数论

炼石计划

2023 年 11 月 4 日

整除

整除的定义

设 $a, b \in \mathbf{Z}$，$a \neq 0$。如果 $\exists q \in \mathbf{Z}$，使得 $b = aq$，那么就说 b 可被 a 整除，记作 $a \mid b$；b 不被 a 整除记作 $a \nmid b$。

整除的性质：

- $a \mid b \iff -a \mid b \iff a \mid -b \iff |a| \mid |b|$
- $a \mid b \wedge b \mid c \implies a \mid c$
- $a \mid b \wedge a \mid c \iff \forall x, y \in \mathbb{Z}, a \mid (xb + yc)$
- $a \mid b \wedge b \mid a \implies b = \pm a$
- 设 $m \neq 0$，那么 $a \mid b \iff ma \mid mb$。
- 设 $b \neq 0$，那么 $a \mid b \implies |a| \leq |b|$。
- 设 $a \neq 0, b = qa + c$，那么 $a \mid b \iff a \mid c$。

约数

约数（因数）

若 $a \mid b$，则称 b 是 a 的倍数，a 是 b 的约数。

- 0 是所有非 0 整数的倍数。对于整数 $b \neq 0$，b 的约数只有有限个。
- 平凡约数（平凡因数）：对于整数 $b \neq 0$，± 1、$\pm b$ 是 b 的平凡约数。当 $b = \pm 1$ 时，b 只有两个平凡约数。
- 对于整数 $b \neq 0$，b 的其他约数称为真约数（真因数、非平凡约数、非平凡因数）。

约数的性质：

- 设整数 $b \neq 0$。当 d 遍历 b 的全体约数的时候，$\frac{b}{d}$ 也遍历 b 的全体约数。
- 设整数 $b > 0$，则当 d 遍历 b 的全体正约数的时候，$\frac{b}{d}$ 也遍历 b 的全体正约数。

在具体问题中，**如果没有特别说明，约数总是指正约数**。

带余数除法

余数的定义

设 a, b 为两个给定的整数，$a \neq 0$。设 d 是一个给定的整数。那么，一定存在唯一的一对整数 q 和 r，满足 $b = qa + r, d \leq r < |a| + d$。无论整数 d 取何值，r 统称为余数。$a \mid b$ 等价于 $a \mid r$。

- 最小非负余数：一般情况下，d 取 0，此时等式 $b = qa + r, 0 \leq r < |a|$ 称为带余数除法（带余除法）。
- 余数的最小正余数：d 取 1。即 $b = qa + r, 1 \leq r < |a| + 1$。

带余数除法的余数只有最小非负余数。**如果没有特别说明，余数总是指最小非负余数**。

余数的性质：

- 任一整数被正整数 a 除后，余数一定是且仅是 0 到 $(a-1)$ 这 a 个数中的一个。
- 相邻的 a 个整数被正整数 a 除后，恰好取到上述 a 个余数。特别地，一定有且仅有一个数被 a 整除。

最大公约数与最小公倍数

最大公约数 (Greatest Common Divisor, gcd)

一组整数的公约数，是指同时是这组数中每一个数的约数的数。± 1 是任意一组整数的公约数。一组整数的最大公约数，是指所有公约数里面最大的一个。

如果我们已知两个数 a 和 b，如何求出二者的最大公约数呢？

最大公约数与最小公倍数

最大公约数 (Greatest Common Divisor, gcd)

一组整数的公约数，是指同时是这组数中每一个数的约数的数。± 1 是任意一组整数的公约数。一组整数的最大公约数，是指所有公约数里面最大的一个。

如果我们已知两个数 a 和 b，如何求出二者的最大公约数呢？

欧几里得算法

不妨设 $a > b$ 我们发现如果 b 是 a 的约数，那么 b 就是二者的最大公约数。下面讨论不能整除的情况，即 $a = b \times q + r$，其中 $r < b$。我们通过证明可以得到 $\gcd(a, b) = \gcd(b, a \bmod b)$，证明在后页。

欧几里得算法（辗转相除法）

设 $a = bk + c$，显然有 $c = a \bmod b$。设 $d \mid a$，$d \mid b$，则 $c = a - bk$，$\frac{c}{d} = \frac{a}{d} - \frac{b}{d}k$。
由右边的式子可知 $\frac{c}{d}$ 为整数，即 $d \mid c$，所以对于 a, b 的公约数，它也会是 $b, a \bmod b$ 的公约数。反过来也需要证明：
设 $d \mid b$，$d \mid (a \bmod b)$，我们还是可以像之前一样得到以下式子

$$\frac{a \bmod b}{d} = \frac{a}{d} - \frac{b}{d}k, \quad \frac{a \bmod b}{d} + \frac{b}{d}k = \frac{a}{d}$$

因为左边式子显然为整数，所以 $\frac{a}{d}$ 也为整数，即 $d \mid a$，所以 $b, a \bmod b$ 的公约数也是 a, b 的公约数。
既然两式公约数都是相同的，那么最大公约数也会相同。
所以得到式子 $\gcd(a, b) = \gcd(b, a \bmod b)$ ，那么就得到了关于两个数的最大公约数的一个递归求法，时间复杂度为 $\mathcal{O}(\log \max(a, b))$ 。

更相减损术

大整数取模的时间复杂度较高，而加减法时间复杂度较低。针对大整数，我们可以用加减代替乘除求出最大公约数。

更相减损法

已知两数 a 和 b，求 $\gcd(a,b)$。不妨设 $a \geq b$，

- 若 $a = b$，则 $\gcd(a,b) = a = b$。
- 否则，$\forall d \mid a, d \mid b$，可以证明 $d \mid a - b$。

因此，a 和 b 的 **所有公因数都是** $a - b$ 和 b 的公因数，$\gcd(a,b) = \gcd(a-b,b)$。

如果 $a \gg b$，更相减损术的 $\mathcal{O}(n)$ 复杂度将会达到最坏情况。

更相减损术

考虑一个优化

- 若 $2 \mid a, 2 \mid b$，$\gcd(a, b) = 2\gcd\left(\frac{a}{2}, \frac{b}{2}\right)$。
- 否则，若 $2 \mid a$（$2 \mid b$ 同理），因为 $2 \mid b$ 的情况已经讨论过了，所以 $2 \nmid b$。因此 $\gcd(a, b) = \gcd\left(\frac{a}{2}, b\right)$。

优化后的算法（即 Stein 算法）时间复杂度是 $\mathcal{O}(\log n)$。

复杂度证明

若 $2 \mid a$ 或 $2 \mid b$，每次递归至少会将 a, b 之一减半。
否则，$2 \mid a - b$，回到了上一种情况。

最小公倍数 (Least Common Multiple, LCM)

最小公倍数的定义

一组整数的公倍数，是指同时是这组数中每一个数的倍数的数。0 是任意一组整数的公倍数。一组整数的最小公倍数，是指所有正的公倍数里面，最小的一个数。

设 $a = p_1^{k_{a_1}} p_2^{k_{a_2}} \cdots p_s^{k_{a_s}}$，$b = p_1^{k_{b_1}} p_2^{k_{b_2}} \cdots p_s^{k_{b_s}}$ 我们发现，对于两个数的情况，二者的最大公约数等于

$$p_1^{\min(k_{a_1},k_{b_1})} p_2^{\min(k_{a_2},k_{b_2})} \cdots p_s^{\min(k_{a_s},k_{b_s})}$$

最小公倍数等于

$$p_1^{\max(k_{a_1},k_{b_1})} p_2^{\max(k_{a_2},k_{b_2})} \cdots p_s^{\max(k_{a_s},k_{b_s})}$$

由于 $k_a + k_b = \max(k_a, k_b) + \min(k_a, k_b)$
所以得到结论是 $\gcd(a, b) \times \mathrm{lcm}(a, b) = a \times b$ 要求两个数的最小公倍数，先求出最大公约数即可。

CF 511(Div.1) C.Region Separation

一棵 n 个点的树，每个点有权值 a_i 。你想砍树。
你可以砍任意多次，每次会选择某些边断开，需要满足砍完后每个连通块的权值和是相等的。求有多少种砍树方案。
$n \leq 10^6, 1 \leq a_i \leq 10^9$

CF 511(Div.1) C.Region Separation

先假设只砍一次。令所有点权和为 S，那么假设要砍成 k 个连通块，则每个连通块的权值和均为 $\frac{S}{k}$。
考虑如何得到砍的方案，以 1 号点为根 dfs，若当前点 i 的子树和满足 $\frac{S}{k}|sum_i$，则当前子树可以砍下来。若最后恰好砍了 k 次，那么就得到了一个合法的砍树方案。这等价于 $\sum_{i=1}^{n}[\frac{S}{k}|sum_i] = k$。不难看出这个**对应且仅对应**一种方案。如果不足 k，那么就没有那么多个点可以分；多于 k 的情况是不可能的，因为总和不够分配。这个式子还不够优秀，我们转化一下：

$$
\begin{aligned}
[\frac{S}{k}|sum_i] &= [S|k \times sum_i] \\
&= [\frac{S}{\gcd(S, sum_i)}|k \times \frac{sum_i}{\gcd(S, sum_i)}] \\
&= [\frac{S}{\gcd(S, sum_i)}|k] \qquad (\because \frac{S}{\gcd(S, sum_i)} \perp \frac{sum_i}{\gcd(S, sum_i)})
\end{aligned}
$$

CF 511(Div.1) C.Region Separation

然后 $\sum_{i=1}^{n}[\frac{S}{k}|sum_i] = k$ 就变成了 $\sum_{i=1}^{n}[\frac{S}{\gcd(S, sum_i)}|k] = k$

显然这个我们可以枚举倍数在 $O(n \ln n)$ 的时间内解决（注意 $k \leq n$ ）

那么如果砍多次呢？可以看出如果第一次砍成了 x 块，那么第二次砍成的块数 y 必须满足 $x|y$ 。

因为你之后的权值只能比之前分的更多，且每个联通块的权值是之前的一个因子。这部分也可以 $\mathcal{O}(n \ln n)$ 算。

扩展欧几里得算法 (Extended Euclidean, EXGCD)

扩展欧几里得算法，常用于求 $ax+by=\gcd(a,b)$ 的一组可行解。
设

$$ax_1+by_1=\gcd(a,b)$$

$$bx_2+(a \bmod b)y_2=\gcd(b,a \bmod b)$$

由欧几里得定理可知：$\gcd(a,b)=\gcd(b,a \bmod b)$ 所以
$ax_1+by_1=bx_2+(a \bmod b)y_2$ 又因为 $a \bmod b=a-(\lfloor \frac{a}{b} \rfloor \times b)$
所以 $ax_1+by_1=bx_2+(a-(\lfloor \frac{a}{b} \rfloor \times b))y_2$

$$ax_1+by_1=ay_2+bx_2-\lfloor \frac{a}{b} \rfloor \times by_2=ay_2+b(x_2-\lfloor \frac{a}{b} \rfloor y_2)$$

因为 $a=a,b=b$，所以 $x_1=y_2,y_1=x_2-\lfloor \frac{a}{b} \rfloor y_2$ 将 x_2,y_2 不断代入递归求解直至 $\gcd$ 为 0 递归 x=1, y=0 回去求解。

扩展欧几里得算法 (Extended Euclidean, EXGCD)

首先，当 $x = 1$，$y = 0$，$x_1 = 0$，$y_1 = 1$ 时有：

$$\begin{cases} ax + by & = a \\ ax_1 + by_1 & = b \end{cases}$$

已知 $a \bmod b = a - (\lfloor \frac{a}{b} \rfloor \times b)$，下面令 $q = \lfloor \frac{a}{b} \rfloor$。参考迭代法求 gcd，每一轮的迭代过程可以表示为：

$$(a, b) \to (b, a - qb)$$

将迭代过程中的 a 替换为 $ax + by = a$，b 替换为 $ax_1 + by_1 = b$，可以得到：

$$\begin{cases} ax + by & = a \\ ax_1 + by_1 & = b \end{cases}$$

$$\to \begin{cases} ax_1 + by_1 & = b \\ a(x - qx_1) + b(y - qy_1) & = a - qb \end{cases}$$

据此就可以得到迭代法求 exgcd。

扩展欧几里得算法 (Extended Euclidean, EXGCD)

另一种推导的写法：

$$\begin{aligned} ax + by &= \gcd(a, b) \\ &= \gcd(b, a \bmod b) \\ &\Rightarrow bx + (a \bmod b)\ y \\ &= bx + (a - \lfloor \frac{a}{b} \rfloor\ b)\ y \\ &= ay + b\ (x - \lfloor \frac{a}{b} \rfloor\ y) \end{aligned}$$

```
void Exgcd(ll a, ll b, ll &x, ll &y) {
    if (!b) x = 1, y = 0;
    else Exgcd(b, a % b, y, x), y -= a / b * x;
}
```

洗牌

对于扑克牌的一次洗牌是这样定义的，将一叠 n（n 为偶数）张扑克牌平均分成上下两叠，取下面一叠的第一张作为新的一叠的第一张，然后取上面一叠的第一张作为新的一叠的第二张，再取下面一叠的第二张作为新的一叠的第三张……如此交替直到所有的牌取完。

问洗 m 次之后第 k 张牌是开始的第几张。

$n, m \leq 10^{10}$

洗牌

很简单的一道例题。
不难发现一次洗牌后第 i 张的位置变成了 $(2 \times i) \bmod n$。
那么我们解方程 $x \times 2^m \equiv k \pmod{n}$ 即可。
解模方程很容易用扩欧解决，转化为 $x \times 2^m + y \times n = k$ 求解就行。
复杂度为 $\mathcal{O}(\log n + \log m)$ 。

互素

互素的定义

- 两个整数互素（既约）：若 $\gcd(a_1, a_2) = 1$，则称 a_1 和 a_2 互素（既约）。
- 多个整数互素（既约）：若 $\gcd(a_1, \ldots, a_k) = 1$，则称 $a_1, \ldots, a_k$ 互素（既约）。

多个整数互素，不一定两两互素。例如 6、10 和 15 互素，但是任意两个都不互素。

裴蜀定理

设 a, b 是不全为零的整数，则存在整数 x, y，使得 $ax + by = \gcd(a, b)$.

素数与合数

素数与合数定义

- 设整数 $p \neq 0, \pm 1$。如果 p 除了平凡约数外没有其他约数，那么称 p 为素数（不可约数）。
- 若整数 $a \neq 0, \pm 1$ 且 a 不是素数，则称 a 为合数。

p 和 $-p$ 总是同为素数或者同为合数。**如果没有特别说明，素数总是指正的素数**。一些性质：

- 大于 1 的整数 a 是合数，等价于 a 可以表示为整数 d 和 e $(1 < d, e < a)$ 的乘积。
- 大于 1 的整数 a 一定可以表示为素数的乘积。
- 对于合数 a，一定存在素数 $p \leq \sqrt{a}$ 使得 $p \mid a$。
- 所有大于 3 的素数都可以表示为 $6n \pm 1$ 的形式。

算术基本定理

算术基本引理

设 p 是素数，$p \mid a_1 a_2$，那么 $p \mid a_1$ 和 $p \mid a_2$ 至少有一个成立。
算术基本引理是素数的本质属性，也是素数的真正定义。

算术基本定理（唯一分解定理）

设正整数 a，那么必有表示：

$$a = p_1 p_2 \cdots p_s, \qquad p_j (1 \le j \le s) \text{是素数}$$

并且不计次序的意义下，该表示唯一。将上述表示中，相同的素数合并，可得标准素因数分解式：

$$a = {p_1}^{\alpha_1} {p_2}^{\alpha_2} \cdots {p_s}^{\alpha_s}, \qquad p_1 < p_2 < \cdots < p_s$$

算术基本定理和算术基本引理，两个定理是等价的。

同余

同余的定义

设整数 $m \neq 0$。若 $m \mid (a-b)$，称 m 为模数（模），a 同余于 b 模 m，b 是 a 对模 m 的剩余。记作 $a \equiv b \pmod{m}$。

否则，a 不同余于 b 模 m，b 不是 a 对模 m 的剩余。记作 $a \not\equiv b \pmod{m}$。这样的等式，称为模 m 的同余式，简称同余式。
根据整除的性质，上述同余式也等价于 $a \equiv b \pmod{(-m)}$，一般认为模数为正。
式中的 b 是 a 对模 m 的剩余，这个概念与余数完全一致。通过限定 b 的范围，相应的有 a 对模 m 的最小非负剩余、绝对最小剩余、最小正剩余。

同余

同余的性质：

- 自反性：$a \equiv a \pmod{m}$。
- 对称性：若 $a \equiv b \pmod{m}$，则 $b \equiv a \pmod{m}$。
- 传递性：若 $a \equiv b \pmod{m}, b \equiv c \pmod{m}$，则 $a \equiv c \pmod{m}$。
- 线性运算：若 $a, b, c, d \in \mathbf{Z}, m \in \mathbf{N}^*, a \equiv b \pmod{m}, c \equiv d \pmod{m}$ 则：
  - $a \pm c \equiv b \pm d \pmod{m}$。
  - $a \times c \equiv b \times d \pmod{m}$。
- 若 $a, b \in \mathbf{Z}, k, m \in \mathbf{N}^*, a \equiv b \pmod{m}$，则 $ak \equiv bk \pmod{mk}$。
- 若 $a, b \in \mathbf{Z}, d, m \in \mathbf{N}^*, d \mid a, d \mid b, d \mid m$，则当 $a \equiv b \pmod{m}$ 成立时，有 $\frac{a}{d} \equiv \frac{b}{d} \left(\mathrm{mod}\ \frac{m}{d}\right)$。
- 若 $a, b \in \mathbf{Z}, d, m \in \mathbf{N}^*, d \mid m$，则当 $a \equiv b \pmod{m}$ 成立时，有 $a \equiv b \pmod{d}$。
- 若 $a, b \in \mathbf{Z}, d, m \in \mathbf{N}^*$，则当 $a \equiv b \pmod{m}$ 成立时，有 $\gcd(a, m) = \gcd(b, m)$。若 d 能整除 m 及 a, b 中的一个，则 d 必定能整除 a, b 中的另一个。

乘法逆元

逆元定义

若 $a \times x \equiv 1 \pmod{b}$，且 a 与 b 互质，那么我们就能定义：x 为 a 的逆元，记为 a^{-1}，所以我们也可以称 x 为 a 在 $\bmod b$ 意义下的倒数。

求逆元的几个方式：

- 拓展欧几里得：

  求解线性同余方程 $a \times x \equiv c \pmod{b}$ 的 $c = 1$ 的情况。我们就可以转化为解 $a \times x + b \times y = 1$ 这个方程。

- 快速幂：

费马小定理

若 p 为素数，a 为正整数，且 a、p 互质。则有 $a^{p-1} \equiv 1 \pmod{p}$

然后我们就可以推得 $a^{p-2} \pmod{p}$ 为 a 的逆元.

乘法逆元

- **线性算法**:
  用于求一连串数字对于一个 $\bmod p$ 的逆元。首先我们有 $1^{-1} \equiv 1 \pmod{p}$ 然后设 $p = k \times i + r, (1 < r < i < p)$ 也就是 k 是 p/i 的商，r 是余数。
  再将这个式子放到 $\pmod{p}$ 意义下就会得到:

$$k \times i + r \equiv 0 \pmod{p}$$

  然后乘上 i^{-1}, r^{-1} 就可以得到:

$$\begin{aligned} k \times r^{-1} + i^{-1} &\equiv 0 \pmod{p} \\ i^{-1} &\equiv -k \times r^{-1} \pmod{p} \\ i^{-1} &\equiv -\lfloor \frac{p}{i} \rfloor \times (p \bmod i)^{-1} \pmod{p} \end{aligned}$$

  这个就给了我们一个递推式。

乘法逆元

- 阶乘逆元：
  因为有如下一个递推关系。

$$inv[i+1] = \frac{1}{(i+1)!}$$

$$inv[i+1] \times (i+1) = \frac{1}{i!} = inv[i]$$

所以我们可以求出 $n!$ 的逆元，然后逆推，就可以求出 $1...n!$ 所有的逆元了。递推式为 $inv[i+1] \times (i+1) = inv[i]$

所以我们可以求出 $\forall i, i!, \frac{1}{i!}$ 的取值了。然后这个也可以导出 $\frac{1}{i} \pmod p$ 的取值，也就是

$$\frac{1}{i!} \times (i-1)! = \frac{1}{i} \pmod p$$

求解模线性方程组

问题描述

给定了 n 组除数 m_i 和余数 r_i ，通过这 n 组 (m_i, r_i) 求解一个 x ，使得 $x \bmod m_i = r_i$ 这就是**模线性方程组**。
数学形式表达就是：

$$\begin{cases} x \equiv r_1 \pmod{m_1} \\ x \equiv r_2 \pmod{m_2} \\ \quad \vdots \\ x \equiv r_n \pmod{m_n} \end{cases}$$

求解一个 x 满足上列所有方程。

需要一个中国剩余定理才能解决。

求解模线性方程组

中国剩余定理

如果 $m_1, m_2, \ldots, m_n$ 两两互质，则对于任意的整数 $r_1, r_2, \ldots, r_n$ 同余方程组有解，并且可以用如下方法来构造：

令 $M = \prod_{i=1}^{n} m_i$ ，并设 $M_i = \frac{M}{m_i}$ ，令 t_i 为 M_i 在模 m_i 意义下的逆元（也就是 $t_i \times M_i \equiv 1 \pmod{m_i}$ ）。不难发现这个 t_i 是一定存在的，因为由于前面要满足两两互质，那么 $M_i \perp m_i$ ，所以必有逆元。

那么在模 M 意义下，方程 (S) 有且仅有一个解：

$x = (\sum_{i=1}^{n} r_i t_i M_i) \bmod M$ 。

不难发现这个特解是一定满足每一个方程的，只有一个解的证明可以考虑特解 x 对于第 i 个方程，下个继续满足条件的 x 为 $x + m_i$ 。那么对于整体来说，下个满足条件的数就是 $x + \operatorname{lcm}(m_1, m_2, \ldots, m_n) = x + M$ 。

求解模线性方程组

对于模数互质的情况我们能用中国剩余定理很快的求出逆元，模数不互质的情况呢？

求解模线性方程组

对于模数互质的情况我们能用中国剩余定理很快的求出逆元，模数不互质的情况呢？考虑拓欧合并方程组，比如从 $n = 2$ 开始递推。

$$\begin{cases} x \equiv r_1 \pmod{m_1} \\ x \equiv r_2 \pmod{m_2} \end{cases}$$

也就等价于

$$\begin{cases} x = m_1 \times k_1 + r_1 \\ x = m_2 \times k_2 + r_2 \end{cases}$$

此处 $k_1, k_2 \in \mathbb{N}$ 。联立后就得到了如下一个方程：

$$m_1 \times k_1 + r_1 = m_2 \times k_2 + r_2$$

$$m_1 \times k_1 - m_2 \times k_2 = r_2 - r_1$$

我们令 $a = m_1, b = m_2, c = r_2 - r_1$ 就变成了 $ax + by = c$ 的形式，用之前讲过的 EXGCD 可以求解。首先先判断有无解。假设存在解，我们先解出一组 (k_1, k_2) ，然后带入解出 $x = x_0$ 的一个特解。我们将这个可以扩展成一个解系：

$$x = x_0 + k \times \operatorname{lcm}(m_1, m_2) \ , \ k \in \mathbb{N}$$

求解模线性方程组

解系的扩展

对于拓欧我们有时候要求对于 x or y 的最小非负整数解，这个的话我们需要将单个 (x, y) 扩展成一个解系。
如果学过不定方程的话，就可以轻易得到这个解系的，在此不过多赘述了。

$$\begin{cases} x = x_0 + db \\ y = y_0 - da \end{cases} \quad (d = \gcd(a, b) \in \mathbb{Z})$$

由于前面不定方程的结论，k_1 与其相邻解的间距为 $\frac{m_2}{\gcd(m_1, m_2)}$，又有 $x = m_1 \times k_1 + r_1$ 。所以 x 与其相邻解的距离为 $\frac{m_1 m_2}{\gcd(m_1, m_2)} = \operatorname{lcm}(m_1, m_2)$ 。
所以我们令 $M = \operatorname{lcm}(m_1, m_2), R = x_0$ 则又有新的模方程 $x \equiv R \pmod{M}$ 。然后我们就将两个方程合并成一个了，只要不断重复这个过程就能做完了。

求解模线性方程组

实现时注意细节，比如求两个数的 gcd 的时候，一定要先除再乘，防止溢出。

```
ll mod[N], rest[N];
ll Solve() {
    For (i, 1, n - 1) {
        ll a = mod[i], b = mod[i + 1], c = rest[i + 1] - rest[i], gcd = __gcd(a, b), k1, k2;
        if (c % gcd) return - 1;
        a /= gcd; b /= gcd; c /= gcd;
        Exgcd(a, b, k1, k2);

        k1 = ((k1 * c) % b + b) % b;
        mod[i + 1] = mod[i] / __gcd(mod[i], mod[i + 1]) * mod[i + 1] ;
        rest[i + 1] = (mod[i] * k1 % mod[i + 1] + rest[i]) % mod[i + 1];
    }
    return rest[n];
}
```

取模（同余）运算的一些重要的定理

费马小定理

若 p 为素数，$\gcd(a,p)=1$，则 $a^{p-1}\equiv 1 \pmod p$。
另一个形式：对于任意整数 a，有 $a^p\equiv a \pmod p$。

我们可以用剩余系证明，我们在这里用上节课讲的二项式定理归纳证明。然后 $1^p\equiv 1 \pmod p$，假设 $a^p\equiv a \pmod p$ 成立，那么通过二项式定理有

$$(a+1)^p=a^p+\binom{p}{1}a^{p-1}+\binom{p}{2}a^{p-2}+\cdots+\binom{p}{p-1}a+1$$

因为 $\binom{p}{k}=\dfrac{p(p-1)\cdots(p-k+1)}{k!}$ 对于 $1\le k\le p-1$ 成立，在模 p 意义下 $\binom{p}{1}\equiv\binom{p}{2}\equiv\cdots\equiv\binom{p}{p-1}\equiv 0 \pmod p$，那么 $(a+1)^p\equiv a^p+1 \pmod p$，将 $a^p\equiv a \pmod p$ 带入得 $(a+1)^p\equiv a+1 \pmod p$ 得证。

取模（同余）运算的一些重要的定理

费马小定理其实是欧拉定理的特殊形式，在了解欧拉定理之前，我们先了解欧拉函数（后面再详细介绍）。

欧拉函数

欧拉函数（Euler's totient function），即 $\varphi(n)$，表示的是小于等于 n 和 n 互质的数的个数。
比如说 $\varphi(1) = 1$，当 n 是质数的时候，显然有 $\varphi(n) = n - 1$。

欧拉定理

若 $\gcd(a, m) = 1$，则 $a^{\varphi(m)} \equiv 1 \pmod{m}$。

在证明这个定理之前，我们介绍一下剩余系的一些知识。

剩余系

剩余系

对于某一个特定的正整数 n，一个整数集中的数模 n 所得的余数域，称作**剩余系**。如果一个剩余系中包含了这个正整数所有可能的余数（一般地，对于任意正整数 n，有 n 个余数：$0, 1, 2, \ldots, n-1$），那么就被称为是模 n 的一个**完全剩余系**。

简化剩余系

简化剩余系也称既约剩余系或缩系，是 n 的完全剩余系中与 n 互素的数构成的子集，如果模 n 的一个剩余类里所有数都与 n 互素，就把它叫做与模 n 互素的剩余类。

欧拉定理

这个证明过程需要类似于 **构造一个与 m 互质的数列**。
设 $r_1, r_2, \cdots, r_{\varphi(m)}$ 为模 m 意义下的一个简化剩余系，则 $ar_1, ar_2, \cdots, ar_{\varphi(m)}$ 也为模 m 意义下的一个简化剩余系。（因为 a 与 n 互质并且 $\{ar_i\}$ 中任意两个元素互不相同）
所以 $r_1 r_2 \cdots r_{\varphi(m)} \equiv ar_1 \cdot ar_2 \cdots ar_{\varphi(m)} \equiv a^{\varphi(m)} r_1 r_2 \cdots r_{\varphi(m)} \pmod{m}$（因为简化剩余系唯一），左右可约去 $r_1 r_2 \cdots r_{\varphi(m)}$，即得 $a^{\varphi(m)} \equiv 1 \pmod{m}$。
那么当 m 为素数时，由于 $\varphi(m) = m - 1$，然后可得到费马小定理。

扩展欧拉定理

当 a, b, p 任意时候有

$$a^b \equiv \begin{cases} a^{b \bmod \varphi(p)} & b < \varphi(p) \\ a^{b \bmod \varphi(p) + \varphi(p)} & b \geq \varphi(p) \end{cases} \pmod{p}$$

通常用在降幂的题目当中。

「SHOI2017」相逢是问候

给出一个长度为 n 的序列 $\{a_i\}$ 以及一个数 p，现在有 m 次操作，每次操作将 $[l, r]$ 区间内的 a_i 变成 c^{a_i} 。
或者询问 $[l, r]$ 之间所有 a_i 的和对 p 取模的结果。
$n, m \leq 5 \times 10^4, p \leq 2^{14}$

「SHOI2017」相逢是问候

考虑欧拉降幂（扩展欧拉定理），然后对于这些不断叠加的指数，有如下式子

$$\begin{aligned} & c^{c^x} && \pmod p \\ \equiv\ & c^{c^x \bmod \varphi(p)+\varphi(p)} && \pmod p \\ \equiv\ & c^{c^x \bmod \varphi(\varphi(p))+\varphi(\varphi(p))\ \bmod \varphi(p)+\varphi(p)} && \pmod p \end{aligned}$$

我们发现多次操作后 $\varphi(...\varphi(p)) = 1$ 时，最高层就 $\bmod$ 变成 0，然后结果以后就不会改变了。这个次数约是 $\mathcal{O}(\log p)$ 次的，我们就有个很直观的想法。

考虑预处理每个数进行多次操作后变成的数，然后每次暴力改需要改的数。然后对于线段树每个区间，维护这个区间中的数改变次数的最小值，如果最小值 $\geq$ 当前的 φ 的层数，直接退出即可。

然后为了降低复杂度，我们在预处理的时候，对于底数 c 求一个幂次对于 p 模的值，可以利用大步小步预处理也就是 $\sqrt{p}$ 打个表，然后最后复杂度就可以做到 $O(\sqrt{p}\log^2 p + n(\log n + \log p))\log p)$ 了。

阶与原根

阶

由欧拉定理可知，对 $a \in \mathbb{Z}$，$m \in \mathbb{N}^*$，若 $(a, m) = 1$，则 $a^{\varphi(m)} \equiv 1 \pmod{m}$。因此满足同余式 $a^n \equiv 1 \pmod{m}$ 的最小正整数 n 存在，这个 n 称作 a 模 m 的阶，记作 $\delta_m(a)$ 或 $\mathrm{ord}_m(a)$.

一些性质：

- $a, a^2, \cdots, a^{\delta_m(a)}$ 模 m 两两不同余。
- 若 $a^n \equiv 1 \pmod{m}$，则 $\delta_m(a) \mid n$。
- 设 $m \in \mathbb{N}^*$，$a, b \in \mathbb{Z}$，$(a, m) = (b, m) = 1$，则 $\delta_m(ab) = \delta_m(a)\delta_m(b)$ 的充分必要条件是 $(\delta_m(a), \delta_m(b)) = 1$。
- 设 $k \in \mathbb{N}$，$m \in \mathbb{N}^*$，$a \in \mathbb{Z}$，$(a, m) = 1$，则

$$\delta_m(a^k) = \frac{\delta_m(a)}{(\delta_m(a), k)}$$

阶与原根

原根

设 $m \in \mathbb{N}^*$，$g \in \mathbb{Z}$. 若 $(g, m) = 1$，且 $\delta_m(g) = \varphi(m)$，则称 g 为模 m 的原根。即 g 满足 $\delta_m(g) = |\mathbb{Z}_m^*| = \varphi(m)$. 当 m 是质数时，我们有 $g^i \bmod m$, $0 < i < m$ 的结果互不相同。

原根判定定理

设 $m \geqslant 3, (g, m) = 1$，则 g 是模 m 的原根的充要条件是，对于 $\varphi(m)$ 的每个素因数 p，都有 $g^{\frac{\varphi(m)}{p}} \not\equiv 1 \pmod{m}$

若一个数 m 有原根，则它原根的个数为 $\varphi(\varphi(m))$ 。

原根存在定理

一个数 m 存在原根当且仅当 $m = 2, 4, p^\alpha, 2p^\alpha$，其中 p 为奇素数，$\alpha \in \mathbb{N}^*$ 。

最小原根的范围估计：有证明素数 p 的最小原根 $g_p = O\left(p^{0.25+\epsilon}\right) = \Omega(\log p)$，其中 $\epsilon > 0$ ，所以暴力找原根复杂度是能接受的。

筛法

如果我们想要知道小于等于 n 有多少个素数呢？
一个自然的想法是对于小于等于 n 的每个数进行一次质数检验，这就是筛法。对于筛法我们有很多不同的优化。

埃拉托斯特尼筛法 (Eratosthenes)

对于任意一个大于 1 的正整数 n，那么它的 x 倍就是合数 $(x > 1)$。因此可以避免很多次不必要的检测。
如果我们从小到大考虑每个数，然后同时把当前这个数的 > 1 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，时间复杂度是 $\mathcal{O}(n \ln \ln n)$ 。

如果只需要求 $< n$ 的所有质数，可以只筛到 $\sqrt{n}$ 的素数，复杂度优化为 $\mathcal{O}(n \ln \ln \sqrt{n})$ 。

筛法

埃氏筛法仍有优化空间，它会将一个合数重复多次标记。有没有什么办法省掉无意义的步骤呢？
如果能**让每个合数都只被标记一次**，那么时间复杂度就可以降到 $\mathcal{O}(n)$ 了，我们一般称其为线性筛。
对于一个合数 m 由唯一分解定理，可以分解为 $m = p_1^{r_1} \times \ldots \times p_n^{r_n}$ 其中 p_i 为质数，那么我们筛 m 的时候之前把 p_1 筛掉了，所以在枚举 i 的时候

1. 如果 i 为素数没问题，直接向后继续推（因为筛出的质数都类似 $m = p_1 \times p_2$ 的形式，所以不可能重复）。
2. 如果为合数，那么 i 可以分解成 $i = {p_1}^{r_1} \times \cdots \times {p_n}^{r_n}$ 形式其中 $p_1 \to p_n$ 是递增的，那么 p_1 是最小的那个质数。
   $i \bmod p_1 = 0$ 的时候，就不用继续枚举了，所以我们就只能筛出不大于 p_1 的质数 $\times i$。

筛法

线性筛代码如下

```
void init(int n) {
  for (int i = 2; i <= n; ++i) {
    if (!vis[i]) {
      pri[cnt++] = i;
    }
    for (int j = 0; j < cnt; ++j) {
      if (1ll * i * pri[j] > n) break;
      vis[i * pri[j]] = 1;
      if (i % pri[j] == 0) {
        // i % pri[j] == 0
        // 换言之，i 之前被 pri[j] 筛过了
        // 由于 pri 里面质数是从小到大的，所以 i乘上其他的质数的结果一定会被
        // pri[j]的倍数筛掉，就不需要在这里先筛一次，所以这里直接 break
        // 掉就好了
        break;
      }
    }
  }
}
```

CF396 B.On Sum of Fractions

我们设

- $u(n)$ 为不超过 n 的最大素数
- $v(n)$ 为大于 n 的最大素数

给定 n 求

$$\sum_{i=2}^{n} \frac{1}{u(i)v(i)}$$

$2 \leq n \leq 10^9$

CF396 B.On Sum of Fractions

拆开这个式子：

$$\sum_{i=2}^{n}\frac{1}{u(i)v(i)}=\frac{1}{2\times3}+\frac{1}{3\times5}+\frac{1}{3\times5}+\frac{1}{5\times7}+\frac{1}{5\times7}+\cdots+\frac{1}{u(n)\times v(n)}$$

我们发现，这几个分式中有几个会是重复的，像是这里有两个 $\frac{1}{3\times5}$ 和两个 $\frac{1}{5\times7}$ ，他们的数量就是 $u(i)-v(i)$ 个，因为重复的在 $[u(i),v(i))$ 这一段中，所以有

$$\begin{aligned}&\frac{1}{2\times3}+\frac{1}{3\times5}+\frac{1}{3\times5}+\frac{1}{5\times7}+\frac{1}{5\times7}+\cdots+\frac{1}{u(n)\times v(n)}\\=&\frac{1}{2\times3}+\frac{5-3}{3\times5}+\frac{7-5}{5\times7}+\cdots+\frac{v(n)-u(n)}{u(n)\times v(n)}\\=&\left(\frac{1}{2}-\frac{1}{3}\right)+\left(\frac{1}{3}-\frac{1}{5}\right)+\left(\frac{1}{5}-\frac{1}{7}\right)+\cdots+\left(\frac{1}{u(n)}-\frac{1}{v(n)}\right)\end{aligned}$$

到这里，你会发现式子里面可以消掉一些分式，只会剩下 $\frac{1}{2}-\frac{1}{v(n)}$ 我们可以暴力找那个质数复杂度是可以估计的。

「JLOI2014」聪明的燕姿

给出一个数 S，输出所有约数和等于 S 的数。
$S \leq 2 \times 10^9$，数据组数 ≤ 100。

「JLOI2014」聪明的燕姿

首先用约数和定理：

$$n = \prod_i p_i^{a_i}$$

$$\Rightarrow \sigma(n) = \prod_i (\sum_{j=0}^{a_i} p_i^j)$$

那么，我们可以通过从小到大来枚举质数 p_i 及其指数 a_i 来搜索。

- 若当前需要得到的 S 可以表示为一个未搜索过的质数与 1 的和，那么之前的数与这个质数的乘积是一个合法答案。
- 对于每一个使得 $(p+1) \times (p+1) < S$ 的 p，枚举可能的 a_i，递归。

因为在第一种情况会把 $a_i = 1$ 的特判掉（这种数出现并且仅出现一次），然后剩下的 $a_i \geq 2$，所以至少为 $1 + p + p^2$，最后当一直除到 1 的时候就是答案了，最开始随便用筛法筛素数就行了。

Lucas 定理

Lucas 定理

对于质数 p，有

$$\binom{n}{m} \bmod p = \binom{\lfloor n/p \rfloor}{\lfloor m/p \rfloor} \cdot \binom{n \bmod p}{m \bmod p} \bmod p$$

观察上述表达式，可知 $n \bmod p$ 和 $m \bmod p$ 一定是小于 p 的数，可以直接求解，$\binom{\lfloor n/p \rfloor}{\lfloor m/p \rfloor}$ 可以继续用 Lucas 定理求解。这也就要求 p 的范围不能够太大，一般在 10^5 左右。边界条件：当 $m = 0$ 的时候，返回 1。

时间复杂度为 $O(f(p) + g(n) \log n)$，其中 $f(n)$ 为预处理组合数的复杂度，$g(n)$ 为单次求组合数的复杂度。

拓展 Lucas

问题概述

求

$$\binom{n}{m} \bmod p$$

$1 \le m \le n \le 10^{18}, 2 \le p \le 10^6$ 不保证 p 为质数。

首先可以考虑 p 分解质因数。假设分解后 $p = \prod_i {p_i}^{k_i}$ ，我们可以对于每个 p_i 单独考虑，假设我们求出了 $\binom{n}{m} \bmod {p_i}^{k_i}$ 的值。然后我们可以考虑用前文提到的 CRT 来进行合并，那么问题就转化成如何求 $\binom{n}{m} \bmod {p_i}^{k_i}$ 。

拓展 Lucas

首先我们考虑它有多少个关于 p_i 的因子（也就是多少次方）。

我们令 $f(n)$ 为 $n!$ 中包含 p_i 的因子数量，那么 $\binom{n}{m} = \frac{n!}{m!(n-m)!}$，

所以因子数量就为 $f(n) - f(m) - f(n-m)$，那如何求 $f(n)$ 呢。

我们举出一个简单的例子来讨论。$n = 19, p_i = 3, k_i = 2$ 时:

$$\begin{aligned} n! &= 1 * 2 * 3 * \cdots * 19 \\ &= (1*2*4*5*7*8*10*11*13*14*16*17*19)*3^6*6! \\ &\equiv (1 * 2 * 4 * 5 * 7 * 8)^2 * 19 * 3^6 * 6! \\ &\equiv (1*2*4*5*7*8)^2*19*3^6*6! \pmod{3^2} \end{aligned}$$

不难发现每次把 p_i 倍数提出来的东西，就是 $\lfloor \frac{n}{p_i} \rfloor !$，那么容易得到如下递推式:

$$f(n) = f(\lfloor \frac{n}{p_i} \rfloor) + \lfloor \frac{n}{p_i} \rfloor$$

不难发现这个求单个的复杂度是 $\mathcal{O}(\log n)$ 的，十分优秀。

拓展 Lucas

然后考虑剩下与 p_i 互不相关的如何求，不难发现在 ${p_i}^{k_i}$ 意义下会存在循环节（比如前面的 $(1*2*4*5*7*8)^2$ ，可能最后多出来一个或者多个不存在循环节中的数。
不难发现循环节长度是 $< {p_i}^{k_i}$ ，因为同余的在 $> {p_i}^{k_i}$ 之后马上出现，然后把多余的 $\lfloor \frac{n}{p_i} \rfloor$ 的部分递归处理就行了。
然后就可以快速处理出与 p_i 无关的了，最后合并一下就行了。
所以算法大致流程如下：

1. 对于每个 ${p_i}^{k_i}$ 单独考虑。
   1. 用 $f(n)$ 处理出有关于 p_i 的因子个数。
   2. 然后递归处理出无关 p_i 的组合数的答案。

   把这两个乘起来合并即可。
2. 最后用 CRT 合并所有解即可。

单次询问复杂度是 $\mathcal{O}(p \log p)$ 的，可以通过预处理优化到 $\mathcal{O}(p + \log p)$ 。

拓展 Lucas

代码如下：

```
inline ll fac(ll n, ll pi, ll pk) {
    if (!n) return 1;
    ll res = 1;
    For (i, 2, pk) if (i % pi) (res *= i) %= pk;
    res = fpm(res, n / pk, pk);
    For (i, 2, n % pk) if (i % pi) (res *= i) %= pk;
    return res * fac(n / pi, pi, pk) % pk;
}

inline ll factor(ll x, ll Mod) {
    return x ? factor(x / Mod, Mod) + (x / Mod) : 0;
}

inline ll Comb(ll n, ll m, ll pi, ll pk) {
    ll k = factor(n, pi) - factor(m, pi) - factor(n - m, pi);
    if (!fpm(pi, k, pk)) return 0;
    return fac(n, pi, pk) * Inv(fac(m, pi, pk), pk) % pk *
        Inv(fac(n - m, pi, pk), pk) % pk * fpm(pi, k, pk) % pk;
}

inline ll ExLucas(ll n, ll m) {
    ll res = 0, tmp = p;
    For (i, 2, sqrt(p + .5)) if (!(tmp % i)) {
        ll pk = 1; while (!(tmp % i)) pk *= i, tmp /= i;
        (res += CRT(Comb(n, m, i, pk), pk)) %= p;
    }
    if (tmp > 1) (res += CRT(Comb(n, m, tmp, tmp), tmp)) %= p;
    return res;
}
```

类欧几里得

问题描述

求一条直线下的面积，其中 n 相当大，需要 $\mathcal{O}(\log n)$ 复杂度解决

$$f(a,b,c,n)=\sum_{i=0}^{n}\left\lfloor\frac{ai+b}{c}\right\rfloor$$

类欧几里得

问题描述

求一条直线下的面积，其中 n 相当大，需要 $\mathcal{O}(\log n)$ 复杂度解决

$$f(a,b,c,n)=\sum_{i=0}^{n}\left\lfloor\frac{ai+b}{c}\right\rfloor$$

如果说 $a\geq c$ 或者 $b\geq c$，意味着可以将 a,b 对 c 取模以简化问题：

$$\begin{aligned}
f(a,b,c,n)&=\sum_{i=0}^{n}\left\lfloor\frac{ai+b}{c}\right\rfloor\\
&=\sum_{i=0}^{n}\left\lfloor\frac{\left(\left\lfloor\frac{a}{c}\right\rfloor c+a\bmod c\right)i+\left(\left\lfloor\frac{b}{c}\right\rfloor c+b\bmod c\right)}{c}\right\rfloor\\
&=\frac{n(n+1)}{2}\left\lfloor\frac{a}{c}\right\rfloor+(n+1)\left\lfloor\frac{b}{c}\right\rfloor+\sum_{i=0}^{n}\left\lfloor\frac{(a\bmod c)i+(b\bmod c)}{c}\right\rfloor\\
&=\frac{n(n+1)}{2}\left\lfloor\frac{a}{c}\right\rfloor+(n+1)\left\lfloor\frac{b}{c}\right\rfloor+f(a\bmod c,b\bmod c,c,n)
\end{aligned}$$

类欧几里得

那么问题转化为了 $a < c, b < c$ 的情况。观察式子，你发现只有 i 这一个变量。因此要推就只能从 i 下手。在推求和式子中有一个常见的技巧，就是条件与贡献的放缩与转化。具体地说，在原式 $f(a,b,c,n) = \sum_{i=0}^{n} \left\lfloor \frac{ai+b}{c} \right\rfloor$ 中，$0 \le i \le n$ 是条件，而 $\left\lfloor \frac{ai+b}{c} \right\rfloor$ 是对总和的贡献。

要加快一个和式的计算过程，所有的方法都可以归约为 **贡献合并计算**。但你发现这个式子的贡献难以合并，怎么办？

将贡献与条件做转化得到另一个形式的和式。具体地，我们直接把原式的贡献变成条件：

$$\sum_{i=0}^{n} \left\lfloor \frac{ai+b}{c} \right\rfloor = \sum_{i=0}^{n} \sum_{j=0}^{\left\lfloor \frac{ai+b}{c} \right\rfloor - 1} 1$$

类欧几里得

多了一个变量 j，既然算 i 的贡献不方便，我们就想办法算 j 的贡献。因此想办法搞一个和 j 有关的贡献式，具体来说，在上面的和式中 n 限制 i 的上界，而 i 限制 j 的上界。为了搞 j，就先把 j 放到贡献的式子里，于是我们交换一下 i,j 的求和算子，强制用 n 限制 j 的上界。

$$= \sum_{j=0}^{\lfloor \frac{an+b}{c} \rfloor - 1} \sum_{i=0}^{n} \left[j < \left\lfloor \frac{ai+b}{c} \right\rfloor \right]$$

这样做的目的是让 j 摆脱 i 的限制，现在 i,j 都被 n 限制，而贡献式看上去是一个条件，但是我们仍把它叫作贡献式，再对贡献式做变换后就可以改变 i,j 的限制关系。于是我们做一些放缩的处理。首先把向下取整的符号拿掉

$$j < \left\lfloor \frac{ai+b}{c} \right\rfloor \iff j+1 \le \left\lfloor \frac{ai+b}{c} \right\rfloor \iff j+1 \le \frac{ai+b}{c}$$

然后可以做一些变换

$$j+1 \le \frac{ai+b}{c} \iff jc+c \le ai+b \iff jc+c-b-1 < ai$$

类欧几里得

最后一步，向下取整得到：

$$jc+c-b-1<ai \iff \left\lfloor \frac{jc+c-b-1}{a} \right\rfloor < i$$

这一步的重要意义在于，我们可以把变量 i 消掉了！具体地，令 $m=\left\lfloor \frac{an+b}{c} \right\rfloor$，那么原式化为

$$\begin{aligned} f(a,b,c,n) &= \sum_{j=0}^{m-1}\sum_{i=0}^{n}\left[i > \left\lfloor \frac{jc+c-b-1}{a} \right\rfloor \right] \\ &= \sum_{j=0}^{m-1}\left(n-\left\lfloor \frac{jc+c-b-1}{a} \right\rfloor\right) \\ &= nm-f(c,c-b-1,a,m-1) \end{aligned}$$

这是一个递归的式子。并且你发现 a,c 分子分母换了位置，又可以重复上述过程。先取模，再递归。这就是一个辗转相除的过程，这也是类欧几里德算法的得名。这样可以类似推导复杂度为 $\mathcal{O}(\log n)$。