i \cdot j \cdot [\gcd(\frac{i}{d}, \frac{j}{d}) = 1] \quad \left( i=id, j=jd \text{ (} d=\gcd(i, j) \text{ 是 } i, j \text{ 的倍数)} \right) \\
 &= \sum_{d=1}^n d \sum_{i=1}^{\frac{n}{d}} \sum_{j=1}^{\frac{n}{d}} i \cdot d \cdot j \cdot d \cdot [\gcd(i, j) = 1] \\
 &= \sum_{d=1}^n d^3 \sum_{i=1}^{\frac{n}{d}} \sum_{j=1}^{\frac{n}{d}} i \cdot j \cdot [\gcd(i, j) = 1] \\
 &= \sum_{d=1}^n d^3 \sum_{i=1}^{\frac{n}{d}} \sum_{j=1}^{\frac{n}{d}} i \cdot j \sum_{k|\gcd(i, j)} \mu(k) \\
 &= \sum_{d=1}^n d^3 \sum_{i=1}^{\frac{n}{d}} \sum_{j=1}^{\frac{n}{d}} i \cdot j \sum_{k=1}^{\frac{n}{d}} \mu(k) [k|i] [k|j] \\
 &= \sum_{d=1}^n d^3 \sum_{k=1}^{\frac{n}{d}} \mu(k) \sum_{i=1}^{\frac{n}{dk}} \sum_{j=1}^{\frac{n}{dk}} i \cdot j [k|i] [k|j] \\
 &= \sum_{d=1}^n d^3 \sum_{k=1}^{\frac{n}{d}} \mu(k) \sum_{i=1}^{\frac{n}{dk}} \sum_{j=1}^{\frac{n}{dk}} i \cdot k \cdot j \cdot k \quad \left( i=i \cdot k, j=j \cdot k \text{ (} i, j \text{ 是 } k \text{ 的倍数)} \right) \\
 &= \sum_{d=1}^n d^3 \sum_{k=1}^{\frac{n}{d}} \mu(k) k^2 \sum_{i=1}^{\frac{n}{dk}} \sum_{j=1}^{\frac{n}{dk}} i \cdot j \\
 &= \sum_{d=1}^n d^3 \sum_{k=1}^{\frac{n}{d}} \mu(k) k^2 \left( \sum_{i=1}^{\frac{n}{dk}} i \right)^2
 \end{aligned}$$

设  $T=kd$

$$\begin{aligned}
 &= \sum_{T=1}^n T^2 \sum_{d|T} \mu\left(\frac{T}{d}\right) d \left( \sum_{i=1}^{\frac{n}{T}} i \right)^2 \\
 &= \sum_{T=1}^n T^2 \varphi\left(\frac{T}{d}\right) \left( \sum_{i=1}^{\frac{n}{T}} i \right)^2 \\
 &= \sum_{T=1}^n T^2 \varphi(T) \left( \sum_{i=1}^{\frac{n}{T}} i \right)^2 \\
 &= \sum_{T=1}^n \varphi(T) T^2 \sum_{i=1}^{\frac{n}{T}} i^2
 \end{aligned}$$

$n \leq 10^6$  考虑杜教筛

$$\text{公式: } g(1) S(n) = \sum_{i=1}^n h(i) - \sum_{i=2}^n g(i) S\left(\left\lfloor \frac{n}{i} \right\rfloor\right)$$

$$\begin{aligned}
 & \text{设 } f(x) = x^2 \varphi(x) \quad S(n) = \sum_{i=1}^n f(i) \\
 & h(i) = f * g(i)
 \end{aligned}$$

$$= \sum_{d|i} f(d) g\left(\frac{i}{d}\right)$$

$$= \sum_{d|i} d^2 \varphi(d) \times g\left(\frac{i}{d}\right)$$

$$\text{显然 } \sum_{d|i} \varphi(d) = i \quad (\varphi * 1 = id)$$

$$\text{故设 } g(x) = i^2 \quad \text{则: } h(i) = \sum_{d|i} d^2 \varphi(d) \cdot \frac{i^2}{d^2} = \sum_{d|i} \varphi(d) i^2 = i^3$$

故根据杜教筛公式得:

$$S(n) = \frac{\sum_{i=1}^n h(i) - \sum_{i=2}^n g(i) S\left(\left\lfloor \frac{n}{i} \right\rfloor\right)}{g(1)}$$

$$= \sum_{i=1}^n i^3 - \sum_{i=2}^n i^2 S\left(\left\lfloor \frac{n}{i} \right\rfloor\right)$$

数论分块 ( $i^2$  的区间前缀和)

$$\text{显然 } \sum_{i=1}^n i^3 = 1^3 + 2^3 + \dots + n^3 = (1+2+3+\dots+n)^2 = \frac{n^2(n+1)^2}{4}$$

$$\sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6}$$

注意一点: 这种需要输入模数mod的题, 一定要等输入模数之后再初始化 init!! 不然会RE...膜0可不就RE了嘛...

Time

$$O(n^{\frac{2}{3}})$$

Code

```

1 #include <bits/stdc++.h>
2 #define int long long
3 using namespace std;
4 typedef long long ll;
5

```

https://blog.csdn.net/weixin\_45697774

```

6  const int N = 4641589; // [(10^10)^(2/3)]
7
8  int primes[N + 7], cnt;
9  ll phi[N + 7];
10 bool vis[N + 7];
11 unordered_map<ll, ll> sum_f;
12 int mod, n;
13 int inv4, inv6;
14
15 int qpow(int a, int b)
16 {
17     int res = 1;
18     while(b) {
19         if(b & 1) res = 1ll * res * a % mod;
20         a = 1ll * a * a % mod;
21         b >>= 1;
22     }
23     return res % mod;
24 }
25
26 void init(int n)
27 {
28     phi[1] = 1;
29     for(int i = 2; i <= n; ++ i) {
30         if(vis[i] == 0) {
31             primes[ ++ cnt] = i;
32             phi[i] = i - 1;
33         }
34         for(int j = 1; j <= cnt && i * primes[j] <= n; ++ j) {
35             vis[i * primes[j]] = true;
36             if(i % primes[j] == 0) {
37                 phi[i * primes[j]] = 1ll * phi[i] * primes[j] % mod;
38                 break;
39             }
40             phi[i * primes[j]] = 1ll * phi[i] * phi[primes[j]] % mod;
41         }
42     }
43     for(int i = 1; i <= n; ++ i) {
44         phi[i] = (1ll * phi[i - 1] + (1ll * phi[i] * i % mod * i % mod)) % mod;
45     }
46 }
47
48
49
50 inline int g_sum(int n) //i^2
51 {
52     n %= mod;
53     return 1ll * n * inv6 % mod * (n + 1) % mod * (2 * n + 1) % mod;
54 }
55
56 inline int h_sum(int n) //i^3
57 {
58     n %= mod;
59     return 1ll * n * inv4 % mod * n % mod * (n + 1) % mod * (n + 1) % mod;
60 }
61
62 inline int get_s_sum(int x)
63 {
64     if(x <= N - 7) return phi[x];
65     if(sum_f.find(x) != sum_f.end()) return sum_f[x];
66
67     ll ans = h_sum(x);
68     for(ll l = 2, r; l <= x; l = r + 1) {
69         r = x / (x / l);
70         ans = (ans - 1ll * (g_sum(r) - g_sum(l - 1) + mod) % mod * get_s_sum(x / l) % mod) % mod ;
71     }
72     return sum_f[x] = ans / g_sum(1);
73 }
74
75 void solve(int n)
76 {

```

```
77     ll ans = 0;
78     for(int l = 1, r; l <= n; l = r + 1) {
79         r = n / (n / l);
80         ans = (ans + (1ll * h_sum(n / l) * (get_s_sum(r) - get_s_sum(l - 1) + mod) % mod)) % mod;
81     }
82     printf("%lld\n", ans);
83     return ;
84 }
85
86 signed main()
87 {
88
89     scanf("%lld%lld", &mod, &n);
90     init(N - 7);
91     inv4 = qpow(4, mod - 2);
92     inv6 = qpow(6, mod - 2);
93     solve(n);
94     //cout << "ok" << endl;
95     return 0;
96 }
```

**Problem B. Product** (2019 ACM/ICPC 全国邀请赛 (西安) B)

给定  $n(n \leq 10^9), m(m \leq 2 \times 10^9), p(p \leq 2 \times 10^9)$

计算:

$$\prod_{i=1}^n \prod_{j=1}^n \prod_{k=1}^n m^{\gcd(i,j)[k|\gcd(i,j)]}$$

## Solution

计算:  $\prod_{i=1}^n \prod_{j=1}^n \prod_{k=1}^n m^{\gcd(i,j)[k|\gcd(i,j)]} \bmod p, n \leq 10^9, m \leq 2 \times 10^9, p \leq 2 \times 10^9, p \text{ 是质数}$

显然直接欧拉降幂,  $p$  是质数  $\therefore \gcd(a, p) = 1 \Rightarrow (x^a \bmod p) = x^{a \bmod \varphi(p)} = x^{a \bmod (p-1)}$

$$\left( \prod_{i=1}^n \prod_{j=1}^n \prod_{k=1}^n m^{\gcd(i,j)[k|\gcd(i,j)]} \right) \bmod p$$

$$= m^{\left( \sum_{i=1}^n \sum_{j=1}^n \sum_{k=1}^n \gcd(i,j)[k|\gcd(i,j)] \right) \bmod (p-1)} \quad \text{显然连乘} \rightarrow \text{次幂相加}$$

我们仅需计算次幂中的和式, 最后快速幂计算即可。

$$\text{即: } \sum_{i=1}^n \sum_{j=1}^n \sum_{k=1}^n \gcd(i,j)[k|\gcd(i,j)] \bmod (p-1)$$

考虑其实际意义为: 所有  $k|\gcd(i,j)$  即  $\gcd(i,j)$  为  $k$  的倍数的个数

显然可以交换枚举顺序, 先枚举  $k$ , 再枚举  $k$  的倍数  $\gcd(i,j) = x$ 。

$$\sum_{k=1}^n \sum_{k|x} \sum_{i=1}^n \sum_{j=1}^n \gcd(i,j)[\gcd(i,j) = x]$$

$$\sum_{k=1}^n \sum_{k|x} x \sum_{i=1}^n \sum_{j=1}^n [\gcd(i,j) = x] \quad \gcd(i,j) = x, \text{ 经典套路, 交换位置}$$

$$\sum_{k=1}^n \sum_{k|x} x \sum_{i=1}^{\lfloor \frac{n}{x} \rfloor} \sum_{j=1}^{\lfloor \frac{n}{x} \rfloor} [\gcd(i,j) = 1]$$

分析后面式子的实际含义:  $\sum_{i=1}^{\lfloor \frac{n}{x} \rfloor} \sum_{j=1}^{\lfloor \frac{n}{x} \rfloor} [\gcd(i,j) = 1]$  表示在  $1 \sim \lfloor \frac{n}{x} \rfloor$  中互质的个数, 因为  $\gcd(i,j) = 1$ ,

所以对于任意两数, 例如 1, 3 互质, 3, 1 互质, 注意枚举到的  $i, j$ , 若  $i = j$  则只能算一次, 每次  $i, j$  仅需减一次

$$\text{故 } \sum_{i=1}^{\lfloor \frac{n}{x} \rfloor} \sum_{j=1}^{\lfloor \frac{n}{x} \rfloor} [\gcd(i,j) = 1] = \sum_{i=1}^{\lfloor \frac{n}{x} \rfloor} (2\varphi(i) - 1), n \leq 10^9, \text{ 直接用杜教筛即可}$$

最后仅需计算  $\sum_{k=1}^n \sum_{k|x} x$  即可, 分析其实际意义, 即  $k$  在  $1 \sim n$  中的倍数和

显然对于每对  $k$  和  $n$  均会存在  $\lfloor \frac{n}{k} \rfloor$  个  $k$  的倍数, 显然组成公差为  $k$  的等差数列, 故

$$\sum_{k=1}^n \sum_{k|x} x = \sum_{x=1}^n x d(x) = \sum_{k=1}^n \frac{k(\lfloor \frac{n}{k} \rfloor + 1) \lfloor \frac{n}{k} \rfloor}{2}$$

$$\text{故最后的答案为 } m^b, b = \sum_{k=1}^n \frac{k(\lfloor \frac{n}{k} \rfloor) \lfloor \frac{n}{k} \rfloor}{2} \sum_{i=1}^{\lfloor \frac{n}{k} \rfloor} 2\varphi(i) - \sum_{k=1}^n \frac{k(\lfloor \frac{n}{k} \rfloor + 1) \lfloor \frac{n}{k} \rfloor}{2}$$

使用杜教筛可做到  $O(\sqrt{n})$

## Hint

```
1 res = (res + (111 * (1 + r) * (r - 1 + 1) / 2) % mod * ((111 + x / 1) * (x / 1) / 2) % mod) % mod; //AC
2 res = (res + (111 * (1 + r) * (r - 1 + 1) / 2) % mod * (111 + x / 1) % mod * (x / 1) / 2 % mod) % mod; //WA
```

想想看, ^^ (答案就是我脑子抽了)

## Time

$$O(\sqrt{n} + n^{\frac{2}{3}})$$

## AC Code



```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 const int N = 5e6 + 6;
5 int mod;
6 #define int long long
7 #define mult(x, y) (1ll * x * y >= mod ? 1ll * x * y % mod : 1ll * x * y)
8 #define minus(x, y) (1ll * x - y < 0 ? 1ll * x - y + mod : 1ll * x - y)
9 #define plus(x, y) (1ll * x + y >= mod ? 1ll * x + y - mod : 1ll * x + y)
10 #define ck(x) (x >= mod : x - mod : x)
11 typedef long long ll;
12
13 ll n, m, p;
14
15 ll primes[N], cnt, num[N], d[N];
16 ll phi[N];
17 //ll sum[N]; // x * d(x)
18 bool vis[N];
19
20 unordered_map<ll, ll> M_sum;
21 unordered_map<ll, ll> M_phi;
22
23 void init(ll n)
24 {
25     d[1] = 1;
26     phi[1] = 1;
27     for(ll i = 2; i <= n; ++ i) {
28         if(vis[i] == 0) {
29             primes[ ++ cnt] = i;
30             phi[i] = i - 1;
31             d[i] = 2, num[i] = 1;
32         }
33         for(ll j = 1; j <= cnt && i * primes[j] <= n; ++ j) {
34             vis[i * primes[j]] = true;
35             if(i % primes[j] == 0) {
36                 phi[i * primes[j]] = phi[i] * primes[j];
37                 num[i * primes[j]] = num[i] + 1;
38                 d[i * primes[j]] = (d[i] / num[i * primes[j]] * (num[i * primes[j]] + 1)) % mod;
39                 break;
40             }
41             phi[i * primes[j]] = phi[i] * (primes[j] - 1);
42             num[i * primes[j]] = 1;
43             d[i * primes[j]] = (d[i] * 2) % mod;
44         }
45     }
46
47     for(ll i = 1; i <= n; ++ i) {
48         phi[i] = (phi[i] + phi[i - 1]) % mod;
49         d[i] = d[i] * i % mod;
50         d[i] = (d[i] + d[i - 1]) % mod;
51     }
52 }
53
54 ll qpow(ll a, ll b, ll mod)
55 {
56     ll res = 1;
57     while(b) {
58         if(b & 1) res = res * a % mod;
59         a = a * a % mod;
60         b >>= 1;
61     }
62     return res;
63 }
64
65 inline int g_sum(int x)
66 {
67     return x;
68 }
69
70 inline ll get_sum_phi(int x)
71 {

```

```

72     if(x <= N - 7) return phi[x];
73     if(M_phi[x]) return M_phi[x];
74
75     ll ans = mult(x, (1ll * x + 1) / 2);
76     ll res = 0;
77     for(ll l = 2, r; l <= x; l = r + 1) {
78         r = x / (x / l);
79         res = plus(res, 1ll * (g_sum(r) - g_sum(l - 1)) * get_sum_phi(x / l)) % mod;
80     }
81     return M_phi[x] = minus(ans, res) % mod;
82 }
83
84 inline ll get_sum_sum(ll x)
85 {
86     if(x <= N - 7) return d[x];
87     if(M_sum[x]) return M_sum[x];
88
89     ll res = 0;
90     for(ll l = 1, r; l <= x; l = r + 1) {
91         r = x / (x / l);
92         //sum_k=1^r = 平均值乘上长度 (公差为1的等差数列)
93         res = (res + (1ll * (1 + r) * (r - 1 + 1) / 2) % mod * ((1ll + x / l) * (x / l) / 2) % mod) % mod; //AC
94         //res = (res + (1ll * (1 + r) * (r - 1 + 1) / 2) % mod * (1ll + x / l) % mod * (x / l) / 2 % mod) % mod; //WA
95     }
96     return M_sum[x] = res;
97 }
98
99 signed main()
100 {
101     scanf("%lld%lld%lld", &n, &m, &p);
102     mod = p - 1;
103     init(N - 7);
104     ll ans = 0;
105     for(ll l = 1, r; l <= n; l = r + 1) {
106         r = n / (n / l);
107         ans = plus(ans, 1ll * get_sum_phi(n / l) * minus(get_sum_sum(r), get_sum_sum(l - 1)) % mod) % mod;
108     }
109     ans = plus(ans, ans) % mod;
110     ans = minus(ans, get_sum_sum(n));
111     printf("%lld\n", qpow(m, ans, p) % p);
112     return 0;
113 }

```

### Problem C. Grisaia (2018ACM四川省赛G)

计算:

$$ans = \sum_{i=1}^n \sum_{j=1}^i (n \bmod (i \times j))$$

其中  $T \leq 5, n \leq 10^{11}$

### Solution

使用模的展开式将上述和式展开后, 显然套路枚举  $k = i \times j$ , 由于  $n \leq 10^{11}$ , 杜教筛即可。

筛出:

$$f(x) = x \times d(x)$$

$$g(x) = x \times \mu(x)$$

然后整除分块即可。



$$\begin{aligned} \text{计算: } ans &= \sum_{i=1}^n \sum_{j=1}^i (n \bmod i \times j) \\ &= \sum_{i=1}^n \sum_{j=1}^i (n - \lfloor \frac{n}{ij} \rfloor \times ij) \\ &= n \sum_{i=1}^n \sum_{j=1}^i 1 - \sum_{i=1}^n \sum_{j=1}^i \lfloor \frac{n}{ij} \rfloor \times ij \\ &= n \frac{n(n+1)}{2} - \sum_{i=1}^n \sum_{j=1}^i \lfloor \frac{n}{ij} \rfloor \times ij \end{aligned}$$

$$\sum_{i=1}^n \sum_{j=1}^i \lfloor \frac{n}{ij} \rfloor \times ij$$

设  $ij=k$ , 考虑枚举  $k$

$i \leq n, j \leq i$

若  $i \cdot j > n$ , 则  $\lfloor \frac{n}{ij} \rfloor = 0$

故只需枚举  $k \leq n$  即可

$$\text{即: } \sum_{k=1}^n \lfloor \frac{n}{k} \rfloor \sum_{i=1}^n \sum_{j=1}^i [ij=k] k$$

和式:  $\sum_{i=1}^n \sum_{j=1}^i$  很难处理

考虑统一为:  $\sum_{j=1}^n$

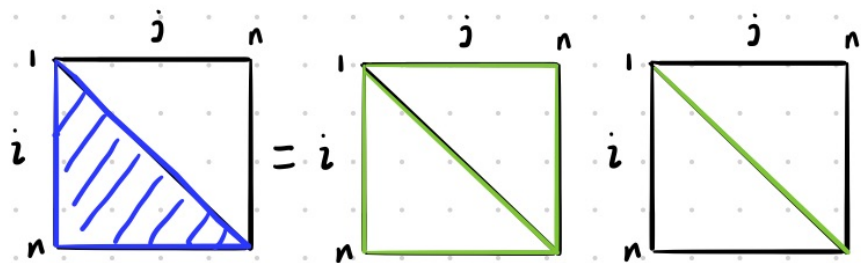
显然有  $\sum_{i=1}^n \sum_{j=1}^i [ij=k] \cdot k$

$$= \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n [ij=k] k + \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n [ij=k] [i=j] k$$

$$= \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n [ij=k] k + \frac{1}{2} \sum_{i=1}^n [i^2=k] i^2$$

$$\begin{aligned} &\sum_{k=1}^n \lfloor \frac{n}{k} \rfloor \sum_{i=1}^n \sum_{j=1}^i [ij=k] k \\ &= \sum_{k=1}^n \lfloor \frac{n}{k} \rfloor \cdot \frac{1}{2} \left( \sum_{i=1}^n \sum_{j=1}^n [ij=k] k + \sum_{i=1}^n [i^2=k] i^2 \right) \end{aligned}$$

$$\begin{aligned} &\sum_{k=1}^n \lfloor \frac{n}{k} \rfloor \left( \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n [ij=k] k \right) \\ &= \frac{1}{2} \sum_{k=1}^n \lfloor \frac{n}{k} \rfloor \sum_{i=1}^n \sum_{j=1}^n [ij=k] k \end{aligned}$$



$$\sum_{i=1}^n \sum_{j=1}^i [ij=k] \cdot k = \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n [ij=k] k + \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n [ij=k] [i=j] k$$

(j可等于i) 对角线的值乘了1/2只算了一半  
应包含全部对角线  $i=j$  的值

加上缺的那一半对角线的值

(可以打表验证)

与前面式子一起整除分块即可

$$\text{即: 设 } f(x) = \sum_{k=1}^x \lfloor \frac{x}{k} \rfloor \sum_{i=1}^x [i^2=k] i^2$$

$$\text{整除分块时前缀和: } \text{sum}(x) = \sum_{k=1}^x \sum_{i=1}^x [i^2=k] i^2$$

$$= \sum_{i=1}^{\sqrt{x}} i^2$$

$$= \frac{\sqrt{x}(\sqrt{x}+1)(2\sqrt{x}+1)}{6}$$

$$\sum_{k=1}^n \left\lfloor \frac{n}{k} \right\rfloor \sum_{i=1}^n \sum_{j=1}^n [ij=k] k \longrightarrow \text{显然 } \sum_{i=1}^n \sum_{j=1}^n [ij=k]$$

$$= \sum_{k=1}^n \left\lfloor \frac{n}{k} \right\rfloor k \cdot d(k)$$

的实际含义为  $i \times j = k$  的  $i, j$  的对数  $\rightarrow k$  的因子数  $\rightarrow \sum_{i|k} 1 = d(k)$

$$\text{设 } f(k) = k d(k)$$

考虑杜教筛, 即构造出  $g(x), h(x)$

$$\text{使得 } h(n) = \sum_{t|n} f(t) g\left(\frac{n}{t}\right)$$

$$\text{根据 } f(t) = t d(t)$$

$$\text{杜教筛公式: } S(n) = \frac{\sum_{i=1}^n h(i) - \sum_{i=2}^n g(i) S\left(\left\lfloor \frac{n}{i} \right\rfloor\right)}{g(1)}$$

我们希望构造出的  $h(i)$  是常数方便计算.

$$\text{显然有 } t \times \frac{n}{t} = n \text{ 为常数}$$

$$\text{故 } g\left(\frac{n}{t}\right) = \frac{n}{t} (\quad)$$

$$\text{显然有 } d * \mu = 1 \Rightarrow \sum_{t|n} d(t) \times \mu\left(\frac{n}{t}\right) = 1$$

$$\text{故 } g(n) = \frac{n}{t} \mu\left(\frac{n}{t}\right)$$

$$\text{则 } h(n) = \sum_{t|n} f(t) g\left(\frac{n}{t}\right)$$

$$= \sum_{t|n} \frac{n}{t} \times \mu\left(\frac{n}{t}\right) \times t \times d(t)$$

$$= \sum_{t|n} n \times \mu\left(\frac{n}{t}\right) \times d(t)$$

$$= n \sum_{t|n} \mu\left(\frac{n}{t}\right) \times d(t)$$

$$= n \times 1$$

$$= n$$

$$S(n) = \frac{\sum_{i=1}^n h(i) - \sum_{i=2}^n g(i) S\left(\left\lfloor \frac{n}{i} \right\rfloor\right)}{g(1)}$$

$$= \frac{n(n+1)}{2} - \sum_{i=2}^n g(i) S\left(\left\lfloor \frac{n}{i} \right\rfloor\right)$$

$$O(Tn^{\frac{2}{3}})$$

$\rightarrow$  考虑杜教筛计算  $g(i)$  的前缀和

设:

$$f(x) = x \cdot \mu(x)$$

$$h(n) = f * g = \sum_{d|n} f(d) g\left(\frac{n}{d}\right)$$

$$\text{同理有 } d \times \frac{n}{d} = n$$

$$\text{故: } g\left(\frac{n}{d}\right) = \frac{n}{d}$$

$$\text{则 } h(n) = \sum_{d|n} f \cdot g$$

$$= \sum_{d|n} d \cdot \mu(d) \cdot \frac{n}{d}$$

$$= \sum_{d|n} \mu(d) n$$

$$= [n=1]$$

$$= 1$$

$$\text{故: } S(n) = \frac{\sum_{i=1}^n h(i) - \sum_{i=2}^n g(i) S\left(\left\lfloor \frac{n}{i} \right\rfloor\right)}{g(1)}$$

$$= 1 - \sum_{i=2}^n i \cdot S\left(\left\lfloor \frac{n}{i} \right\rfloor\right)$$

综上所述:

$$ans = \frac{n(n+1)}{2} \times n - ans2$$

$$ans2 = \frac{1}{2} \sum_{k=1}^{\sqrt{n}} \left\lfloor \frac{n}{k} \right\rfloor \left( f(k) + \frac{\sqrt{k}(\sqrt{k}+1)(\sqrt{k}+2)}{6} \right)$$

$$f(k) = k d(k)$$

$$g(k) = k \mu(k)$$

$$S_1(n) = \sum_{i=1}^n f(i)$$

$$= \frac{n(n+1)}{2} - \sum_{i=2}^n g(i) S\left(\left\lfloor \frac{n}{i} \right\rfloor\right)$$

$$S_2(n) = \sum_{i=1}^n g(i)$$

$$= 1 - \sum_{i=2}^n i S_2\left(\left\lfloor \frac{n}{i} \right\rfloor\right)$$

Hint

注意  $n \leq 10^{11}$ , 中间多处会爆 `long long`, 强转成 `__int128` 即可。

Code

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4 #define int long long
5 #define ll __int128
6 const int N = 31644346;
7
```

https://blog.csdn.net/weixin\_45697774

```

8 int n, m;
9 int mu[N];
10 int primes[N], cnt;
11 int d[N];
12 int num[N];
13 unordered_map<int, ll> sum_mui;
14 unordered_map<int, ll> sum_dk;
15 bool vis[N];
16 int sum[N];
17
18 inline ll read()
19 {
20     register ll x = 0, f = 1;
21     char c = getchar();
22     while(c < '0' || c > '9') {if(c == '-') f = -1; c = getchar();}
23     while(c >= '0' && c <= '9') x = x * 10 + c - 48, c = getchar();
24     return x * f;
25 }
26
27 inline void print(ll x)
28 {
29     if(x < 10)
30     {
31         putchar(x + 48);
32         return;
33     }
34     print(x / 10), print(x % 10);
35 }
36
37 void init(int n)
38 {
39     vis[0] = vis[1] = 1;
40     mu[1] = d[1] = 1;
41     for(int i = 2; i <= n; ++ i) {
42         if(vis[i] == 0) {
43             primes[ ++ cnt] = i;
44             mu[i] = -1;
45             d[i] = 2 * i;
46             num[i] = 1;
47         }
48         for(int j = 1; j <= cnt && i * primes[j] <= n; ++ j) {
49             vis[i * primes[j]] = 1;
50             if(i % primes[j] == 0) {
51                 mu[i * primes[j]] = 0;
52                 num[i * primes[j]] = num[i] + 1;
53                 d[i * primes[j]] = (ll)d[i] / num[i * primes[j]] * (num[i * primes[j]] + 1) * primes[j];
54                 break;
55             }
56             mu[i * primes[j]] -= mu[i];
57             num[i * primes[j]] = 1;
58             d[i * primes[j]] = d[i] * d[primes[j]];
59         }
60     }
61     for(int i = 1; i <= n; ++ i) {
62         sum[i] = sum[i - 1] + mu[i] * i;
63         d[i] = d[i - 1] + d[i];
64     }
65 }
66
67 inline ll get_sum_mui(int x)
68 {
69     if(x <= N - 7) return sum[x];
70     if(sum_mui.find(x) != sum_mui.end()) return sum_mui[x];
71
72     ll ans = 1;
73     for(ll l = 2, r; l <= x; l = r + 1) {
74         r = x / (x / l);
75         ans -= (ll)(r - l + 1) * (1 + r) / 2 * get_sum_mui(x / l);
76     }
77     return sum_mui[x] = ans;
78 }

```

```
79
80 inline ll get_sum_dk(ll x)
81 {
82     if(x <= N - 7) return d[x];
83     if(sum_dk.find(x) != sum_dk.end()) return sum_dk[x];
84     ll ans = x * (x + 1) / 2;
85     for(ll l = 2, r; l <= x; l = r + 1) {
86         r = x / (x / l);
87         ans -= (ll)(get_sum_mui(r) - get_sum_mui(l - 1)) * get_sum_dk(x / l);
88     }
89     return sum_dk[x] = ans;
90 }
91
92 ll cal(ll x)
93 {
94     ll limit = sqrt(x + 0.99);
95     ll more = limit * (limit + 1) * (2 * limit + 1) / 6;
96     return (get_sum_dk(x) + more) / 2;
97 }
98
99 void solve()
100 {
101     ll ans = (ll)n * n * (n + 1) / 2;
102     for(ll l = 1, r; l <= n; l = r + 1) {
103         r = n / (n / l);
104         //cout << "ok" << cal(r) - cal(l - 1) * (n / l) << endl;
105         ans -= (ll)(cal(r) - cal(l - 1)) * (n / l);
106     }
107     print(ans);
108     puts("");
109 }
110
111 signed main()
112 {
113     int t;
114     init(N - 7);
115     t = read();
116     while(t -- ) {
117         n = read();
118         solve();
119     }
120     return 0;
121 }
```

## 0x53 Min\_25筛

Min\_25筛的博客实在是太难写了，所以就直接摘抄OI-Wiki的讲解吧>\_<

author: Marcythm, Xeonacid

link: <https://oiwiki.org/math/number-theory/min-25/>

由于其由 Min\_25 发明并最早开始使用，故称「Min\_25 筛」。

从此种筛法的思想方法来说，其又被称为「Extended Eratosthenes Sieve」。

其可以在  $O\left(\frac{n^{\frac{3}{4}}}{\log n}\right)$  或  $\Theta\left(n^{1-\epsilon}\right)$  的时间复杂度下解决一类 **积性函数** 的前缀和问题。

要求： $f(p)$  是一个关于  $p$  的项数较少的多项式或可以快速求值； $f(p^c)$  可以快速求值。

### 记号

- 如无特别说明，本节中所有记为  $p$  的变量的取值集合均为全体质数。
- $x/y := \left\lfloor \frac{x}{y} \right\rfloor$
- $\text{isprime}(n) := [|\{d : d \mid n\}| = 2]$ ，即  $n$  为质数时其值为 1，否则为 0。
- $p_k$ ：全体质数中第  $k$  小的质数（如： $p_1 = 2, p_2 = 3$ ）。特别地，令  $p_0 = 1$ 。
- $\text{lpf}(n) := [1 < n] \min\{p : p \mid n\} + [1 = n]$ ，即  $n$  的最小质因数。特别地， $n = 1$  时，其值为 1。
- $F_{\text{prime}}(n) := \sum_{2 \leq p \leq n} f(p)$
- $F_k(n) := \sum_{i=2}^n [p_k \leq \text{lpf}(i)] f(i)$

### 具体方法



观察  $F_k(n)$  的定义, 可以发现答案即为  $F_1(n) + f(1) = F_1(n) + 1$ 。

考虑如何求出  $F_k(n)$ 。通过枚举每个  $i$  的最小质因子及其次数可以得到递推式:

$$\begin{aligned} F_k(n) &= \sum_{i=2}^n [p_k \leq \text{lpf}(i)] f(i) \\ &= \sum_{k \leq i} \sum_{\substack{c \geq 1 \\ p_i^c \leq n}} f(p_i^c) ([c > 1] + F_{i+1}(n/p_i^c)) + \sum_{\substack{k \leq i \\ p_i \leq n}} f(p_i) \\ &= \sum_{k \leq i} \sum_{\substack{c \geq 1 \\ p_i^c \leq n}} f(p_i^c) ([c > 1] + F_{i+1}(n/p_i^c)) + F_{\text{prime}}(n) - F_{\text{prime}}(p_{k-1}) \\ &= \sum_{\substack{k \leq i \\ p_i^c \leq n}} \sum_{\substack{c \geq 1 \\ p_i^{c+1} \leq n}} (f(p_i^c) F_{i+1}(n/p_i^c) + f(p_i^{c+1})) + F_{\text{prime}}(n) - F_{\text{prime}}(p_{k-1}) \end{aligned}$$

最后一步推导基于这样一个事实: 对于满足  $p_i^c \leq n < p_i^{c+1}$  的  $c$ , 有  $p_i^{c+1} > n \iff n/p_i^c < p_i < p_{i+1}$ , 故  $F_{i+1}(n/p_i^c) = 0$ 。 其边界值即为  $F_k(n) = 0(p_k > n)$ 。

假设现在已经求出了所有的  $F_{\text{prime}}(n)$ , 那么有两种方式可以求出所有的  $F_k(n)$ :

1. 直接按照递推式计算。
2. 从大到小枚举  $p$  转移, 仅当  $p^2 < n$  时转移增加值不为零, 故按照递推式后缀和优化即可。

现在考虑如何计算  $F_{\text{prime}}(n)$ 。 观察求  $F_k(n)$  的过程, 容易发现  $F_{\text{prime}}$  有且仅有  $1, 2, \dots, \lfloor \sqrt{n} \rfloor, n/\sqrt{n}, \dots, n/2, n$  这  $O(\sqrt{n})$  处的点值是有用的。 一般情况下,  $f(p)$  是一个关于  $p$  的低次多项式, 可以表示为  $f(p) = \sum a_i p^{c_i}$ 。 那么对于每个  $p^{c_i}$ , 其对  $F_{\text{prime}}(n)$  的贡献即为  $a_i \sum_{2 \leq p \leq n} p^{c_i}$ 。 分开考虑每个  $p^{c_i}$  的贡献, 问题就转变为了: 给定  $n, s, g(p) = p^s$ , 对所有的  $m = n/i$ , 求  $\sum_{p \leq m} g(p)$ 。

Notice:  $g(p) = p^s$  是完全积性函数!

于是设  $G_k(n) := \sum_{i=1}^n [p_k < \text{lpf}(i) \vee \text{isprime}(i)] g(i)$ , 即埃筛第  $k$  轮筛完后剩下的数的  $g$  值之和。 对于一个合数  $x$ , 必定有  $\text{lpf}(x) \leq \sqrt{x}$ , 则  $F_{\text{prime}} = G_{\lfloor \sqrt{n} \rfloor}$ , 故只需筛到  $G_{\lfloor \sqrt{n} \rfloor}$  即可。 考虑  $G$  的边界值, 显然为  $G_0(n) = \sum_{i=2}^n g(i)$ 。(还记得吗? 特别约定了  $p_0 = 1$ ) 对于转移, 考虑埃筛的过程, 分开讨论每部分的贡献, 有:

1. 对于  $n < p_k^2$  的部分,  $G$  值不变, 即  $G_k(n) = G_{k-1}(n)$ 。
2. 对于  $p_k^2 \leq n$  的部分, 被筛掉的数必有质因子  $p_k$ , 即  $-g(p_k)G_{k-1}(n/p_k)$ 。
3. 对于第二部分, 由于  $p_k^2 \leq n \iff p_k \leq n/p_k$ , 故会有  $\text{lpf}(i) < p_k$  的  $i$  被减去。这部分应当加回来, 即  $g(p_k)G_{k-1}(p_{k-1})$ 。

则有:

$$G_k(n) = G_{k-1}(n) - [p_k^2 \leq n] g(p_k) (G_{k-1}(n/p_k) - G_{k-1}(p_{k-1}))$$

## 复杂度分析

对于  $F_k(n)$  的计算, 其第一种方法的时间复杂度被证明为  $O(n^{1-\epsilon})$  (见 zzt 集训队论文 2.3); 对于第二种方法, 其本质即为洲阁筛的第二部分, 在洲阁论文中也有提及 (6.5.4), 其时间复杂度被证明为

$$O\left(\frac{n^{\frac{3}{4}}}{\log n}\right)。$$

对于  $F_{\text{prime}}(n)$  的计算, 事实上, 其实现与洲阁筛第一部分是相同的。 考虑对于每个  $m = n/i$ , 只有在枚举满足  $p_k^2 \leq m$  的  $p_k$  转移时会对时间复杂度产生贡献, 则时间复杂度可估计为:

$$\begin{aligned} T(n) &= \sum_{i^2 \leq n} O\left(\pi\left(\sqrt{i}\right)\right) + \sum_{i^2 \leq n} O\left(\pi\left(\sqrt{\frac{n}{i}}\right)\right) \\ &= \sum_{i^2 \leq n} O\left(\frac{\sqrt{i}}{\ln \sqrt{i}}\right) + \sum_{i^2 \leq n} O\left(\frac{\sqrt{\frac{n}{i}}}{\ln \sqrt{\frac{n}{i}}}\right) \\ &= O\left(\int_1^{\sqrt{n}} \frac{\sqrt{\frac{n}{x}}}{\log \sqrt{\frac{n}{x}}} dx\right) \\ &= O\left(\frac{n^{\frac{3}{4}}}{\log n}\right) \end{aligned}$$

对于空间复杂度, 可以发现不论是  $F_k$  还是  $F_{\text{prime}}$ , 其均只在  $n/i$  处取有效点值, 共  $O(\sqrt{n})$  个, 仅记录有效值即可将空间复杂度优化至  $O(\sqrt{n})$ 。

首先, 通过一次数论分块可以得到所有的有效值, 用一个大小为  $O(\sqrt{n})$  的数组 lis 记录。对于有效值  $v$ , 记  $\text{id}(v)$  为  $v$  在 lis 中的下标, 易得: 对于所有有效值  $v$ ,  $\text{id}(v) \leq \sqrt{n}$ 。

然后分开考虑小于等于  $\sqrt{n}$  的有效值和大于  $\sqrt{n}$  的有效值: 对于小于等于  $\sqrt{n}$  的有效值  $v$ , 用一个数组 le 记录其  $\text{id}(v)$ , 即  $\text{le}_v = \text{id}(v)$ ; 对于大于  $\sqrt{n}$  的有效值  $v$ , 用一个数组 ge 记录  $\text{id}(v)$ , 由于  $v$  过大所以借助  $v' = n/v < \sqrt{n}$  记录  $\text{id}(v)$ , 即  $\text{ge}_{v'} = \text{id}(v)$ 。

这样，就可以使用两个大小为  $O(\sqrt{n})$  的数组记录所有有效值的 id 并  $O(1)$  查询。在计算  $F_k$  或  $F_{\text{prime}}$  时，使用有效值的 id 代替有效值作为下标，即可将空间复杂度优化至  $O(\sqrt{n})$ 。

### 有关代码实现

对于  $F_k(n)$  的计算，我们实现时一般选择实现难度较低的第一种方法，其在数据规模较小时往往比第二种方法的表现要好；

对于  $F_{\text{prime}}(n)$  的计算，直接按递推式实现即可。

对于  $p_k^2 \leq n$ ，可以用线性筛预处理出  $s_k := F_{\text{prime}}(p_k)$  来替代  $F_k$  递推式中的  $F_{\text{prime}}(p_{k-1})$ 。相应地， $G$  递推式中的  $G_{k-1}(p_{k-1}) = \sum_{i=1}^{k-1} g(p_i)$  也可以用此方法预处理。

用 Extended Eratosthenes Sieve 求 **积性函数**  $f$  的前缀和时，应当明确以下几点：

- 如何快速（一般是线性时间复杂度）筛出前  $\sqrt{n}$  个  $f$  值；
- $f(p)$  的多项式表示；
- 如何快速求出  $f(p^c)$ 。

明确上述几点之后按顺序实现以下几部分即可：

1. 筛出  $[1, \sqrt{n}]$  内的质数与前  $\sqrt{n}$  个  $f$  值；
2. 对  $f(p)$  多项式表示中的每一项筛出对应的  $G$ ，合并得到  $F_{\text{prime}}$  的所有  $O(\sqrt{n})$  个有用点值；
3. 按照  $F_k$  的递推式实现递归，求出  $F_1(n)$ 。

### 例题

#### 求莫比乌斯函数的前缀和

求  $\sum_{i=1}^n \mu(i)$ 。

易知  $f(p) = -1$ 。则  $g(p) = -1, G_0(n) = \sum_{i=2}^n g(i) = -n + 1$ 。直接筛即可得到  $F_{\text{prime}}$  的所有  $O(\sqrt{n})$  个所需点值。

#### 求欧拉函数的前缀和

求  $\sum_{i=1}^n \varphi(i)$ 。

首先易知  $f(p) = p - 1$ 。对于  $f(p)$  的一次项  $(p)$ ，有  $g(p) = p, G_0(n) = \sum_{i=2}^n g(i) = \frac{(n+2)(n-1)}{2}$ ；对于  $f(p)$  的常数项  $(-1)$ ，有  $g(p) = -1, G_0(n) = \sum_{i=2}^n g(i) = -n + 1$ 。

筛两次加起来即可得到  $F_{\text{prime}}$  的所有  $O(\sqrt{n})$  个所需点值。

### 「LOJ #6053」简单的函数

给定  $f(n)$ ：

$$f(n) = \begin{cases} 1 & n = 1 \\ p \text{ xor } c & n = p^c \\ f(a)f(b) & n = ab \wedge a \perp b \end{cases}$$

易知  $f(p) = p - 1 + 2[p = 2]$ 。则按照筛  $\varphi$  的方法筛，对 2 讨论一下即可。

此处给出一种 C++ 实现：

```
1 #include <algorithm>
2 #include <cmath>
3 #include <cstdio>
4
5 const int maxs = 200000; // 2sqrt(n) const int mod = 1000000007;
6
7 template <typename x_t, typename y_t> inline void inc(x_t &x, const
8 y_t &y) { x += y; (mod <= x) && (x -= mod); } template <typename
9 x_t, typename y_t> inline void dec(x_t &x, const y_t &y) { x -= y;
10 (x < 0) && (x += mod); } template <typename x_t, typename y_t> inline
11 int sum(const x_t &x, const y_t &y) { return x + y < mod ? x + y :
12 (x + y - mod); } template <typename x_t, typename y_t> inline int
13 sub(const x_t &x, const y_t &y) { return x < y ? x - y + mod : (x -
14 y); } template <typename _Tp> inline int div2(const _Tp &x) { return
15 ((x & 1) ? x + mod : x) >> 1; } //以上目的均为防负数和取模 template <typename _Tp>
16 inline long long sqrl1(const _Tp &x) { //平方函数 return (long long)x *
17 x; }
```



```

18
19 int pri[maxs / 7], lpf[maxs + 1], spri[maxs + 1], pcnt;
20
21 inline void sieve(const int &n) { for (int i = 2; i <= n; ++i) {
22     if (lpf[i] == 0) { //记录质数
23         lpf[i] = ++pcnt;
24         pri[lpf[i]] = i;
25         spri[pcnt] = sum(spri[pcnt - 1], i); //前缀和
26     }
27     for (int j = 1, v; j <= lpf[i] && (v = i * pri[j]) <= n; ++j) lpf[v] = j; } }
28
29 long long global_n; int lim; int le[maxs + 1], // x <= \sqrt{n}
30     ge[maxs + 1]; // x > \sqrt{n}
31 #define idx(v) (v <= lim ? le[v] : ge[global_n / v])
32
33 int G[maxs + 1][2], Fprime[maxs + 1]; long long lis[maxs + 1]; int
34 cnt;
35
36 inline void init(const long long &n) { for (long long i = 1, j, v; i
37 <= n; i = n / j + 1) {
38     j = n / i;
39     v = j % mod;
40     lis[++cnt] = j;
41     (j <= lim ? le[j] : ge[global_n / j]) = cnt;
42     G[cnt][0] = sub(v, 111);
43     G[cnt][1] = div2((long long)(v + 211) * (v - 111) % mod); } }
44
45 inline void calcFprime() { for (int k = 1; k <= pcnt; ++k) {
46     const int p = pri[k];
47     const long long sqrp = sqrrl(p);
48     for (int i = 1; lis[i] >= sqrp; ++i) {
49         const long long v = lis[i] / p;
50         const int id = idx(v);
51         dec(G[i][0], sub(G[id][0], k - 1));
52         dec(G[i][1], (long long)p * sub(G[id][1], spri[k - 1]) % mod);
53     } } /* F_prime = G_1 - G_0 */ for (int i = 1; i <= cnt; ++i) Fprime[i] = sub(G[i][1], G[i][0]); }
54
55 inline int f_p(const int &p, const int &c) { /* f(p^{c}) = p xor c
56 */ return p xor c; }
57
58 int F(const int &k, const long long &n) { if (n < pri[k] || n <= 1)
59 return 0; const int id = idx(n); long long ans = Fprime[id] -
60 (spri[k - 1] - (k - 1)); if (k == 1) ans += 2; for (int i = k; i
61 <= pcnt && sqrrl(pri[i]) <= n; ++i) {
62     long long pw = pri[i], pw2 = sqrrl(pw);
63     for (int c = 1; pw2 <= n; ++c, pw = pw2, pw2 *= pri[i])
64         ans +=
65             ((long long)f_p(pri[i], c) * F(i + 1, n / pw) + f_p(pri[i], c + 1)) %
66             mod; } return ans % mod; }
67
68 int main() { scanf("%lld", &global_n); lim = sqrt(global_n); //上限
69
70 sieve(lim + 1000); //预处理 init(global_n); calcFprime();
71 printf("%lld\n", (F(1, global_n) + 111 + mod) % mod);
72
73 return 0; } ``

```

## Ox54 洲阁筛

author: Early0v0

link: <https://oiwiki.org/math/number-theory/zhou/>

### 定义

洲阁筛是一种能在亚线性时间复杂度内求出大多数积性函数前缀和的筛法。

下面将以求解  $\sum_{i=1}^n f(i)$  为例，具体阐述洲阁筛的原理。

## 约定

- $\mathbb{P}$  表示质数集,  $p_i$  表示第  $i$  个质数。
- $m$  表示  $\sqrt{n}$  内的质数个数。

## 要求

当  $p \in \mathbb{P}, c \in \mathbb{N}$  时,  $f(p^c)$  为一个关于  $p$  的低阶多项式。

## 思想

- 对于任意  $[1, n]$  内的整数, 其至多只有一个  $> \sqrt{n}$  的质因子。
- 利用  $\left\lfloor \frac{n}{i} \right\rfloor (i \in [1, n] \cap \mathbb{N})$  只有  $\sqrt{n}$  级别个取值的性质来降低时间复杂度。

## 思路

将  $[1, n]$  内的所有整数按是否有  $> \sqrt{n}$  的质因子分为两类:

$$\sum_{i=1}^n f(i) = \sum_{i=1}^n [\exists d \in (\sqrt{n}, n] \cap \mathbb{P}, d \mid i] f(i) + \sum_{i=1}^n [\forall d \in (\sqrt{n}, n] \cap \mathbb{P}, d \nmid i] f(i)$$

对于前半部分, 枚举最大因子, 根据积性函数的性质可以转换:

$$\sum_{i=1}^n f(i) = \sum_{i=1}^{\sqrt{n}} f(i) \cdot \left( \sum_{d=\lfloor \sqrt{n} \rfloor + 1}^{\lfloor \frac{n}{i} \rfloor} [d \in \mathbb{P}] f(d) \right) + \sum_{i=1}^n [\forall d \in (\sqrt{n}, n] \cap \mathbb{P}, d \nmid i] f(i)$$

前后部分可以分别计算。

## Part 1

计算  $\sum_{i=1}^{\sqrt{n}} f(i) \cdot \left( \sum_{d=\lfloor \sqrt{n} \rfloor + 1}^{\lfloor \frac{n}{i} \rfloor} [d \in \mathbb{P}] f(d) \right)$ 。

考虑枚举  $i$ , 然后  $\mathcal{O}(1)$  计算括号内部分。

记  $g(t, l) = \sum_{i=1}^l [\forall j \in [1, t], \gcd(i, p_j) = 1] f(i)$ , 即  $[1, l]$  中与  $p_1, p_2, \dots, p_t$  均互质的数的  $f$  值之和。

这样 Part 1 的计算就变成了  $\sum_{i=1}^{\sqrt{n}} f(i) \cdot g\left(m, \left\lfloor \frac{n}{i} \right\rfloor\right)$ 。

边界  $g(0, l) = \sum_{i=1}^l f(i)$ , 转移  $g(t, l) = g(t-1, l) - f(p_t) \cdot g\left(t-1, \left\lfloor \frac{l}{p_t} \right\rfloor\right)$ 。

$l$  共有  $\sqrt{n}$  级别种取值, 对于每种取值则需要枚举其质因子, 所以复杂度为  $\mathcal{O}\left(\frac{\sqrt{n}}{\ln \sqrt{n}} \cdot \sqrt{n}\right) = \mathcal{O}\left(\frac{n}{\log n}\right)$ , 需要优化。

注意到  $p_{t+1}^2 > l$  时符合条件的数只有 1, 所以此时  $g(t, l) = f(1) = 1$ 。

代入递推式可得: 当  $p_t^2 > l$  时,  $g(t, l) = g(t-1, l) - f(p_t)$ 。

所以一旦发现  $p_t^2 > l$  就停止转移, 记此时的  $t$  为  $t_l$ , 则  $\forall t > t_l, g(t, l) = g(t_l, l) - \sum_{i=t_l}^{t-1} f(p_i)$ 。

预处理质数的  $f$  值前缀和即可快速求出  $g$ , 时间复杂度被优化至  $\mathcal{O}\left(\frac{n^{\frac{3}{4}}}{\log n}\right)$ 。

## Part 2

计算  $\sum_{i=1}^n [\forall d \in (\sqrt{n}, n] \cap \mathbb{P}, d \nmid i] f(i)$ 。

记  $h(t, l) = \sum_{i=1}^l \left[ i = \prod_{j=t}^m p_j^{c_j}, c_j \in \mathbb{N} \right] f(i)$ , 即  $[1, l]$  中所有只含  $p_t, p_{t+1}, \dots, p_m$  质因子的数的  $f$  值之和。

Part 2 即为求  $h(0, n)$ 。

边界  $h(m+1, l) = 1$ , 转移  $h(t, l) = h(t+1, l) + \sum_{c \in \mathbb{N}^*} f(p_t^c) \cdot h\left(t+1, \left\lfloor \frac{l}{p_t^c} \right\rfloor\right)$ 。

$l$  共有  $\sqrt{n}$  级别种取值, 所以直接转移复杂度为  $\mathcal{O}\left(\sqrt{n} \cdot \frac{\sqrt{n}}{\ln \sqrt{n}}\right) = \mathcal{O}\left(\frac{n}{\log n}\right)$ , 需要优化。

与  $g$  的优化方式类似, 注意到  $p_t > l$  时, 能用  $p_t, p_{t+1}, \dots, p_m$  组成的数只有 1, 此时的  $h(t, l) = f(1) = 1$ 。

类似的, 推出  $\forall p_t^2 > l, h(t, l) = h(t-1, l) + f(p_t)$ 。

所以一旦发现  $p_t^2 > l$  就停止转移，记此时的  $t$  为  $t_l$ ，之后用到  $h$  时，把此时的  $h$  值加上  $\sum_{i=p_{t_l}}^{\min(l,\sqrt{n})} [i \in \mathbb{P}] f(i)$  即可。

时间复杂度被优化至  $\mathcal{O}\left(\frac{n^{\frac{3}{4}}}{\log n}\right)$ 。

求和

算出了 Part 1 和 Part 2 的答案，将其相加即为  $\sum_{i=1}^n f(i)$ 。

参考

积性函数线性筛/杜教筛/洲阁筛学习笔记 | Bill Yang's Blog

ox60 原根

本章只简单涉及了一些原根的性质和应用，具体原根更详细的性质证明详见：[https://oiwiki.org/math/number-theory/primitive-root/#\\_7](https://oiwiki.org/math/number-theory/primitive-root/#_7)

ox61 整数的阶、原根与指标

ox61.1 整数的阶

**定义：** 若 $a, p$ 为正整数, 且  $gcd(a, p) = 1$  , 称满足  $a^r \equiv 1 \pmod{p}$  的**最小正整数**  $r$  为  $a$  **模  $p$  的阶** , 记作 $ord_p a$ 。

我们可以将整数的阶理解为  $a \% p$  下的阶  $t$  相当于  $a \% p$  的循环节，每次乘上  $a^t$  , 值都不变。

- **基本性质**

**性质61.1.1：** 由欧拉定理  $a^{\varphi(p)} \equiv 1 \pmod{p}$ , 故可以得到 阶  $t \mid \varphi(p)$ 。

**性质61.1.2：** 如果  $a$  是  $p$  互素的整且  $p > 0$  , 则正整数  $x$  是同余式  $a^x \equiv 1 \pmod{p}$  的一个解当且仅当  $ord_p a \mid x$ 。

**性质61.1.3：** 如果  $a, n$  是互素的整数且  $n > 0$  , 那么  $a^i \equiv a^j \pmod{p}$ , 当且仅当  $i \equiv j \pmod{ord_p a}$ , 其中  $i$  和  $j$  是非负整数。

因为  $a$  的阶  $r, a^r \equiv 1 \pmod{p}, a^i \equiv a^j \pmod{p}$  两边同余两边一直除以  $a^r$  , 相当于同时减去  $r$  , 所以  $i \equiv j \pmod{ord_p a}$  的正确性显然。

**性质61.1.4：** 设  $a$  关于模  $p$  的阶为  $r$ , 则  $a^0, a^1, a^2, \dots, a^{r-1}$  互不相同。

ox61.2 整数的原根

**定义：** 若  $Ord_p(a) = \varphi(p)$  , 则称这样的  $a$  为**模数  $p$  的原根**, 记作 $Rt(p) = a$ 。

**原根就是特殊情况**的阶。

**理解：** 对于模数  $p$  , 找到一些数  $a$ , 满足  $a$  的次方**循环节**恰好为 $\varphi(p)$ , 这些数就是模数  $p$  的原根

- **基本性质定理：**

**定理61.2.1：** 如果  $r$  和  $n$  互质且 $n > 0$ , 则如果  $r$  是模  $n$  的一个原根，那么下列整数：

$a, a^2 \dots a^{\varphi(n)}$

构成了模  $n$  的既约剩余系（简化剩余系）并且有 $p - 1$ 个，也就是恰好为 $1 \dots p - 1$ 中与  $p$  互质的数的一个排列，对应着欧拉函数的定义。

**即可以得到：若  $p$  为素数，对于模  $p$  的原根  $g$  , 满足  $g^1, g^2, \dots, g^{p-1}$  , 在模  $p$  的意义下，互不相同。**（这点非常重要，引申出了 **0x64.1的乘法换加法算法**）

**性质61.2.2：** 若一个正整数  $n$  有原根，则其有  $\varphi(\varphi(n))$  个原根 。

**性质61.2.3：**  $x^d \equiv 1 \pmod{p}$  恰好有  $d$  个解 ( $d \mid (p - 1)$ )

**性质61.2.3：** 如果  $ord_p a = t$  并且  $u$  是一个正整数，那么有  $ord_p(a^u) = \frac{t}{gcd(t, u)}$ 。

**性质61.2.2证明：** 如果  $a$  是模  $m$  的原根，那么对于任意和  $\varphi(m)$  互质的正整数  $s$  ,  $a^s$  也是模  $m$  的原根（此时  $s, 2s, \dots \varphi(m) \cdot s$  在模  $\varphi(m)$  下互不相同，于是 $a^s, a^{2s}, \dots a^{\varphi(m) \cdot s}$  在模  $m$  下互不相同）。容易证明，它们就是  $m$  的全部原根（注意到模  $m$  下与  $m$  互质的数一共只有  $\varphi(m)$  个，已经被  $a^1, a^2, \dots a^{\varphi(m)}$  占据完了）。所以原根的数量就是模  $m$  意义下  $s$  的数量，即  $\varphi(\varphi(m))$  个。

**推论61.2.4：** 恒等式,  $a^{\frac{\varphi(xy)}{2}} \equiv 1 \pmod{xy}$  ( $x, y$ 互素且 $a$ 与 $xy$ 互素)

在2次幂的时候就有 $a^{2^k} \equiv 1 \pmod{2k+2}$  恒成立，所以 2 的 3 次及以上次幂没有原根

5 模  $2^{k+2}$  的指数正好是  $2^k$ 。

### • 求解原根

验证一个数 $a$ 是否为  $Rt(P)$ ，我们可以使用试除法。显然  $a^{\varphi(P)} \equiv 1$ ，那么我们现在只需要证明**最小循环节**为  $\varphi(P)$  即可，而如果存在更小的循环节，则一定为  $\varphi(P)$  的因子。

我们知道一个数  $n$  的原根最多只有  $\varphi(\varphi(n))$  个，我们知道 $\varphi(n) \leq \sqrt{n}$ ，这也就意味着我们在求原根的时候可以直接暴力枚举，时间复杂度也只有 $O(n^{\frac{1}{4}})$ 。

那么预处理出  $\varphi(p)$  的所有的质因数 $(p_1, p_2 \dots p_k)$ ，暴力判断一下，如果  $\exists i, a^{\frac{\varphi(P)}{p_i}} \equiv 1 \pmod{P-1}$

说明存在更小的循环节，也就说明  $a$  不是模  $P$  的原根，因为根据原根的定义，需要保证是第一个满足  $\varphi(P)$  ( $a^{P-1} \equiv 1 \pmod{P-1}$ )的数。

### Code

```
1 ll n, m;
2 vector<int>factor;
3 vector<int> get_factor(ll n)
4 {
5     vector<int>res;
6     for(int i = 1; i * i <= n; ++ i) {
7         if(n % i == 0) {
8             res.push_back(i);
9             if(i != n / i) {
10                 res.push_back(n / i);
11             }
12         }
13     }
14     return res;
15 }
16 int qpow(int a, int b, int c)
17 {
18     int res = 1;
19     while(b){
20         if(b & 1) res = ((ll)res * a) % c;
21         a = ((ll)a * a) % c;
22         b >>= 1;
23     }
24     return res % c;
25 }
26 int main()
27 {
28     scanf("%lld", &n);
29     ll phi_m = n - 1;
30     factor = get_factor(phi_m);
31     for(int i = 2; i < n; ++ i) {
32         bool flag = 1;
33         for(int j = 0; j < factor.size(); ++ j) {
34             if(factor[j] != phi_m && qpow(i, factor[j], n) == 1) {
35                 flag = 0;
36                 break;
37             }
38         }
39         if(flag) {
40             printf("%d\n", i);
41             break;
42         }
43     }
44     return 0;
45 }
46
```

其他的原根可用最小原根的 $g^u$ 表示 ( $g^u$ 是  $p$  的原根当且仅当 $\gcd(u, \varphi(p)) = 1$ )

### 找到所有的原根

```
1 const int N = 1000010;
2 int phi[N], p[N], a[N], ans[N], n, r, t;
3 bool v[N];
4 int gcd(int a, int b) {
5     return (b == 0) ? a : gcd(b, a % b);
6 }
7 ll qpow(ll x, int y, int mo) {
8     ll s = 1;
```

```

9     while (y) {
10         if (y & 1)
11             s = s * x % mo;
12         x = x * x % mo;
13         y >>= 1;
14     }
15     return s;
16 }
17 int check(int n) {
18     if (n % 2 == 0)
19         return -1;
20     if (!v[n])
21         return n;
22     for (int i = 2; p[i]*p[i] <= n; i++)
23         if (n % p[i] == 0) {
24             while (n % p[i] == 0)
25                 n /= p[i];
26             if (n > 1)
27                 return -1;
28             else
29                 return p[i];
30         }
31     return -1;
32 }
33 bool solve(int g, int mo) {
34     if (gcd(g, mo) != 1)
35         return 0;
36     for (int i = 1; i <= r; i++)
37         if (qpow(g, phi[mo] / a[i], mo) == 1)
38             return 0;
39     return 1;
40 }
41 int getg(int mo) {
42     int n = phi[mo];
43     r = 0;
44     for (int i = 1; p[i]*p[i] <= n; i++)
45         if (n % p[i] == 0) {
46             while (n % p[i] == 0)
47                 n /= p[i];
48             a[++r] = p[i];
49         }
50     if (n > 1)
51         a[++r] = n;
52     for (int i = 2; i < mo; i++)
53         if (solve(i, mo))
54             return i;
55 }
56 int main() {
57     phi[1] = 1;
58     for (int i = 2; i <= 1000000; i++) {
59         if (!v[i]) {
60             phi[i] = i - 1;
61             p[++t] = i;
62         }
63         for (int j = 1; j <= t; j++) {
64             if (i * p[j] > 1000000)
65                 break;
66             v[i * p[j]] = 1;
67             if (i % p[j] == 0) {
68                 phi[i * p[j]] = phi[i] * p[j];
69                 break;
70             } else
71                 phi[i * p[j]] = phi[i] * (p[j] - 1);
72         }
73     }
74     while (scanf("%d", &n) != EOF) {
75         if (n == 2) {
76             puts("1");
77             continue;
78         }
79         if (n == 4) {

```

```
80         puts("3");
81         continue;
82     }
83     int ch = (n % 2 == 1) ? check(n) : check(n / 2);
84     if (ch == -1) {
85         puts("-1");
86         continue;
87     }
88     int g = getg(n);
89     int l = 0;
90     for (int i = 1; i <= phi[n]; i++)
91         if (gcd(i, phi[n]) == 1)
92             ans[++l] = qpow(g, i, n);
93     sort(ans + 1, ans + 1 + l);
94     for (int i = 1; i <= l; i++) {
95         printf("%d", ans[i]);
96         if (i == l)
97             puts("");
98         else
99             printf(" ");
100     }
101 }
102 }
```

0x61.3 整数的指标（也称指数、离散对数）

**定义：** 设  $r$  为模  $P$  的原根，有  $r^x \equiv n \pmod{p}, (1 \leq x \leq \varphi(m))$  成立的唯一整数  $x$ ，则称  $x$  为以  $r$  为底， $n$  的离散对数，记作： $x = ind_r(n)$  我们简记为  $x = I_r(n)$

我们发现这个方程实际上就是我们前面讲过的第一种高次同余方程  $a^x \equiv b \pmod{p}$ 。

我们发现指标与数学中的对数拥有很多相似的性质，只需要将等式用模  $\varphi(m)$  的同余式来代替即可。

● 基本性质定理

**性质61.3.1：**  $a \equiv b \pmod{p} \iff I_r(a) \equiv I_r(b) \pmod{\varphi(p)}$

**性质61.3.2：** 指标类似对数，可以进行乘法与加法的转换： $I_r(ab) \equiv I_r(a) + I_r(b) \pmod{\varphi(p)}$

**性质61.3.2：** 类似对数， $I_r(a^k) \equiv k \times I_r(a) \pmod{\varphi(p)}$

● 求解

求整数的指标实际上就是求出  $r^x \equiv n \pmod{p}$ 。我们可以先用枚举法求出  $p$  的原根  $r$ ，再用 BSGS 算法求出以  $r$  为底的  $n$  的离散对数  $x$  即可。

0x62 素数的原根

**定理62.1.1：** 每个素数都有原根。

**定理62.1.2：** 若  $g$  是某一奇素数  $p$  的原根，那么  $g$  也是  $p^k$  的原根

这是一条非常有用的定理。

0x62.1 多项式同余

假设  $f(x)$  是一个整数系数多项式，称整数  $c$  是  $f(x)$  模  $m$  的根是指  $f(c) \equiv 0 \pmod{m}$ 。

易知，如果  $c$  是  $f(x)$  模  $m$  的根，那么每一个模  $m$  同余于与  $c$  的整数也是一个根。

**定理62.1.1（拉格朗日定理）：** 假设  $f(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$  是一个次数为  $n$ ，首项系数  $a_n$ ，不能被  $p$  整除的整系数多项式，且  $n \geq 1$ ，那么  $f(x)$  至多只有  $n$  个模  $p$  不同余的根。

**定理62.1.2：** 假设  $p$  为素数，且  $d$  是  $p - 1$  的因子，那么多项式  $x^d - 1$  恰有  $d$  个模  $p$  不同余的根。

**定理62.1.3：** 假设  $p$  为素数，且  $d$  是  $p - 1$  的正因子，那么比  $p$  小且模  $p$  的阶为  $d$  的正整数个数不超过  $\varphi(d)$  个。

**定理62.1.3：** 假设  $p$  为素数，且  $d$  是  $p - 1$  的正因子，那么模  $p$  的阶为  $d$  且不同余的整数个数为  $\varphi(d)$  个。

**推论62.1.4：** 每个素数都有原根。

0x63 原根的存在性

**定理63.1.1：** 如果  $p$  是一个素数，且有原根  $r$ ，那么  $r$  或  $r + p$  是模  $p^2$  的一个原根。

**性质63.1.2：** 根据 性质61.2.2 可知，一个模数  $m$  有原根的充要条件是  $m = 1, 2, 4, p^C, 2p^C$  (p is prime,  $C \in \mathbb{Z}^*$ )



## 0x64 原根的应用

### 0x64.1乘法换加法（取模意义下）

我们根据原根的**性质61.2.1**：若  $p$  为素数，对于模  $p$  的原根  $g$ ，满足  $g^1, g^2, \dots, g^{p-1}$ ，在模  $p$  的意义下，互不相同。可得：任何一个小于正整数  $m$  的正整数都可以表示为原根的某次幂，那么求一个由小于  $m$  正整数组成的数列的累乘值可以转化为原根的指数的求和，这样可以不用计算高精度，且由周期性还可以缩小指数的范围。

#### Problem A （[题目链接](#)）

小C有一个集合  $S$ ，里面的元素都是小于  $M$  的非负整数。有一个数列生成器，可以生成一个长度为  $N$  的数列，数列中的每个数都属于集合  $S$ 。他用这个生成器生成了许多这样的数列。问给定整数  $x$ ，求所有可以生成出的，且满足数列中所有数的乘积  $\bmod M$  的值等于  $x$  的不同的数列的有多少个。认为，两个数列  $\{A_i\}$  和  $\{B_i\}$  不同，当且仅当至少存在一个整数  $i$ ，满足  $A_i \neq B_i$ 。输出答案  $\bmod 1004535809$ 。（ $1 \leq N \leq 10^9, 3 \leq M \leq 8000$ ， $M$  为质数， $1 \leq x \leq M - 1$ ，输入数据保证集合  $S$  中元素不重复）

#### Solution

对于本题我们需要求出数列中所有元素的乘积，我们可以应用乘法转加法得到数列乘积的原根指数，再将  $x$  处理成原根的某次幂，根据阶的**性质61.1.3** 即可判定是否同余。

更详细的题解：<https://www.luogu.com.cn/blog/ZigZagKmp/solution-p3321>

#### Code

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4 #define int long long
5 using ll = long long;
6 const int N = 2e6 + 7, mod = 1004535809, G = 3;
7
8 ll a[N], b[N], h[N];
9 int n, m, X, S, GG;
10 int limit, RR[N], L;
11 ll g[N], f[N], dtol[N], ltod[N];
12 vector<int> factor;
13
14 int qpow(int a, int b, int mod = 1004535809)
15 {
16     int res = 1;
17     while(b){
18         if(b & 1) res = 1ll * res * a % mod;
19         a = 1ll * a * a % mod;
20         b >>= 1;
21     }
22     return res;
23 }
24
25 vector<int> get_factor(ll n)
26 {
27     vector<int> res;
28     for(int i = 1; i * i <= n; ++ i) {
29         if(n % i == 0) {
30             res.push_back(i);
31             if(i != n / i) {
32                 res.push_back(n / i);
33             }
34         }
35     }
36     return res;
37 }
38
39 void get_RT(int n)
40 {
41     ll phi_m = n - 1;
42     factor = get_factor(phi_m);
43     for(int i = 2; i < n; ++ i) {
44         bool flag = 1;
45         for(int j = 0; j < (int) factor.size(); ++ j) {
46             if(factor[j] != phi_m && qpow(i, factor[j], n) == 1) {
47                 flag = 0;
48                 break;
49             }
50         }
51         if(flag) RR[i] = i;
52     }
53 }
```

```

49     }
50 }
51 if(flag) {
52     GG = i;
53     break;
54 }
55 }
56 int tmp = 1;
57 for(int i = 0; i < phi_m; ++ i, tmp = 1ll * tmp * GG % n)
58     dtol[tmp] = i, ltod[i] = tmp;
59 return ;
60 }
61
62 ll inv(ll x) {return qpow(x, mod - 2);}
63
64 void NTT(ll *A, int type = 1)
65 {
66     for(int i = 0; i < limit; ++ i)
67         if(i < RR[i])
68             swap(A[i], A[RR[i]]);
69     for(int mid = 1; mid < limit; mid <= 1) {
70         ll wn = qpow(G, (mod - 1) / (mid * 2));
71         if(type == -1) wn = qpow(wn, mod - 2);
72         for(int len = mid << 1, pos = 0; pos < limit; pos += len) {
73             ll w = 1;
74             for(int k = 0; k < mid; ++ k, w = (w * wn) % mod) {
75                 int x = A[pos + k], y = w * A[pos + mid + k] % mod;
76                 A[pos + k] = (x + y) % mod;
77                 A[pos + k + mid] = (x - y + mod) % mod;
78             }
79         }
80     }
81 }
82
83 if(type == -1) {
84     ll limit_inv = inv(limit);
85     for(int i = 0; i < limit; ++ i)
86         A[i] = (A[i] * limit_inv) % mod;
87 }
88 }
89
90 void mul(ll *f, ll *g, ll *ans, int n, int m, int mod_len)
91 {
92     limit = 1, L = 0;
93     while(limit < n + m - 1) limit <= 1, ++ L;
94     for(int i = 0; i < limit; ++ i)
95         RR[i] = (RR[i] >> 1) >> 1 | ((i & 1) << (L - 1));
96
97     for(int i = 0; i < n; ++ i)
98         a[i] = f[i];
99     for(int i = n; i < limit; ++ i)
100         a[i] = 0;
101     for(int i = 0; i < m; ++ i)
102         b[i] = g[i];
103     for(int i = m; i < limit; ++ i)
104         b[i] = 0;
105
106     NTT(a), NTT(b);
107
108     for(int i = 0; i < limit; ++ i) {
109         a[i] = 1ll * a[i] * b[i] % mod;
110     }
111     NTT(a, -1);
112
113     for(int i = 0; i < mod_len; ++ i)
114         ans[i] = a[i];
115     for(int i = mod_len; i < limit; ++ i)
116         ans[i % mod_len] = (ans[i % mod_len] + a[i]) % mod;
117     for(int i = mod_len; i < limit; ++ i)
118         ans[i] = 0;
119 }

```

```
120
121 signed main()
122 {
123     scanf("%lld%lld%lld%lld", &n, &m, &X, &S);
124     get_RT(m);
125     for(int i = 1; i <= S; ++ i) {
126         int x;
127         scanf("%lld", &x);
128         if(x != 0) {
129             f[dtol[x]] ++ ;
130         }
131     }
132     g[0] = 1;
133     while(n) {
134         if(n & 1) mul(f, g, g, m - 1, m - 1, m - 1);
135         mul(f, f, f, m - 1, m - 1, m - 1);
136         n >>= 1;
137     }
138
139     printf("%lld\n", g[dtol[X]]);
140     return 0;
141 }
```

0x64.2 快速数论变换

快速数论变换实际上是

我们发现FFT里的大多操作，都跟单位根没有什么关系，（例如选点插值，奇偶分治），我们随便选择一个  $n$  个点 $v, v^2, v^3, \dots, v^N$ ，最后可以得到公式  $A(k) = A1(x) + V^k \times A2(k)$ ，现在问题是这个公式只有在  $k < \frac{N}{2}$  的时候才能成立，我们需要找到  $k \geq \frac{N}{2}$  的时候的答案，这样才能得到整体的答案。所以FFT的关键问题就是如何得到  $k \geq \frac{N}{2}$  时的统一的递推式，进而可以把两个子问题合并，使用分治的思想 $O(nlogn)$ 的时间完成整个过程。

我们之所以使用单位根，即令 $v = \omega_N$ ，也就是 $e^{\frac{2i\pi}{N}}$ ，就是 因为单位根拥有四个重要性质：

- $\omega_N^N = 1$
- $\omega_N^{\frac{N}{2}} = -1$
- $\omega_{2N}^{2k} = \omega_N^k$ （折半定理）
- $\omega_N^{k+\frac{N}{2}} = -\omega_N^k$ （消去定理）

我们根据性质1和3，可以得到 $k < \frac{N}{2}$ 的时候， $A1(\omega_N^k) = A1(\omega_{\frac{N}{2}}^k) + \omega_N^k \times A2(\omega_{\frac{N}{2}}^k)$

根据性质2和4，可以得到 $k \geq \frac{N}{2}$ 的时候， $A1(\omega_N^k) = A1(\omega_{\frac{N}{2}}^k) - \omega_N^k \times A2(\omega_{\frac{N}{2}}^k)$

（上面的证明请参考我的[快速傅里叶变换](#)）

那么  $\omega_N$  涉及到复数，可能会出现精度误差，那么有没有同样拥有上述性质的  $n$  个数  $v$ ，并且满足互不相等呢？

我们做数学题经常会用到取模运算，所以我们考虑模**质数**  $p$  意义下的一个神奇的东西：原根。上面我们讲述了原根的性质，所以我们可以来证明一下它是否具有和 $\omega$ 相同的性质。

我们设  $g$  是  $p$  的原根，令  $g_N = g^{\frac{p-1}{N}}$ ，其中要求 $p - 1$  能被  $N$  整除

我们参照  $\omega$  考虑证明下面同样的四个性质。

- $g_N^N \equiv 1 \pmod{p}$
- $g_N^{\frac{N}{2}} \equiv -1 \pmod{p}$
- $g_{2N}^{2k} \equiv g_N^k \pmod{p}$
- $g_N^{\frac{N}{2}} = g^{p-1} \equiv -1 \pmod{p}$

简单证明：

性质1：根据原根的定义  
 $g^{\varphi(p)} = g^{p-1} \equiv 1 \pmod{p}$   
 $g_N^N = (g_N)^N = (g^{\frac{p-1}{N}})^N = g^{p-1} \equiv 1 \pmod{p}$  □

性质2：我们化简： $g_N^{\frac{N}{2}} = (g^{\frac{p-1}{N}})^{\frac{N}{2}} = g^{\frac{p-1}{2}}$   
而  $(g^{\frac{p-1}{2}})^2 = g^{p-1} \equiv 1 \pmod{p}$   
因为  $g$  是原根，根据原根的定义， $g^{\frac{p-1}{2}}$  不能等于1，因为等于了就不是原根了，所以  $g^{\frac{p-1}{2}}$  就只能等于-1了hhh □  
（这里也一并证明出了奇素数的原根必是它的二次非剩余）  
剩余性质的证明参见 [我的博文](#)，里面包含了NTT具体的实现以及代码，这里不再列出。

0x65 高次同余方程（二）（N次剩余）

在前面的章节我们讲解了第一种高次同余方程的解法，利用BSGS可以在非常优秀的时间复杂度下解决。这里我们来探讨第二种高次同余方程：  
 $x^a \equiv b \pmod{p}$  的解法。

## Template 1 $X^A \bmod P$ (51 nod 1038)

解高次同余方程:  $x^a \equiv b \pmod{p}$ , 其中  $P$  为质数。给出  $P$  和  $AB$ , 求小于  $P$  的所有  $X$ 。所有数据中, 解的数量不超过  $\sqrt{p}$ 。

$1 \leq T \leq 100, 1 \leq A, B < P \leq 10^9$ ,  $P$  为质数。

### Solution

$p$  是奇质数且  $p$  不能整除  $d$ , 判定  $x^n \equiv d \pmod{p}$  是否有解

若  $n|(p-1)$ , 则  $d$  是模  $p$  的  $n$  次剩余的充要条件为:  $d^{\frac{p-1}{n}} \equiv 1 \pmod{p}$  **且有解时, 解数为  $n$ 。**

若  $n$  不能整除  $p-1$ , 则  $d$  是模  $p$  的  $n$  次剩余的充要条件为:  $d^{\frac{p-1}{k}} \equiv 1 \pmod{p}$ , 其中  $k = \gcd(n, p-1)$ , **且有解时解数为  $k$ 。**

#### ● 解高次同余方程的步骤

1. 先求出  $p$  的一个原根  $g$ 。
2. 求出以  $g$  为底的  $b$  关于模  $p$  的指数  $r$ , 即  $g^r \equiv b \pmod{p}$ , 即离散对数, 前面讲解了如何求离散对数, 直接使用BSGS求解即可。
3. 令  $x \equiv g^y \pmod{p}$ , 带回原式就是求  $g^{ay} \equiv g^r \pmod{p}$ 。(因为  $1 \sim p$  的数都可以表示为原根  $g$  的次幂模  $p$  的形式)
4. 那么根据阶的 **性质61.1.3** 我们知道:  $ay \equiv r \pmod{\text{ord}_p g}$ , 即为一个一次同余方程, 我们可以直接使用 `exgcd` 来求解。
5. 在  $0 \sim \varphi(p) - 1$  中求得  $y$  的解, 带回式子  $x \equiv g^y \pmod{p}$ , 又是一个一次同余方程, 继续exgcd求解即可。

#### ● 一些细节

根据原根的**性质61.2.1**可得, 我们求出来的答案一定没有重复的。由于我们使用的是原根,  $p$  是质数, 原根一共有  $\varphi(\varphi(p))$  个, 仅求出通解的高次同余方程的时间复杂度为  $O(\sqrt{p})$ 。

### Code

```
1 #define int long long
2 const int N = 500007;
3 unordered_map<int, int> hsh;
4 int prime[N], tot, t, phi, ans[N], num, a, b, p, g, x, y;
5 void solve(int x) {
6     int tmp = x;
7     tot = 0;
8     for (int i = 2; i * i <= x; i++) {
9         if (tmp % i == 0) {
10             prime[++tot] = i;
11             while (tmp % i == 0)
12                 tmp /= i;
13         }
14     }
15     if (tmp > 1)
16         prime[++tot] = tmp;
17 }
18 int qpow(int x, int b) {
19     int tmp = 1;
20     while (b) {
21         if (b & 1) tmp = tmp * x % p;
22         b >>= 1;
23         x = x * x % p;
24     }
25     return tmp;
26 }
27 int BSGS(int a, int b) {
28     hsh.clear();
29     int block = sqrt(p) + 1;
30     int tmp = b;
31     for (int i = 0; i < block; i++, tmp = tmp * a % p)
32         hsh[tmp] = i;
33     a = qpow(a, block);
34     tmp = 1;
35     for (int i = 0; i <= block; i++, tmp = tmp * a % p)
36         if (hsh.count(tmp) && i * block - hsh[tmp] >= 0)
37             return i * block - hsh[tmp];
38 }
39 int exgcd(int &x, int &y, int a, int b) {
40     if (b == 0) {
41         x = 1;
42         y = 0;
43         return a;
44     }
45     int d = exgcd(x, y, b, a % b);
46     int z = x; x = y; y = z - (a / b) * y;
47     return d;
48 }
```

```

49 int cnt;
50 signed main() {
51     scanf("%lld", &t);
52     while (t -- ) {
53         scanf("%lld%lld%lld", &p, &a, &b);
54         b %= p;
55         phi = p - 1;
56         solve(phi);
57         for (int i = 1; i <= p; i++) {
58             bool flag = false;
59             for (int j = 1; j <= tot; j++)
60                 if (qpow(i, phi / prime[j]) == 1) {
61                     flag = true;
62                     break;
63                 }
64             if (flag == false) {
65                 g = i;
66                 break;
67             }
68         }
69         int r = BSGS(g, b);
70         int gcd = exgcd(x, y, a, phi);
71
72         if (r % gcd != 0) {
73             puts("No Solution");
74             continue;
75         }
76
77         x = x * r / gcd;
78         int k = phi / gcd;
79         x = (x % k + k) % k;
80         num = 0;
81         while (x < phi) ans[++num] = qpow(g, x), x += k;
82         sort(ans + 1, ans + 1 + num);
83         for (int i = 1; i <= num; i ++ ) printf("%lld%s", ans[i], i == num ? "\n" : " ");
84     }
85     return 0;
86 }

```

### Template 2 【模板】N次剩余 (Luogu P5668)

你需要解方程  $x^n \equiv k \pmod{m}$ , 其中  $x \in [0, m - 1]$ 。

输出第一行 为不同解的个数  $c$ 。

若  $c \neq 0$ , 接下来第二行共  $c$  个整数, **升序输出所有可能解**, 空格隔开。

数据保证  $\sum c_i \leq 10^6$ 。

$1 \leq T \leq 100, 1 \leq n \leq 10^9, 1 \leq k < m \leq 10^9$

设  $m$  的唯一分解形式为  $m = \prod_{i=1}^k p_i^{q_i}$ 。

保证方程  $x^n \equiv k \pmod{p_i^{q_i}}$  在  $[0, p_i^{q_i})$  中的解数  $\leq 10^6$

### Solution

### Code

```

1 #include <bits/stdc++.h>
2
3 typedef long long LL;
4
5 int A, B, mod;
6 int pow(int x, int y, int mod = 0, int ans = 1) {
7     if (mod) {
8         for (; y >>= 1, x = (LL) x * x % mod)
9             if (y & 1) ans = (LL) ans * x % mod;
10    } else {
11        for (; y >>= 1, x = x * x)
12            if (y & 1) ans = ans * x;
13    }
14    return ans;
15 }

```

```

16
17 struct factor {
18     int prime[20], expo[20], pk[20], tot;
19     void factor_integer(int n) {
20         tot = 0;
21         for (int i = 2; i * i <= n; ++i) if (n % i == 0) {
22             prime[tot] = i, expo[tot] = 0, pk[tot] = 1;
23             do ++expo[tot], pk[tot] *= i; while ((n /= i) % i == 0);
24             ++tot;
25         }
26         if (n > 1) prime[tot] = n, expo[tot] = 1, pk[tot++] = n;
27     }
28     int phi(int id) const {
29         return pk[id] / prime[id] * (prime[id] - 1);
30     }
31 } mods, _p;
32
33 int p_inverse(int x, int id) {
34     assert(x % mods.prime[id] != 0);
35     return pow(x, mods.phi(id) - 1, mods.pk[id]);
36 }
37
38 void exgcd(int a, int b, int &x, int &y) {
39     if (!b) x = 1, y = 0;
40     else exgcd(b, a % b, y, x), y -= a / b * x;
41 }
42 int inverse(int x, int mod) {
43     assert(std::__gcd(x, mod) == 1);
44     int ret, tmp;
45     exgcd(x, mod, ret, tmp), ret %= mod;
46     return ret + (ret >> 31 & mod);
47 }
48
49 std::vector<int> sol[20];
50
51 void solve_2(int id, int k) {
52     int mod = 1 << k;
53     if (k == 0) { sol[id].emplace_back(0); return; }
54     else {
55         solve_2(id, k - 1); std::vector<int> t;
56         for (int s : sol[id]) {
57             if (!(pow(s, A) ^ B & mod - 1))
58                 t.emplace_back(s);
59             if (!(pow(s | 1 << k - 1, A) ^ B & mod - 1))
60                 t.emplace_back(s | 1 << k - 1);
61         }
62         std::swap(sol[id], t);
63     }
64 }
65
66 int BSGS(int B, int g, int mod) { //  $g^x = B \pmod{M} \Rightarrow g^{iL} = B * g^j \pmod{M} : iL = j$ 
67     std::unordered_map<int, int> map;
68     int L = std::ceil(std::sqrt(mod)), t = 1;
69     for (int i = 1; i <= L; ++i) {
70         t = (LL) t * g % mod;
71         map[(LL) B * t % mod] = i;
72     }
73     int now = 1;
74     for (int i = 1; i <= L; ++i) {
75         now = (LL) now * t % mod;
76         if (map.count(now)) return i * L - map[now];
77     }
78     assert(0);
79 }
80
81 int find_primitive_root(int id) {
82     int phi = mods.phi(id); _p.factor_integer(phi);
83     auto check = [&] (int g) {
84         for (int i = 0; i < _p.tot; ++i)
85             if (pow(g, phi / _p.prime[i], mods.pk[id]) == 1)
86                 return 0;

```



```

87     return 1;
88 };
89 for (int g = 2; g < mods.pk[id]; ++g) if (check(g)) return g;
90 assert(0);
91 }
92
93 void division(int id, int a, int b, int mod) { // ax = b (mod M)
94     int M = mod, g = std::__gcd(std::__gcd(a, b), mod);
95     a /= g, b /= g, mod /= g;
96     if (std::__gcd(a, mod) > 1) return;
97     int t = (LL) b * inverse(a, mod) % mod;
98     for (; t < M; t += mod) sol[id].emplace_back(t);
99 }
100
101 void solve_p(int id, int B = ::B) {
102     int p = mods.prime[id], e = mods.expo[id], mod = mods.pk[id];
103     if (B % mod == 0) {
104         int q = pow(p, (e + A - 1) / A);
105         for (int t = 0; t < mods.pk[id]; t += q)
106             sol[id].emplace_back(t);
107     } else if (B % p != 0) {
108         int phi = mods.phi(id);
109         int g = find_primitive_root(id), z = BSGS(B, g, mod);
110         division(id, A, z, phi);
111         for (int &x : sol[id]) x = pow(g, x, mod);
112     } else {
113         int q = 0; while (B % p == 0) B /= p, ++q;
114         int pq = pow(p, q);
115         if (q % A != 0) return;
116         mods.expo[id] -= q, mods.pk[id] /= pq;
117         solve_p(id, B);
118         mods.expo[id] += q, mods.pk[id] *= pq;
119         if (!sol[id].size()) return;
120
121         int s = pow(p, q - q / A);
122         int t = pow(p, q / A);
123         int u = pow(p, e - q);
124
125         std::vector<int> res;
126         for (int y : sol[id]) {
127             for (int i = 0; i < s; ++i)
128                 res.emplace_back((i * u + y) * t);
129         }
130         std::swap(sol[id], res);
131     }
132 }
133
134 std::vector<int> allans;
135 void dfs(int dep, int ans, int mod) {
136     if (dep == mods.tot) {allans.emplace_back(ans); return;}
137     int p = mods.pk[dep], k = p_inverse(mod % p, dep);
138     for (int a : sol[dep]) {
139         int nxt = (LL) (a - ans % p + p) * k % p * mod + ans;
140         dfs(dep + 1, nxt, mod * p);
141     }
142 }
143
144 void solve() {
145     std::cin >> A >> mod >> B, mods.factor_integer(mod);
146     allans.clear();
147     for (int i = mods.tot - 1; ~i; --i) {
148         sol[i].clear();
149         mods.prime[i] == 2 ? solve_2(i, mods.expo[i]) : solve_p(i);
150         if (!sol[i].size()) {return std::cout << 0 << '\n', void(0);}
151     }
152     dfs(0, 0, 1), std::sort(allans.begin(), allans.end());
153     std::cout << allans.size() << '\n';
154     for (int i : allans) std::cout << i << ' '; std::cout << '\n';
155 }
156
157 int main() {

```

```
158     std::ios::sync_with_stdio(0), std::cin.tie(0);
159     int tc; std::cin >> tc; while (tc--) solve();
160     return 0;
161 }
162
```

我们知道此方法解高次同余方程的时间复杂度为  $\sqrt{p}$ ，但是  $p$  很大，效率还是有些低下，所以当  $a = 2$  的特殊情况，即二次同余方程  $x^2 \equiv n \pmod{p}$ ，我们将在下一章介绍一个更为优秀的 Cipolla 算法来解决这类问题。

# 0x70 二次剩余

## 0x71 二次剩余与二次非剩余

• 定义

对于二次同余方程  $x^2 \equiv n \pmod{p}$  有解，则称  $n$  为  $p$  的二次剩余， $x$  为该二次同余方程的解。

二次剩余  $n$ ，就是一个二次项  $\%p$  后的剩余。

$n$  当对任意  $x$ ， $x^2 \equiv n \pmod{p}$  不成立时，称  $d$  是模  $p$  的二次非剩余。

• 应用

求  $\sqrt{n}\%p$ ，若  $n$  为  $p$  的二次剩余，则  $\sqrt{n}\%p = x\%p$ 。

即：若该二次同余方程有解，则  $n$  可以在模  $p$  的意义下开根。

## 0x72 Cipolla 算法解算法二次同余方程

需要保证模  $p$  是奇素数。

引入勒让德符号： $\left(\frac{n}{p}\right)$

表示  $n$  是否为  $p$  的二次剩余，1 和 -1 表示是与否，0 表示  $n = 0$  的情况。

• 定理

**定理72.1：**  $\left(\frac{n}{p}\right) = n^{\frac{p-1}{2}} \pmod{p}$

**定理72.2：** 若找到一个  $a$  使得  $a^2 - n = w$ ，且  $\left(\frac{w}{p}\right) = -1$  则  $x = (n + \sqrt{w})^{\frac{p+1}{2}}$  为  $x^2 \equiv n \pmod{p}$  的解

**定理72.3：** 对于二次同余方程  $x^2 \equiv n \pmod{p}$  有  $\frac{p-1}{2} + 1$  个  $n$  使此方程有解

• 实现

求解方程  $x^2 \equiv N \pmod{p}$ 。保证  $p$  是奇素数。输入一个  $T$  代表数据组数，接下来  $T$  行，每行一个  $N$  和一个  $p$ 。对于每一行输出：若有解，则按  $\text{mod } p$  后递增的顺序输出在  $\text{mod } p$  意义下的全部解。若两解相同，只输出其中一个。若无解，则输出 **No Solution**

有了上面的三个定理，现在我们的唯一问题就是如何找到合适的  $a$  使得  $\omega$  满足条件，我们考虑**随机**，由**定理72.3**可知， $a^2 - n$  在模  $p$  意义下为非二次剩余的概 率为  $\frac{p-1}{2p}$ ，那么这样随机的期望次数就接近于 2，可以看做是  $O(1)$  求。因为在整个过程中使用过快速幂，所以时间复杂度为  $O(\log p)$ 。

```
1 int mod;
2 ll I_mul_I; // 虚数单位的平方
3 struct Complex { // 建议自己实现复数
4     ll real, imag;
5     Complex(ll real = 0, ll imag = 0): real(real), imag(imag) { }
6 };
7 inline bool operator == (Complex x, Complex y) {
8     return x.real == y.real and x.imag == y.imag;
9 }
10 inline Complex operator * (Complex x, Complex y) {
11     return Complex((x.real * y.real + I_mul_I * x.imag % mod * y.imag) % mod,
12                    (x.imag * y.real + x.real * y.imag) % mod);
13 }
14 Complex qpow(Complex x, int k) {
15     Complex res = 1;
16     while(k) {
17         if(k & 1) res = res * x;
18         x = x * x;
19         k >>= 1;
20     }
21     return res;
22 }
```

```
23 bool check_if_residue(int x) {
24     return qpow(x, (mod - 1) >> 1) == 1;
25 }
26 int solve(int n, int p) {
27     n %= p, mod = p;
28     if(p == 2) return n;
29     ll a = rand() % mod;
30     if(qpow(n,(mod - 1) / 2) == p - 1) return -1;//不存在
31     while(!a || check_if_residue((a * a + mod - n) % mod))
32         a = rand() % mod;
33     I_mul_I = (a * a + mod - n) % mod;
34
35     return int(qpow(Complex(a, 1), (mod + 1) >> 1).real);
36 }
37 int n, m, p, t;
38 int main()
39 {
40     //srand(time(0));
41     scanf("%d", &t);
42     while(t -- ) {
43         scanf("%d%d", &n, &p);
44         int ans1 = 0, ans2 = 0;
45         if(n == 0) {
46             puts("0");
47             continue;
48         }
49         ans1 = solve(n, p);
50         if(ans1 == -1) puts("No Solution");
51         else {
52             ans2 = p - ans1;
53             if(ans1 > ans2) swap(ans1, ans2);
54             if(ans1 == ans2) printf("%d\n", ans1);
55             else printf("%d %d\n", ans1, ans2);
56         }
57     }
58     return 0;
59 }
```

## 0x80 某些非线性丢番图方程

### 0x81 毕达哥斯拉三元组（勾股数）

满足  $x^2 + y^2 = z^2$  的  $(x, y, z)$  三元组称为毕达哥拉斯三元组，当  $\gcd(x, y, z) = 1$  时，称其为本原的毕达哥斯拉三元组。

毕达哥拉斯三元组，也称为勾股数。

• **基本性质定理**

**引理81.1：** 如果  $x, y, z$  为一个本原毕达哥拉斯三元组，则  $\gcd(x, y) = \gcd(y, z) = \gcd(z, x) = 1$ 。

**引理81.2：** 设  $x, y, z$  为一个本原毕达哥拉斯三元组，则  $x$  为偶数且  $y$  为奇数或者  $x$  为奇数， $y$  为偶数。

**引理81.3：** 若  $r, s$  和  $t$  为正整数，且  $\gcd(r, s) = 1$ ， $rs = t^2$ ，则存在整数  $m, n$ ，使得  $r = m^2$ ， $s = n^2$ 。（两个平方数的乘积还是一个平方数）

由此，我们可以证明得到所有的毕达哥拉斯三元组的解了。

**定理81.4：** 由正整数  $x$ 、 $y$ 、 $z$  构成的三元组  $(x, y, z)$ ，其中  $y$  为偶数，那么由他们构成的本原的毕达哥拉斯三元组当且仅当存在

**互素的一奇一偶**的正整数  $m$ 、 $n$ ，且  $m > n$ ，满足

$$\begin{cases} x = m^2 - n^2 \\ y = 2mn \\ z = m^2 + n^2 \end{cases}$$

我们可以看出，本原的毕达哥拉斯三元组中，**最大的数一定是奇数**。

特别地，由  $f_n f_{n+3}$ ， $2f_{n+1} f_{n+2}$ ， $f_{n+1}^2 + f_{n+2}^2$  构成毕达哥拉斯三元组，将  $f_n = f_{n+2} - f_{n+1}$ ， $f_{n+3} = f_{n+1} + f_{n+2}$  即得

• **求解  $n$  以内本原的毕达哥拉斯三元组个数**

根据  $\begin{cases} x = m^2 - n^2 \\ y = 2mn \\ z = m^2 + n^2 \end{cases}$ ，只要枚举一下  $m$ 、 $n$ （ $m, n \leq \sqrt{n}$ ），然后将三元组乘以  $i$ （保证  $i \times z$  在范围内），即可求出所有的毕达哥拉斯三元组。

```
1 int x[N], y[N], z[N];
2 int pythagoras(int n) {
3     int num = 0; //数组下标
4     int res = 0; //本原三元组的个数
5     int m = sqrt(n * 1.0);
6     for (int i = 1; i <= m; i += 2) { //从 1 开始, 每次 + 2, 保证为奇数
7         for (int j = 2; j <= m; j += 2) { //从 2 开始, 每次 + 2, 保证为偶数
8             a = max(i, j); //大的为 m
9             b = min(i, j); //小的为 n
10            if (gcd(i, j) != 1) //要求 m,n 互质
11                continue;
12            x[num] = a * a - b * b;
13            y[num] = 2 * a * b;
14            z[num] = a * a + b * b;
15            num++;
16            if ((a * a + b * b) <= n) //保证在范围内
17                res++;
18        }
19    }
20    return res;
21 }
```

**Problem A Find Integer （18年CCPC网络赛）**

给你两个整数  $n, a$ , 找到整数  $b, c$ , 使  $a^n + b^n = c^n$ , 其中  $t$  组数据,  $1 \leq t \leq 10^6, 1 \leq a \leq 10^4, 1 \leq n \leq 10^9$ 。

**Solution**

费马大定理可知,  $n > 2$ 或 $n \leq 0$  时, 不存在正整数解。

当  $n = 1$  时我们直接构造即可。当  $n = 2$  时, 既是一个毕达哥斯拉三元组, 按照上述定理求解即可。

```
1 int main() {
2     int t, a, n;
3     ll b, c;
4     scanf("%d", &t);
5     while (t--) {
6         scanf("%d%d", &n, &a);
7         if (n > 2 || n == 0)
8             puts("-1 -1");
9         else if (n == 1) printf("1 %d", a + 1);
10        else {
11            if (a & 1) {
12                c = (a * a + 1) / 2;
13                b = c - 1;
14            } else {
15                c = (a * a / 2 + 2) / 2;
16                b = c - 2;
17            }
18            printf("%lld %lld\n", b, c);
19        }
20    }
21 }
```

**0x82 费马大定理**

费马大定理：

- $m > 2$ 时,  $x^m + y^m = z^m$  无正整数解
- 当 $m = 2$ , 对于式子 $a^2 + b^2 = c^2$  ( $n$ 为任意正整数) :
  - 当  $a$  为奇数时:  $a = 2n + 1, c = n^2 + (n + 1)^2, b = c - 1$
  - 当  $a$  为偶数时:  $a = 2n + 2, c = 1 + (n - 1)^2, b = c - 2$

**0x83 平方和**

- **费马平方和定理**

奇质数能表示为两个平方数之和的充分必要条件是该质数被 4 除余 1。

1.如果两个整数都能表示为两个平方数之和的形式，则他们的积也能表示为两个平方数之和的形式。

$$\begin{aligned}(a^2 + b^2)(c^2 + d^2) &= a^2c^2 + a^2d^2 + b^2c^2 + b^2d^2 \\ &= (a^2c^2 + b^2d^2 - 2abcd) + (a^2d^2 + b^2c^2 + 2abcd) \\ &= (ac - bd)^2 + (ad + bc)^2\end{aligned}$$

2.如果一个能表示为两个平方数之和的整数，能被另一个能表示为两个平方数之和的素数整除，则他们的商也能表示为两个平方数之和。

$$\text{即 } \frac{a^2 + b^2}{p^2 + q^2} = (\frac{qp + bq}{p^2 + q^2})^2 + (\frac{aq - bp}{p^2 + q^2})^2$$

3.如果  $a, b$  互质，则  $a^2 + b^2$  的所有因子都可以表示为两个平方数的和

4.任何形如  $4n + 1$  的素数都能表示为两个平方数之和的形式

- 拉格朗日四平方和定理

每个正整数都能表示为四个整数的平方和形式。

- 欧拉恒等式

$$\begin{aligned}&(a^2 + b^2 + c^2 + d^2)(x^2 + y^2 + z^2 + w^2) \\ &= (ax + by + cz + dw)^2 + (ay - bx + cw - dz)^2 + (az - bw - cx + dy)^2 + (aw + bz - cy - dx)^2\end{aligned}$$

## 0x84 佩尔方程与连分数

### 0x84.1 佩尔方程与连分数

- 基本性质定理

连分数是一种特殊的繁分数，其形式为： $a_1 + \frac{1}{a_2 + \frac{1}{a_3 + \frac{1}{\dots}}}$ ，通常记为： $[a_1, a_2, \dots, a_n] = \frac{p_n}{q_n}$ ，其中  $p_n$  和  $q_n$  称为连分数多项式，对于任意的  $a$  均为一次式，它们的比值称为第  $n$  个渐进值渐进分数。

佩尔 (Pell) 方程是一种不定二次方程，其与连分数，二次型，代数论等有着重要的联系。

其形式为： $x^2 - dy^2 = 1, (d \in N^+)$ ，其中  $d$  不为非平方数

- 佩尔方程迭代公式

定义：设  $p, q$  为整数，且满足  $p^2 - Dq^2 = T$ ，则称  $a = p - q\sqrt{D}$  给出该方程的解

- 推论

设  $\begin{cases} a = x_1 - y_1\sqrt{D} \\ b = x_2 - y_2\sqrt{D} \end{cases}$  给出方程  $x^2 - Dy^2 = T$  的解

则： $ab = A - B\sqrt{D}$  给出方程  $x^2 - Dy^2 = T^2$  的解，其中  $\begin{cases} A = x_1x_2 + Dy_1y_1 \\ B = x_1y_2 + x_2y_1 \end{cases}$

取  $d = D, T = 1$ ，则有佩尔方程  $x^2 - dy^2 = 1, (d \in N^+)$

若佩尔方程的最小特解为  $(x_1, y_1)$ ，故有迭代公式： $\begin{cases} x_n = x_{n-1}x_1 + dy_{n-1}y_1 \\ y_n = x_{n-1}y_1 + y_{n-1}x_1 \end{cases}$

将迭代公式写为矩阵的形式，有： $\begin{pmatrix} x_k \\ y_k \end{pmatrix} = \begin{pmatrix} x_1 & dy_1 \\ y_1 & x_1 \end{pmatrix}^{k-1} \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}$ ，可以通过矩阵快速幂找出第  $k$  大的解

- 连分数求解佩尔方程

对于连分数  $[a_1, a_2, \dots, a_n] = \frac{p_n}{q_n}$  的渐进值来讲，有着递归关系式：

$$\begin{cases} p_1 = a_1, p_2 = a_2a_1 + 1, \dots, p_{n+1} = a_{n+1}p_n + p_{n-1} \\ q_1 = 1, \quad q_2 = a_2, \quad \dots, \quad q_{n+1} = a_{n+1}q_n + q_{n-1} \end{cases}, n \geq 2$$

通过数学归纳法，可得到关系式： $p_{n+1}q_n - p_nq_{n+1} = (-1)^{n-1}, n \geq 1$

定义：对于正整数  $p, q$ ，若有  $|p^2 - a^2q^2| < a$ ，则比值  $\frac{p}{q}$  必为  $a$  的一个渐近值

由此可得：佩尔方程的全部根的集合为  $x^2 - dy^2 = (p^2 - dq^2)^n = 1$

由佩尔方程的迭代公式  $\begin{cases} x_n = x_{n-1}x_1 + dy_{n-1}y_1 \\ y_n = x_{n-1}y_1 + y_{n-1}x_1 \end{cases}$  可得出佩尔方程的最小解

$$\text{即: } \begin{cases} x = \frac{(p+\sqrt{d}q)^n + (p-\sqrt{d}q)^n}{2} \\ y = \frac{(p+\sqrt{d}q)^n - (p-\sqrt{d}q)^n}{2\sqrt{d}} \end{cases}$$

### 0x84.2 解佩尔方程

- 暴力寻找最小解

```
1 int x[N], y[N];
2 void pell(int &a, int &b, int d) { //暴力寻找pell方程最小解
3     b = 1;
4     while (true) {
```

```

5     a = (LL)sqrt(d * b * b + 1);
6     if (a * a - d * b * b == 1)
7         break;
8     b++;
9 }
10 }
11 int main() {
12     int d;
13     while (scanf("%d", &d) != EOF) {
14         int m = (int)sqrt((double)d);
15         if (m * m == d) { //d不能为完全平方数
16             cout << "No Solution" << endl;
17             continue;
18         }
19         int a = 0, b = 0;
20         pell(a, b, d); //暴力找到最小解
21         cout << a << " " << b << endl;
22     }
23     return 0;
24 }

```

#### • 迭代公式求前 $n$ 个解

使用迭代公式求解  $pell$  方程的前  $n$  个解时，应先用暴力寻找到最小解，然后再套用迭代公式求出前  $n$  个解，由于  $pell$  方程相邻两解之间的差值较大， $n$  一般很小

```

1 int x[N], y[N];
2 void pell(int &a, int &b, int d) { //暴力寻找pell方程最小解
3     b = 1;
4     while (true) {
5         a = (LL)sqrt(d * b * b + 1);
6         if (a * a - d * b * b == 1)
7             break;
8         b ++ ;
9     }
10 }
11 int main() {
12     int d;
13     while (scanf("%d", &d) != EOF) {
14         int m = (int)sqrt((double)d);
15         if (m * m == d) { //d不能为完全平方数
16             cout << "No Solution" << endl;
17             continue;
18         }
19         int a = 0, b = 0;
20         pell(a, b, d); //暴力找到最小解
21         x[1] = a, y[1] = b; //第一组解
22         for (int i = 2; i <= 10; i ++ ) { //递推公式
23             x[i] = x[i - 1] * x[1] + 2 * y[i - 1] * y[1];
24             y[i] = x[i - 1] * y[1] + y[i - 1] * x[1];
25         }
26         for (int i = 1; i <= 10; i ++ )
27             cout << x[i] << " " << y[i] << endl;
28     }
29     return 0;
30 }

```

#### • 连分数法

当要求  $pell$  方程的最小解时，暴力可能会  $TLE$ ，此时可以使用连分数法，其关键是计算连分数的展开

```

1 int a[20000];
2 bool pell(int &x, int &y, int d) {
3     int m = (int)sqrt((double)d);
4     if (m * m == d) //d不能为完全平方数
5         return false;
6     //将d以连分数形式存储
7     int num = 0; //连分数数位
8     double sq = sqrt(d); //d的高精度根，相当于r0
9     a[num++] = m; //存储整数部分
10    int b = m; //当前整数部分
11    int c = 1; //连分数最终展开时的分母
12    double temp; //连分数展开时的每一项
13    do {

```



```
14     c = (d - b * b) / c;
15     temp = (sq + b) / c;
16     a[num++] = (int)(floor(temp));
17     b = a[num - 1] * c - b;
18 } while (a[num - 1] != 2 * a[0]); //当有一位等于整数两倍时结束
19 //将连分数形式化为分子分母形式，即求p、q两个值
20 int p = 1, q = 0;
21 for (int i = num - 2; i >= 0; i -- ) {
22     int temp = p;
23     p = q + p * a[i];
24     q = temp;
25 }
26 if ((num - 1) % 2) { //连分数长度为奇数时
27     x = 2 * p * p + 1;
28     y = 2 * p * q;
29 } else { //连分数长度为偶数时
30     x = p;
31     y = q;
32 }
33 return true;
34 }
35 int main() {
36     int d;
37     while (scanf("%d", &d) != EOF) {
38         int x, y;
39         if (pell(x, y, d))
40             cout << x << " " << y << endl;
41         else
42             cout << "No Solution" << endl;
43     }
44     return 0;
45 }
```

0x84.3 竞赛例题选讲

Problem A Square Number (hdu 2281)

输入一个数  $N$ ，找到满足下式的  $n, x$ ，且  $n \leq N$ 。

$$x^2 = \frac{1^2+2^2+\cdots+n^2}{n}$$

Solution

将右边式子的分子求和化简，有： $x^2 = \frac{(n+1)(2*n+1)}{6}$

移项化简，有： $(4n + 3)^2 - 48 * x^2 = 1$

我们发现满足佩尔方程的形式，直接带入佩尔方程公式求解即可。

Problem B Street Numbers (poj 1320)

有  $m$  个编号从 1 到  $m$  的房子，问是否存在  $1 + 2 + 3 + \dots + (N - 1) = (N + 1) + (N + 2) + \dots + (M)$ ，求出前 10 个  $n, m$ 。

Solution

将左右两端的等差数列求和，有： $(2M + 1)^2 - 8N^2 = 1$

满足佩尔方程的形式： $x = 2m + 1, y = n$

可以得到最小的一组解为  $x = 3, y = 1$ ，直接求前 10 个佩尔方程的解即可。

0x90 高斯整数

0x90.0 复数

复数分为实部和虚部，可以描述为一个二元组  $(x, y)$ ，表示这个数等于  $x + y\sqrt{-1}$ 。一般用  $i$  表示  $\sqrt{-1}$ 。

由于是个二元组，所以它在理解的时候可以抽象为一个二维向量，分布在平面直角坐标系上。

事实上，它确实也有不少性质和向量相同。

- 复数的模

定义复数的模，为复平面原点到  $(a, b)$  的距离，即  $|z| = \sqrt{x^2 + y^2}$

定义  $z$  的**范数**  $N(z) = |z|^2 = x^2 + y^2$ 。

- **共轭复数**

复数  $z = a + bi$  的共轭是  $z' = a - bi$ ，记作  $\bar{z}$ 。

性质:  $z \times \bar{z} = a^2 + b^2$ ,  $|z| = |\bar{z}|$ 。

- **复数的大小**

复数可以看做和向量一样，无法比较大小。

- **复数的加减**

对应部 的加减，如  $(a, c) + (b, d) = (a + b, c + d)$

- **复数乘法**

两个复数  $z_1 = a_1 + b_1i$ ,  $z_2 = a_2 + b_2i$  相乘的结果为

$$z_0 = z_1 \times z_2 = (a_1 + b_1i) \times (a_2 + b_2i) = a_1a_2 + a_1b_2i + a_2b_1i + b_1b_2i^2$$

又因为  $i^2 = -1$ ，所以

$$z_0 = (a_1a_2 - b_1b_2) + (a_1b_2 + a_2b_1)i$$

在复平面上， $z_1, z_2, z_0$  所对应的幅角  $\theta_1, \theta_2, \theta_0$

有关系:  $\theta_0 = \theta_1 + \theta_2$

- **复数的除法**

对于两个复数  $z_1 = a_1 + b_1i$ ,  $z_2 = a_2 + b_2i$ ，他们相除的结果为

$$z_0 = \frac{z_1}{z_2}$$

考虑分数上下同时乘  $\overline{z_2}$ ，有

$$z_0 = \frac{z_1 \overline{z_2}}{a_2^2 + b_2^2}$$

分母是一个实数，可以直接将分子的实部虚部除以分母。

或者也可以展开：

$$\frac{a+c \times i}{b+d \times i} = \frac{(a+c \times i)(b-d \times i)}{(b+d \times i)(b-d \times i)} = \frac{(ab+cd)+(bc-ad) \times i}{b^2+d^2}$$

即：

$$\frac{(a,b)}{(c,d)} = \left( \frac{ab+cd}{b^2+d^2}, \frac{bc-ad}{b^2+d^2} \right)$$

- **复数指数幂：**

有**欧拉公式** $e^{i\theta} = \cos \theta + i \sin \theta$ 其中  $e$  是自然对数的底数

当取  $\theta = \pi$  时，有 $e^{i\pi} = \cos \pi + i \sin \pi$ 又因为 $\cos \pi = 1, \sin \pi = 0$

所以 $e^{i\pi} = -1$

也就是:  $e^{i\pi} + 1 = 0$

## 0x91 高斯整数

- **定义：**

形如  $a + bi$ （其中 $a, b$ 是整数）的复数被称为高斯整数，高斯整数全体记作集合  $Z[i] = \{a + bi | a, b \in Z\}$ , where  $i^2 = -1$ ，换言之，高斯整数是实部和虚部都为整数的复数。由于高斯整数在乘法和加法下交换，它们形成了一个**交换环**。也就是高斯整数是加、减、乘运算下封闭。

满足**加法、乘法**以及**交换律、结合律、分配率**并有 **0 元**和**单位元**的集合称为**交换环**。

在复平面上，高斯整数是二维复平面上的**整点** $(a, b)$ 。

高斯整数的模是它和自己共轭复数的乘积，即范数  $N(a + bi) = (a + bi)(a - bi) = a^2 + b^2$ ，它的模可以表示为两个数字的平方和，所以不能表示为  $4k + 1$ , where  $k \in Z$ 。

- **整除**

设  $\alpha, \beta$  是高斯整数，我们称  $\alpha$  整除  $\beta$ ，是指存在一个高斯整数  $\gamma$ ，使得  $\beta = \alpha\gamma$ 。同整数集，若  $\alpha$  整除  $\beta$ ，记作  $\alpha \mid \beta$ 反之记作， $\alpha \nmid \beta$ 。

- **带余除法**

也称欧几里得除法，高斯整环  $Z[i]$  是欧几里得环，所以它具有很多在整数域和多项式域上成立的特性，比如辗转相除，裴蜀定理，主理想，欧几里德引理，唯一分解定理，中国剩余定理。

**定理91.1：** 设  $\alpha, \beta$  为高斯整数且  $\beta \neq 0$ ，则存在高斯整数  $\gamma$  和  $\rho$ ，使得  $\alpha = \beta\gamma + \rho$ 。

而且  $0 \leq N(\rho) < N(\beta)$ 。这里的  $\gamma$  被成为商， $\rho$  被称为 余数。

**欧几里德引理**

$\forall a, b, p$  where  $p$  is a prime

$$p|ab \Rightarrow p|a \text{ or } p|b$$

如果将高斯整数  $a$  写为  $a = bq + r$ , 有 $N(r) \leq \frac{N(b)}{2}$ 。

证明:

令  $\frac{a}{b} = x + yi$ , 令 $-\frac{1}{2} \leq x - m \leq \frac{1}{2}$ ,  $-\frac{1}{2} \leq y - n \leq \frac{1}{2}$ , 令 $q = m + ni$ , 由 $a = bq + r$ ,  $r = b(x - m + (y - n)i)$ , 故 $N(r) \leq \frac{N(b)}{2}$ 。

## 0x92 高斯素数

**定义92.1**：若  $\epsilon \mid 1$ , 则称高斯整数  $\epsilon$  是 **单位** , 若  $\epsilon$  是单位, 则称  $\epsilon\alpha$  是高斯整数  $\alpha$  的一个相伴, 高斯整数的单位为  $1, -1, i, -i$ 。

**定义92.2**：若非零高斯整数  $\pi$  不是单位, 而且只能够被单位和它的相伴整除, 则称之为高斯素数。

**定理92.3**：若  $\pi$  是高斯整数, 而且  $N(\pi) = p$ , 其中  $p$  是有理整数, 则  $\pi$  和  $\bar{\pi}$  是高斯素数, 而  $p$  不是高斯素数。

高斯整数形成了一个唯一分解域。高斯整数只有当且仅当它是一个素数时才不可约分, 高斯整数中  $Z[i]$  的素数称为高斯素数。

- 高斯素数的共轭复数依然为高斯素数。
- 高斯素数的相伴复数依然为高斯素数。

$$\forall a + bi \in Z[i], -a + bi \in Z[i]$$

- 作为高斯素数的整素数  $p$  是  $4k + 3$  型素数, 其余的素数可以写为 2 个共轭高斯素数的乘积
- 一个高斯整数  $a + bi$  是高斯素数当且仅当以下两个条件之一成立:

- $a, b$  中一个为 0, 且另一个数字为  $4k + 3$  型素数

若  $p$  为  $4k + 3$  型素数, 存在  $d$ ,  $d$  不等于单位元, 也不等于  $p$ , 满足  $d \mid p$ , 则  $d\bar{d} \mid p$ , 由于  $d\bar{d}$  为整数, 有  $d\bar{d} = p$ , 得出  $p = a^2 + b^2$ ,  $a^2 + b^2 \equiv 0, 1, 2(mod\ 4)$ , 与  $p$  为  $4k + 3$  型素数矛盾。

- $a, b$  都非 0, 且  $a^2 + b^2$  是素数

令  $u = a + bi$ , 则  $u\bar{u} = p$ ,  $p$  为一个素数, 若存在存在  $d$ ,  $d$  不等于单位元, 也不等于  $u$ ,  $d \mid u$ , 则  $d\bar{d} \mid p$ , 由于  $d\bar{d}$  为整数, 有  $d\bar{d} = p$ , 得出  $d = u$ , 与假设矛盾。

## 0x93 唯一分解

由于高斯整环是一个唯一分解域, 所以每一个高斯整数都可以写为一个单位元和若干高斯素数的乘积, 这种分解是唯一的 (忽略共轭和相伴)。

$$\forall a \in Z[i], a = u \cdot (1 + i)^{e_0} \prod p_m^{e_i}, \text{ where } u \in \{1, -1, i, -i\}, 0 \leq e_i$$

## 0x94 最大公约数

两个高斯整数的最大公约数并不唯一, 加入  $d$  是  $a$  与  $b$  的最大公约数, 则  $a, b$  的最大公约数为  $d, -d, id, -id$ 。

若 $a = i^k \prod p_m^{\nu_m}$ ,  $b = i^n \prod p_m^{\mu_m}$ , 则其中一个最大公约数为

$$d = \prod p_m^{\lambda_m}, \text{ where } \lambda_m = \min(\nu_i, \mu_i)$$

可以根据带余除法里的结论, 进行辗转相除, 时间复杂度为  $O(\log(n))$ 。

```
1 Complex div(Complex a, Complex b) {
2     long double mo = b.norm();
3     Complex c = a * b.conj();
4     long double r = 1. * c.r / mo, i = 1. * c.i / mo;
5     return Complex(round(r), round(i));
6 }
7 Complex gcd(Complex a, Complex b) {
8     if (b.r == 0 && b.i == 0) return a;
9     Complex c = div(a, b);
10    return gcd(b, a - b * c);
11 }
```

## 0x95 同余和剩余系

给定一个高斯整数  $z_0$ , 对于高斯整数  $z_1, z_2$ , 如果它们的差是  $z_0$  的整数倍, 即  $z_1 - z_2 = kz_0, k \in Z$ , 那么称  $z_1$  与  $z_2$  模  $z_0$  同余, 写作  $z_1 \equiv z_2 \pmod{z_0}$ 。

模  $z_0$  同余是一种等价关系, 它将高斯整数分成若干个等价类, 称为剩余类, 剩余类写作  $Z/z_0Z$ , 剩余类形成了一个交换环。

## 0x96 费马二平方和定理

$p$  是一个素数,  $p$  可以写成两个平方数的和, 当且仅当  $p = 2$  或  $p \equiv 1 \pmod{4}$ 。

充分性: 令  $p = a^2 + b^2$ , 对任意一个整数  $x$ , 有  $x^2 \equiv 0, 1, 2 \pmod{4}$ , 故  $a^2 + b^2 \equiv 0, 1, 2 \pmod{4}$ , 由于  $p$  是素数, 所以  $p = 2$  或  $p \equiv 1 \pmod{4}$ 。

必要性: 当  $p = 2$ ,  $2 = 1^2 + 1^2$ , 当  $p$  为  $4k + 1$  型素数, 由于  $p$  不是高斯素数, 所以可以进行分解, 即  $p = u\bar{u}$ ,  $u = a + bi$ , 故  $p = a^2 + b^2$ 。

## 0x97 分解 $4k+1$ 型素数

$p$  为  $4k + 1$  型素数, 如果存在  $k^2 \equiv -1 \pmod{p}$ , 则  $p \mid (k + i)(k - i)$ , 由于  $p = u\bar{u}$ , 有  $u \mid (k + i)$ , 故  $u = (k + i, p)$ , 即可将  $p$  分解为两个平方数的和。

由于有一半的数在模  $p$  下存在平方剩余, 随机一个  $t$ , 检验  $t^{\frac{p-1}{2}} \equiv -1 \pmod{p}$ , 令  $k = t^{\frac{p-1}{4}}$ , 时间复杂度为  $O(T \log(p))$ ,  $T$  为测试次数。

## 0x98 构造 $a^2 + b^2 = n$ 的方案

1. 首先将  $n$  分解为  $n = \prod p_m^{e_m}$  的形式
2. 将 2 分解为  $(1 + i)(1 - i)$ , 将  $4k + 1$  型素数分解为  $u\bar{u}$ , 其中  $u$  为高斯素数,  $4k + 3$  型素数不可分解
3.  $n = \prod p_m^{e_i}$ ,  $0 \leq e_i$ , 由  $n = u\bar{u}$ , 故需要将  $n$  中的高斯素数分成两部分, 使得两部分共轭。考虑素数  $p_m$ , 对于  $4k + 1$  型素因子, 分解为  $u^{e_m}\bar{u}^{e_m}$ , 左边可以放  $k$  个  $u$ ,  $e_m - k$  个  $\bar{u}$ ; 对于  $4k + 3$  型素因子, 因为不能进行分解, 所以左右两边的数量应该一样多, 即  $4k + 3$  型的素数  $p_m$  出现的次数必须为偶数。

令  $f(n) = \frac{1}{4} \sum_{x,y \in \mathbb{Z}} [x^2 + y^2 = n]$ , 除以 4 是因为由 4 个单位元。由于素因子可以分开考虑, 所以该函数为积性函数。

1.  $f(2^k) = 1$
2.  $f(p^e) = e + 1$ , 当  $p$  为  $4k + 1$  型素数
3.  $f(p^{2e+1}) = 0$ ,  $f(p^{2e}) = 1$ , 当  $p$  为  $4k + 3$  型素数

## 0x99 竞赛例题选讲

### Problem A A math problem (HDU 2650)

给出  $a + bj$ , 其中  $j = \sqrt{-2}$ , 判断  $a + bj$  是否为高斯素数。

#### Solution

我们按照上面证明的判断高斯素数的方法直接模拟即可。

注意这里不是正常的复数, 这里设  $j = \sqrt{-2}$ 。

$$|j| = \sqrt{a^2 + (\sqrt{-2}b)^2}$$

很明显原来的判定中  $a^2 + b^2$  就变成了  $a^2 + 2b^2$ 。同理若  $j = \sqrt{-k}$ , 就是  $a^2 + kb^2$ 。

由于这里的数据很大, 所以我们判断素数的时候需要用到Miller\_Rabin算法

```
1 typedef long long ll;
2 ll mul(ll a, ll b, ll m) {
3     ll ans = 0;
4     while (b) {
5         if (b & 1) {
6             ans = (ans + a) % m;
7             b--;
8         }
9         b >>= 1;
10        a = (a + a) % m;
11    }
12    return ans;
13 }
14 ll qpow(ll a, ll b, ll m) {
15     ll ans = 1;
16     a %= m;
17     while (b) {
18         if (b & 1) {
19             ans = mul(ans, a, m);
```

```

20     b--;
21     }
22     b >>= 1;
23     a = mul(a, a, m);
24     }
25     return ans;
26 }
27
28 bool Miller_Rabin(ll n) {
29     if (n == 2)
30         return true;
31     if (n < 2 || !(n & 1))
32         return false;
33     ll a, m = n - 1, x, y;
34     int k = 0;
35     while ((m & 1) == 0) {
36         k++;
37         m >>= 1;
38     }
39     for (int i = 0; i < Times; i++) {
40         a = rand() % (n - 1) + 1;
41         x = qpow(a, m, n);
42         for (int j = 0; j < k; j++) {
43             y = mul(x, x, n);
44             if (y == 1 && x != 1 && x != n - 1)
45                 return false;
46             x = y;
47         }
48         if (y != 1)
49             return false;
50     }
51     return true;
52 }
53
54 int main() {
55     ll a, b;
56     while (~scanf("%lld%lld", &a, &b)) {
57         if (a == 0) {
58             if (b % 4 == 3 && Miller_Rabin(b))
59                 puts("Yes");
60             else
61                 puts("No");
62         } else {
63             ll t = a * a + 2 * b * b;
64             if (Miller_Rabin(t))
65                 puts("Yes");
66             else
67                 puts("No");
68         }
69     }
70     return 0;
71 }

```

### Problem B Gaussian Prime Factors (POJ 3361)

求一个数的高斯素因子。

#### Solution

我们根据高斯素数的判定原理，我们只需要将输入的数进行质因数分解，如果得到的质数是  $4x + 3$  的形式就保存，不是就枚举找到  $a$ ，求得  $b$  看  $b$  是否是整数，找到  $a$  和  $b$ ，保存  $a + bi$  和  $a - bi$ ，最后排序输出即可。

```

1  typedef long long ll;
2  struct complex
3  {
4      ll a, b;
5      char op;
6  } c[1100];
7  ll cp;
8  bool cmp(complex m, complex n)
9  {
10     bool ret = 0;

```

```

11     if (m.a < n.a)
12         ret = 1;
13     if (m.a == n.a && m.b < n.b)
14         ret = 1;
15     if (m.a == n.a && m.b == n.b && m.op == '+' && n.op == '-')
16         ret = 1;
17     return ret;
18 }
19 void getResult(ll p)
20 {
21     ll i, j;
22     if ((p - 3) % 4) {
23         for (i = 1;; i++) {
24             j = ll(sqrt(double(p - i * i)));
25             if (i * i + j * j == p) {
26                 c[cp].a = i, c[cp].b = j, c[cp++].op = '+';
27                 c[cp].a = i;
28                 c[cp].b = j, c[cp++].op = '-';
29                 break;
30             }
31         }
32     }
33     else
34         c[cp].a = p, c[cp].b = 0, c[cp++].op = '*';
35 }
36 int main()
37 {
38     ll in, num = 0, i, j, k, tmp, isFir;
39     while (scanf("%lld", &in) != EOF) {
40         tmp = in;
41         isFir = 1;
42         cp = 0;
43         printf("Case #%lld: ", ++ num);
44
45         for (i = 2; i * i < tmp; i++) {
46             if (tmp % i == 0) {
47                 getResult(i);
48                 while (tmp % i == 0)
49                     tmp /= i;
50             }
51         }
52         if (tmp != 1)
53             getResult(tmp);
54         sort(c, c + cp, cmp);
55         for (i = 0; i < cp; i++) {
56             if (i)
57                 printf(", ");
58             if (!c[i].b)
59                 printf("%lld", c[i].a);
60             else if (c[i].b == 1)
61                 printf("%lld%cj", c[i].a, c[i].op);
62             else
63                 printf("%lld%c%lldj", c[i].a, c[i].op, c[i].b);
64         }
65         printf("\n");
66     }
67     return 0;
68 }

```

## OX100 杂项

### OX101 哥德巴赫猜想及其拓展

#### 哥德巴赫猜想

哥德巴赫猜想认为任一大于2的偶数 ( $n \geq 4$ )，都可表示成两个素数之和。



尽管如今仍未得到证明，但是根据那群科学家的验证，如果有一个偶数不能拆分，那么至少是一个几百位的偶数，因此我们在竞赛的小数据范围内，所有的拆分一定是可以实现的

拓展

- 任何一个大于5的奇数都可以表示为三个素数之和
- 任何一个大于8的整数都可以表示为四个素数之和

Ox102 幂级数展开式常用公式

$$\begin{aligned}e^x &= \sum_{n=0}^{\infty} \frac{1}{n!} x^n = 1 + x + \frac{1}{2!} x^2 + \dots + \frac{1}{n!} x^n + \dots, x \in (-\infty, +\infty) \\ \sin x &= \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1} = x - \frac{1}{3!} x^3 + \frac{1}{5!} x^5 - \dots + \frac{(-1)^n}{(2n+1)!} x^{2n+1} + \dots, x \in (-\infty, +\infty) \\ \cos x &= \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} x^{2n} = 1 - \frac{1}{2!} x^2 + \frac{1}{4!} x^4 - \dots + \frac{(-1)^n}{(2n)!} x^{2n} + \dots, x \in (-\infty, +\infty) \\ \ln(1+x) &= \sum_{n=0}^{\infty} \frac{(-1)^n}{n+1} x^{n+1} = x - \frac{1}{2} x^2 + \frac{1}{3} x^3 - \dots + \frac{(-1)^n}{n+1} x^{n+1} + \dots, x \in (-1, 1] \\ \frac{1}{1-x} &= \sum_{n=0}^{\infty} x^n = 1 + x + x^2 + x^3 + \dots + x^n + \dots, x \in (-1, 1) \\ \frac{1}{1+x} &= \sum_{n=0}^{\infty} (-1)^n x^n = 1 - x + x^2 - x^3 + \dots + (-1)^n x^n + \dots, x \in (-1, 1) \\ (1+x)^\alpha &= 1 + \sum_{n=1}^{\infty} \frac{\alpha(\alpha-1)\dots(\alpha-n+1)}{n!} x^n = 1 + \alpha x + \frac{\alpha(\alpha-1)}{2!} x^2 + \dots + \frac{\alpha(\alpha-1)\dots(\alpha-n+1)}{n!} x^n + \dots, x \in (-1, 1) \\ \arctan x &= \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1} x^{2n+1} = x - \frac{1}{3} x^3 + \frac{1}{5} x^5 - \dots + \frac{(-1)^n}{2n+1} x^{2n+1} + \dots, x \in (-1, 1) \\ \arcsin x &= x + \frac{1}{6} x^3 + \frac{3}{40} x^5 + \dots, x \in (-1, 1) \\ \tan x &= x + \frac{1}{3} x^3 + \frac{2}{15} x^5 + \frac{17}{315} x^7 + \frac{62}{2835} x^9 + \frac{1382}{155925} x^{11} + \dots, x \in (-1, 1) \\ \sec x &= 1 + \frac{1}{2} x^2 + \frac{5}{24} x^4 + \frac{61}{720} x^6 + \dots, x \in (-\frac{\pi}{2}, \frac{\pi}{2}) \\ \csc x &= \frac{1}{x} + \frac{1}{6} x + \frac{7}{360} x^3 + \frac{31}{15120} x^5 + \dots, x \in (0, \pi)\end{aligned}$$

Ox103 各进制整数的位数

十进制数n的位数 $\log_{10}(n)$

以base为基数的进制数n的位数 $\log_{base}(n) = \frac{\log_{10}(n)}{\log_{10}(base)}$

n! 的位数:

$$ans = \frac{\log_{10}(1) + \log_{10}(2) + \dots + \log_{10}(n)}{\log_{10}(base)}$$

Ox104 判断组合数的奇偶性

判断组合数  $C(n, m)$  的奇偶性

当 `n & m == m` 为奇数，反之就是偶数

Ox105 特征根法求数列通项公式

特征根法是解常系数线性微分方程的一种通用方法。

特征根法也可用于通过数列的递推公式求通项公式，原理与微分方程相同。

对形如  $a_{n+2} = pa_{n+1} + qa_n$  的递推式中，显然可以把原式转化为  $a_{n+2} - x_1a_{n+1} = x_2(a_{n+1} - x_1a_n)$

那显然x1, x2为方程 $x^2$ -px-q=0的两根,由此我们可以简单推出特征根数列的通项公式。

对形如  $a_{n+2} = pa_{n+1} + qa_n$  的递推式中 (p, q为常数) 有  $a_n = C_1\alpha^n + C_2\beta^n$  ( $\alpha, \beta$ 为特征解, C1, C2为常数)

- 竞赛例题选讲

Problem A text1

在数列an中, a1=-1,a2=2, 当n∈N+时, 存在  $a_{n+2} = 5a_{n+1} - 6a_n$  , 求数列的通项?

$\because a_{n+2} = 5a_{n+1} - 6a_n$  (将  $a_{n+2}$  设为  $x^2$  , 将  $a_{n+1}$  设为x, 将  $a_n$  设为1)

特征方程为  $x^2 - 5x + 6 = 0$  , 易解得特征解为x1=2, x2=3

则 $\alpha=2, \beta=3, a_n = C_12^n + C_23^n$  ,代入a1=-1, a2=2,

$$\text{解出 } C_1 = -\frac{5}{2} \quad C_2 = \frac{4}{3}$$

得： $a_n = -\frac{5}{2} \cdot 2^n + \frac{4}{3} \cdot 3^n$

Problem B text2

设数列的前n项和为  $S_n$  , 已知  $a_n = \frac{S_{n+1} - 2}{4}$  ,a1= 1,试求  $a_n$  的通项公式?

有个Sn, 不过没关系, 消掉依然特征根。

$\because a_1 = 2, S_{n+1} = 4a_n + 2, S_2 = a_1 + a_2 = 4a_1 + 2$

易得  $a_2 = 5$  , 又有  $S_{n+1} = 4a_n + 2, S_{n+2} = 4a_{n+1} + 2$  两式相减

$a_{n+2} = 4a_{n+1} - 4a_n$  则其特征方程为  $x^2 - 4x + 4 = 0$  解得x1=x2=2

两根相等时, 我们这样设, 设  $a_n = (C_1 + nC_2)2^n$  ,代入a1=2, a2=5

我们就解得  $C_1 = -\frac{1}{4}, C_2 = \frac{3}{4}$ ,

故  $a_n = (-\frac{1}{4} + \frac{3n}{4}) * 2^n = (3n - 1)2^{n-2}$

- 竞赛例题选讲

Problem A The Nth Item (2019-ACM-ICPC-南昌区网络赛 H)

已知:  $f(n) = 3 \cdot f(n - 1) + 2 \cdot f(n - 2), (n \geq 2)$ , 求 $f(n) \pmod{998244353}$ 。  $T$ 组询问, 其中 $T \leq 10^7, n \leq 10^{18}$ 。

Solution

我们利用上面的特征根法求得通项公式为

$$a_n = \frac{\sqrt{17}}{17} \cdot \left( \left( \frac{3+\sqrt{17}}{2} \right)^n - \left( \frac{3-\sqrt{17}}{2} \right)^n \right)$$

$\sqrt{17} \pmod{998244353}$ 我们可以用二次剩余求得。然后就可以用快速幂在  $O(\log(n))$  的时间复杂度内求得  $a_n$  。但是因为 $T \leq 10^7$ , 所以还需优化。

$n \leq 10^{18}$ , 我们通过欧拉降幂之后得到  $n \leq 10^9$

我们令  $k = \lfloor \sqrt{n} \rfloor$  则:

$$n = k \cdot t + r$$

$$x^n = x^{k \cdot t + r} \Leftrightarrow x^n = x^{k \cdot t} + x^r (t, r \leq k)$$

然后我们只需要预处理出  $x^r, r \leq k$  以及  $x^{k \cdot t}, t \leq k$ , 这样我们的 $x^n$  就可以  $O(1)$  查询了, 而不用借助快速幂。

Code

```
1 #define int long long
2 const int maxn = 100005, INF = 0x3f3f3f3f;
3 const int mod = 998244353;
4 int inv17;
5 //求解x^2=n(mod p),即x=sqrt(n)(mod p) O(sqrt(p))
6 /*类似复数 单位元为w(复数的单位元为-1)*/
7 struct Complex {
8     int x, y, w;
9     Complex() {}
10     Complex(int x, int y, int w) : x(x), y(y), w(w) {}
11 };
12 /*类复数乘法 */
13 Complex mul(Complex a, Complex b, int p) {
14     Complex ans;
15     ans.x = (a.x * b.x % p + a.y * b.y % p * a.w % p) % p;
16     ans.y = (a.x * b.y % p + a.y * b.x % p) % p;
17     ans.w = a.w;
18     return ans;
19 }
20 /*类复数快速幂 */
21 Complex Complexfpow(Complex a, int b, int mod) {
22     Complex ans = Complex(1, 0, a.w);
23
24     while (b) {
25         if (b & 1)
26             ans = mul(ans, a, mod);
27
28         a = mul(a, a, mod);
29         b >>= 1;
30     }
31
32     return ans;
```

```

33 }
34 int qpow(int a, int b, int mod) {
35     int ans = 1;
36     a %= mod;
37
38     while (b) {
39         if (b & 1)
40             (ans *= a) %= mod;
41
42         (a *= a) %= mod;
43         b >>= 1;
44     }
45
46     return ans;
47 }
48 /*求解x^2=n(mod p) */
49 int solve(int n, int p) {
50     n %= p;
51
52     if (n == 0)
53         return 0;
54
55     if (p == 2)
56         return n;
57
58     if (qpow(n, (p - 1) / 2, p) == p - 1)
59         return -1; /*勒让德定理判断n不是p的二次剩余 */
60     mt19937 rnd(time(0)); //更加高效的STL自带随机数
61     int a, t, w;
62     do {
63         a = rnd() % p;
64         t = a * a - n;
65         w = (t % p + p) % p; /*构造w=a^2-n */
66     } while (qpow(w, (p - 1) / 2, p) != p - 1); /*找到一个w不是p的二次剩余 */
67
68     Complex ans = Complex(a, 1, w);
69     ans = Complexfpow(ans, (p + 1) / 2, p); /*答案为(a+w)^{(p+1)/2} */
70     return ans.x;
71 }
72 pair<int, int> bit1[maxn], bit2[maxn];
73 signed main() {
74     ios::sync_with_stdio(false);
75     cin.tie(0);
76     inv17 = qpow(17, mod - 2, mod);
77     int x = solve(17, mod); //二次剩余
78     int lim = ceil(sqrt(1e9));
79     int s1 = (3 + x) % mod * qpow(2, mod - 2, mod) % mod,
80         s2 = (3 - x) % mod * qpow(2, mod - 2, mod) % mod;
81     bit1[0].first = bit1[0].second = bit2[0].first = bit2[0].second = 1;
82     for (int i = 1; i <= lim; i++) { //预处理
83         bit1[i].first = bit1[i - 1].first * s1 % mod;
84         bit1[i].second = bit1[i - 1].second * s2 % mod;
85         bit2[i].first = qpow(s1, i * lim, mod);
86         bit2[i].second = qpow(s2, i * lim, mod);
87     }
88     int q, n;
89     cin >> q >> n;
90     int ans = 0;
91     while (q--) {
92         int tmp = 0;
93         if (n == 0)
94             tmp = 0;
95         else if (n == 1)
96             tmp = 1;
97         else {
98             int t = n % (mod - 1);
99             int t2 = t / lim, t1 = t % lim;
100             tmp = (bit1[t1].first * bit2[t2].first % mod - bit1[t1].second * bit2[t2].second % mod) % mod * x % mod *
101                 inv17 % mod;
102         }
103         tmp = (tmp + mod) % mod;

```

```
104         n = tmp * tmp ^ n;
105         ans ^= tmp;
106         // cout<<n<<endl;
107     }
108     cout << (ans + mod) % mod << endl;
109     return 0;
110 }
```

## Ox110 后记

原本只是打算简单花费一天的时间整理一下我学过的数论知识，加上最近刚刚看完《初等数论及其应用》，以及部分的《具体数学》，学到了一些奇怪的新姿势，想要记录一下。发现最后越写越多，从最开始的五千字，一万字，写到了最后的四万五千字，十四万七千的字符，涵盖的内容也越来越多，越来越多的奇技淫巧被我填充了进来，我几乎翻遍了全网的数论博客，抽取出了很多人总结的经验之谈。为了使得整篇不那么枯燥，我为每个算法都精选了 **经典例题 / ICPC / CCPC / NOIP / NOI 真题**，并附上我的详细题解和部分代码。我现在几乎可以负责任地说：**算法竞赛，数论相关，这一篇就够了。**

参考资料：

- 《初等数论及其应用》
- 《具体数学》
- 《算法竞赛进阶指南》
- 《简明数论》
- **【学习笔记】数论、数学一常见定理、结论、性质汇总**
- **神O的数论全家桶**
- **数学笔记：数论**
- **【知识总结】数论全家桶**
- **初等数论学习笔记（一）sola**
- **莫比乌斯反演也是数论**
- **ACM中的数学问题合集**
- **ACM数论基本定理**
- **关于数论的一些总结**
- **夜深人静写算法（三）- 初等数论入门@英雄从哪里来**
- **数论总结**
- **ACM数论总结**

1. 本小节部分内容来源于二项式反演及其应用@GXZlegend ↩