简单数论算法

朱震霆

2019 年 1 月 28 日

安徽师范大学附属中学

总述

本文主要分为两个部分，第一部分从素数和数论函数的角度出发，描述了一些素数计数的方法，提出了一个 $\mathcal{O}\left(\frac{n^{2/3}}{\log n}\right)$ 的积性函数求和方法。第二部分则对同余方程、整数分解、离散对数进行了一些探讨。

素数计数

求 $\pi(X)$ 的精确值，即 X 以内的素数个数。

Eratosthenes 筛法

可以用于筛选 x 以内的所有素数，方法是枚举每个素数 p 并筛去 x 以内的 p 的倍数，时间复杂度 $\mathcal{O}(x \log \log x)$。

Eratosthenes 筛法

可以用于筛选 x 以内的所有素数，方法是枚举每个素数 p 并筛去 x 以内的 p 的倍数，时间复杂度 $\mathcal{O}(x \log \log x)$。

运用?

Eratosthenes 筛法

可以用于筛选 x 以内的所有素数，方法是枚举每个素数 p 并筛去 x 以内的 p 的倍数，时间复杂度 $\mathcal{O}(x \log \log x)$。

运用?

Segmented Sieve;

稍作简单优化后相比 Euler Sieve cache 更友好，在 10^8 以内的数据范围均有较大优势。

第一次改进

上述算法的第一个改进出现在 19 世纪初，Legendre's Formula 提供了仅仅计算 x 以内素数的一种思路：令部分筛函数 $\phi(x, a)$ 表示不超过 x 的最小素因子不小于 a 的数的个数，显然有 $\pi(x) = \phi(x, \sqrt{x}) + \pi(\sqrt{x}) - 1$。

第一次改进

上述算法的第一个改进出现在 19 世纪初，Legendre's Formula 提供了仅仅计算 x 以内素数的一种思路：令部分筛函数 $\phi(x, a)$ 表示不超过 x 的最小素因子不小于 a 的数的个数，显然有 $\pi(x) = \phi(x, \sqrt{x}) + \pi(\sqrt{x}) - 1$。

Legendre 提供了这样一个计算 $\phi(x, a)$ 的方法：利用容斥原理，即有 $\phi(x, a) = \sum_{p_1<p_2<\ldots<p_k\le a}(-1)^k\lfloor\frac{x}{\prod_{i=1}^{k} p_i}\rfloor$，直接利用该式计算即可得到一个 $\mathcal{O}(x)$ 的算法。

围绕部分筛函数的优化

随后的一百多年间，Meissel 和 Lehmer 主要通过用 $\phi(x, \sqrt[3]{x})$ 代替 $\phi(x, \sqrt{x})$ 的方法针对该计算方式分别进行了一些优化，但时间复杂度仍未突破 $\mathcal{O}(x^{1-\epsilon})$ 。1985 年，J. C. Lagarias, V. S. Miller 和 A. M. Odlyzko 基于 Meissel 的计算式提出了一个 $\mathcal{O}(x^{2/3} \log^{-1} x)$ 时间，$\mathcal{O}(x^{1/3} \log^2 x)$ 空间的算法。

围绕部分筛函数的优化

随后的一百多年间，Meissel 和 Lehmer 主要通过用 $\phi(x, \sqrt[3]{x})$ 代替 $\phi(x, \sqrt{x})$ 的方法针对该计算方式分别进行了一些优化，但时间复杂度仍未突破 $\mathcal{O}(x^{1-\epsilon})$ 。1985 年，J. C. Lagarias, V. S. Miller 和 A. M. Odlyzko 基于 Meissel 的计算式提出了一个 $\mathcal{O}(x^{2/3} \log^{-1} x)$ 时间，$\mathcal{O}(x^{1/3} \log^2 x)$ 空间的算法。

1996 年 Deléglise, Marc, and Joël Rivat 提出了计算素数个数的 Deléglise-Rivat 算法，这是目前最具时空效率的计算素数个数的算法，其曾被用来计算 $\pi(10^{27})$ 的值。

围绕部分筛函数的优化

随后的一百多年间，Meissel 和 Lehmer 主要通过用 $\phi(x, \sqrt[3]{x})$ 代替 $\phi(x, \sqrt{x})$ 的方法针对该计算方式分别进行了一些优化，但时间复杂度仍未突破 $\mathcal{O}(x^{1-\epsilon})$ 。1985 年，J. C. Lagarias, V. S. Miller 和 A. M. Odlyzko 基于 Meissel 的计算式提出了一个 $\mathcal{O}(x^{2/3}\log^{-1}x)$ 时间，$\mathcal{O}(x^{1/3}\log^{2}x)$ 空间的算法。

1996 年 Deléglise, Marc, and Joël Rivat 提出了计算素数个数的 Deléglise-Rivat 算法，这是目前最具时空效率的计算素数个数的算法，其曾被用来计算 $\pi(10^{27})$ 的值。

值得一提的是，1987 年 J. C. Lagarias 和 A. M. Odlyzko 提出了一个计算 $\pi(x)$ 的解析方法，时间复杂度是 $\mathcal{O}(x^{1/2+\epsilon})$ 的。

Deléglise-Rivat 算法

不妨令 $x^{1/3} \leq B < x^{1/2}$，则

$$\pi(x) = \phi(x, B) + \pi(B) - 1 - \sum_{p,q \text{ are primes}, B < p \leq q \leq \frac{x}{p}} 1$$

首先对不超过 $\frac{x}{B}$ 的部分进行筛法，那么统计 $\pi(B)$ 是 $\mathcal{O}(1)$ 的，统计后面的 p, q 对数是 $\mathcal{O}\left(\sqrt{x}\log^{-1} x\right)$ 的，那么只要考虑计算 $\phi(x, B)$。

Deléglise-Rivat 算法

考虑递归 $\phi(\frac{x}{n}, c)$ (开始时 $n = 1, c = B$) 的过程，当 $n > B$ 我们加入剪枝暂缓处理，当 $n \leq B$ 时的贡献和显然为：

$$\sum_{1 \leq n \leq B} \mu(n) \lfloor \frac{x}{n} \rfloor$$

Deléglise-Rivat 算法

考虑递归 $\phi(\frac{x}{n}, c)$ (开始时 $n = 1, c = B$) 的过程，当 $n > B$ 我们加入剪枝暂缓处理，当 $n \leq B$ 时的贡献和显然为：

$$\sum_{1 \leq n \leq B} \mu(n) \lfloor \frac{x}{n} \rfloor$$

对于其他的 n ，令 $small_n$ 表示其最小素因子，显然剩余部分为：

$$\sum_{B < n \leq B \times small_n, small_n \leq B} \mu(n) \phi(\frac{x}{n}, small_n - 1)$$

Deléglise-Rivat 算法

考虑递归 $\phi(\frac{x}{n}, c)$ (开始时 $n = 1, c = B$) 的过程，当 $n > B$ 我们加入剪枝暂缓处理，当 $n \leq B$ 时的贡献和显然为：

$$\sum_{1 \leq n \leq B} \mu(n) \lfloor \frac{x}{n} \rfloor$$

对于其他的 n ，令 $small_n$ 表示其最小素因子，显然剩余部分为：

$$\sum_{B < n \leq B \times small_n, small_n \leq B} \mu(n) \phi(\frac{x}{n}, small_n - 1)$$

令 $p = small_n, q = \frac{n}{small_n}$ ，该式即为：

$$- \sum_{\frac{B}{p} < q \leq B, p \leq B, small_q > p} \mu(q) \phi(\frac{x}{pq}, p - 1)$$

Deléglise-Rivat 算法

当 $p > \sqrt{B}$ 时，显然有 q 为质数，即统计：

$$\sum_{\sqrt{B} \le p < q \le B} \phi(\frac{x}{pq}, p-1)$$

Deléglise-Rivat 算法

当 $p > \sqrt{B}$ 时，显然有 q 为质数，即统计：

$$\sum_{\sqrt{B} \le p < q \le B} \phi(\frac{x}{pq}, p-1)$$

当 $p^2 q > x$，后半部分的 ϕ 值均为 1，可以枚举 p 并统计 q 的个数，时间复杂度 $\mathcal{O}(B \log^{-1} x)$。

Deléglise-Rivat 算法

否则显然 p 不超过 $x^{1/3}$，此时若 $p > x^{1/4}$，显然有 $\phi(\frac{x}{pq}, p-1) = 1 + \sum_{p \le r \le \frac{x}{pq}} [r \text{ is a prime}]$。

Deléglise-Rivat 算法

否则显然 p 不超过 $x^{1/3}$，此时若 $p > x^{1/4}$，显然有 $\phi(\frac{x}{pq}, p-1) = 1 + \sum_{p \le r \le \frac{x}{pq}} [r \text{ is a prime}]$。

可以发现 $\pi(\frac{x}{pq})$ 至多只有 $\mathcal{O}(\sqrt{\frac{x}{p}} \log^{-1} x)$ 种取值，要计算

$$\sum_{x^{1/4} \le p < q \le B, p \le x^{1/3}} \sum_{p \le r \le \frac{x}{pq}} [r \text{ is a prime}]$$

Deléglise-Rivat 算法

否则显然 p 不超过 $x^{1/3}$，此时若 $p > x^{1/4}$，显然有 $\phi(\frac{x}{pq}, p-1) = 1 + \sum_{p \le r \le \frac{x}{pq}} [r \text{ is a prime}]$。

可以发现 $\pi(\frac{x}{pq})$ 至多只有 $\mathcal{O}(\sqrt{\frac{x}{p}} \log^{-1} x)$ 种取值，要计算

$$\sum_{x^{1/4} \le p < q \le B, p \le x^{1/3}} \sum_{p \le r \le \frac{x}{pq}} [r \text{ is a prime}]$$

直接统计复杂度为 $\mathcal{O}(x^{2/3} \log^{-2} x)$，但为了优化空间复杂度我们可以转而枚举 r，即

$$\sum_{x^{1/4} \le p \le \min(x^{1/3}, r), p < q \le B, pqr \le x} 1 = \sum_{r \le x^{1/2}} \sum_{x^{1/4} \le p \le \min(r, x^{1/3})} \sum_{p < q \le \min(B, \frac{x}{pr})} 1$$

Deléglise-Rivat 算法

否则显然 p 不超过 $x^{1/3}$，此时若 $p > x^{1/4}$，显然有 $\phi(\frac{x}{pq}, p-1) = 1 + \sum_{p \le r \le \frac{x}{pq}} [r \text{ is a prime}]$。

可以发现 $\pi(\frac{x}{pq})$ 至多只有 $\mathcal{O}(\sqrt{\frac{x}{p}} \log^{-1} x)$ 种取值，要计算

$$\sum_{x^{1/4} \le p < q \le B, p \le x^{1/3}} \sum_{p \le r \le \frac{x}{pq}} [r \text{ is a prime}]$$

直接统计复杂度为 $\mathcal{O}(x^{2/3} \log^{-2} x)$，但为了优化空间复杂度我们可以转而枚举 r，即

$$\sum_{x^{1/4} \le p \le \min(x^{1/3}, r), p < q \le B, pqr \le x} 1 = \sum_{r \le x^{1/2}} \sum_{x^{1/4} \le p \le \min(r, x^{1/3})} \sum_{p < q \le \min(B, \frac{x}{pr})} 1$$

这样空间复杂度可以做到 $\mathcal{O}(B)$。（筛法部分可以分段区间筛）。

Deléglise-Rivat 算法

当 $p \le \sqrt{B}$，由下式

$$
\begin{aligned}
&- \sum_{\frac{B}{p} < q \le B, p \le \sqrt{B}, small_q > p} \mu(q)\phi(\frac{x}{pq}, p-1) \\
&= - \sum_{\frac{B}{p} < q \le B, p \le \sqrt{B}, small_q > p} \sum_{rpq \le x, small_r \ge p} \mu(q) \\
&= - \sum_{p \le \sqrt{B}} \sum_{r \le B^{-1}n, small_r \ge p} \sum_{\frac{B}{p} < q \le \mathbf{min}(B, \frac{x}{rp}), small_q > p} \mu(q)
\end{aligned}
$$

同样枚举 r 统计贡献，复杂度约为 $\mathcal{O}(x^{1/2}B^{1/4}\log x + B^{-1}n\log x)$ 。

Deléglise-Rivat 算法

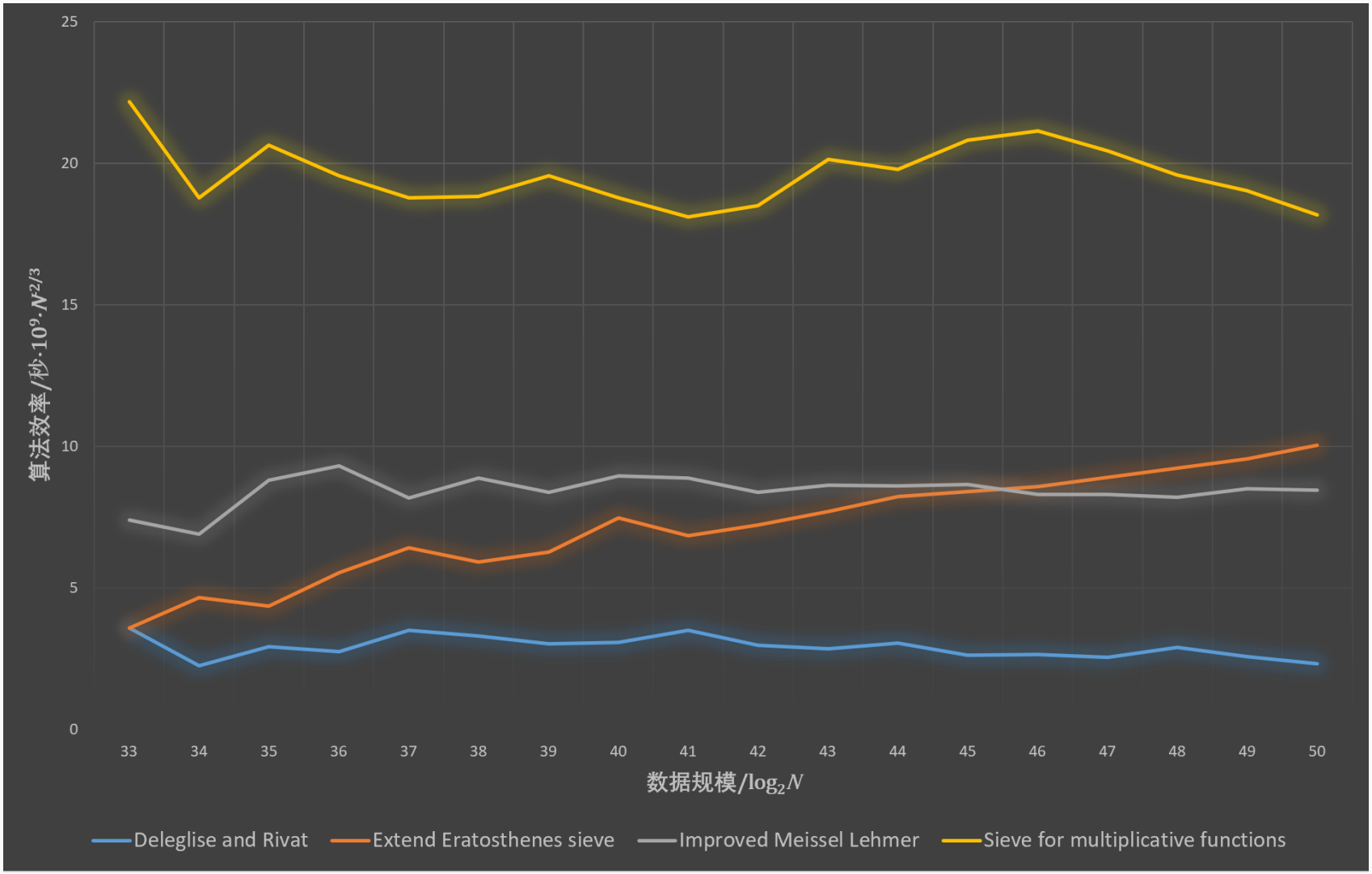
当 $p \le \sqrt{B}$，由下式

$$
\begin{aligned}
&- \sum_{\frac{B}{p} < q \le B, p \le \sqrt{B}, small_q > p} \mu(q)\phi(\frac{x}{pq}, p-1) \\
&= - \sum_{\frac{B}{p} < q \le B, p \le \sqrt{B}, small_q > p} \sum_{rpq \le x, small_r \ge p} \mu(q) \\
&= - \sum_{p \le \sqrt{B}} \sum_{r \le B^{-1}n, small_r \ge p} \sum_{\frac{B}{p} < q \le \min(B, \frac{x}{rp}), small_q > p} \mu(q)
\end{aligned}
$$

同样枚举 r 统计贡献，复杂度约为 $\mathcal{O}(x^{1/2}B^{1/4}\log x + B^{-1}n\log x)$ 。

当 $\sqrt{B} < p \le x^{1/4}$ 时情况类似，不加赘述。该算法在 $B = \mathcal{O}(x^{1/3}\log^3 x)$ 可取得最优复杂度 $\mathcal{O}(x^{2/3}\log^{-2} x)$，且空间复杂度较优。

效率对比


25
20
15
10
5
0
算法效率/秒·$10^9 \cdot N^{2/3}$
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
数据规模/$\log_2 N$
Deleglise and Rivat
Extend Eratosthenes sieve
Improved Meissel Lehmer
Sieve for multiplicative functions

积性函数

定义域为正整数，陪域为复数的函数称为数论函数。

积性函数

定义域为正整数，陪域为复数的函数称为数论函数。

在此基础上，若一个数论函数满足当 $\gcd(p,q)=1$ 有 $f(pq)=f(p)f(q)$ 时，我们称该数论函数为积性函数，不考虑 $f=0$ 的特殊情况，显然此时有 $f(1)=1$ 。

积性函数

定义域为正整数，陪域为复数的函数称为数论函数。

在此基础上，若一个数论函数满足当 $\gcd(p,q)=1$ 有 $f(pq)=f(p)f(q)$ 时，我们称该数论函数为积性函数，不考虑 $f=0$ 的特殊情况，显然此时有 $f(1)=1$ 。

更进一步的，若对于任意 p,q 均有 $f(pq)=f(p)f(q)$，则称其为完全积性函数。

常见的积性函数

定义 Euler 函数 $\varphi(x)$ 表示 $[1,x]$ 中和 x 互质的数的个数，该函数是积性函数；

定义 Möbius 函数 $\mu(x)$，当 n 有平方因子时其值为 0，否则值为 $(-1)^k$，其中 k 是 x 的质因子个数，该函数是积性函数；

定义除数函数 $\sigma_k(x)$ 表示 x 的所有正因子的 k 次幂之和，该函数是积性函数；

常见的积性函数

定义 Euler 函数 $\varphi(x)$ 表示 $[1,x]$ 中和 x 互质的数的个数，该函数是积性函数；

定义 Möbius 函数 $\mu(x)$，当 n 有平方因子时其值为 0，否则值为 $(-1)^k$，其中 k 是 x 的质因子个数，该函数是积性函数；

定义除数函数 $\sigma_k(x)$ 表示 x 的所有正因子的 k 次幂之和，该函数是积性函数；

定义单位函数 $\epsilon(x)=[x=1]$，该函数是完全积性函数；

定义恒等函数 $I(x)=id_0(x)=1$，该函数是完全积性函数；

.....

常见的积性函数

定义 Euler 函数 $\varphi(x)$ 表示 $[1,x]$ 中和 x 互质的数的个数，该函数是积性函数；

定义 Möbius 函数 $\mu(x)$，当 n 有平方因子时其值为 0，否则值为 $(-1)^k$，其中 k 是 x 的质因子个数，该函数是积性函数；

定义除数函数 $\sigma_k(x)$ 表示 x 的所有正因子的 k 次幂之和，该函数是积性函数；

定义单位函数 $\epsilon(x)=[x=1]$，该函数是完全积性函数；

定义恒等函数 $I(x)=id_0(x)=1$，该函数是完全积性函数；

……

容易发现，两个积性函数的点积仍是积性函数，两个完全积性函数的点积也仍是完全积性函数。

Dirichlet Convolution

定义作用于数论函数的运算 Dirichlet Convolution，满足 $(f*g)(n)=\sum_{d|n} f(d)g(\frac{n}{d})$。这意味着 Dirichlet Convolution 具有交换律、结合律，同时对加法有分配律。实际上全体满足 $f(1)\neq 0$ 的数论函数对 Dirichlet Convolution 组成一个阿贝尔群，可以注意到，单位函数 $\epsilon(x)$ 是该运算下的单位元。

Dirichlet Inversion

我们考虑一个数论函数 $f(x)$ 的 Dirichlet Inversion，也即寻找一个数论函数 g，使得 $f * g = \epsilon$。

Dirichlet Inversion

我们考虑一个数论函数 $f(x)$ 的 Dirichlet Inversion，也即寻找一个数论函数 g，使得 $f * g = \epsilon$。

容易发现，当 $f(1) \neq 0$ 时，数论函数 f 存在 Dirichlet Inverse，记为 f^{-1}。

Dirichlet series

定义一个数论函数 w 的 Dirichlet series 为 $f(s) = \sum_{n=1}^{\infty} \frac{w(n)}{n^s}$。

Dirichlet series

定义一个数论函数 w 的 Dirichlet series 为 $f(s)=\sum_{n=1}^{\infty}\frac{w(n)}{n^s}$。

相信很多人对这个级数形式一点都不陌生，黎曼 ζ 函数最常见的定义就是 Dirichlet series：$\zeta(s)=\sum_{n=1}^{\infty}\frac{1}{n^s}(s>1)$。

Dirichlet series

定义一个数论函数 w 的 Dirichlet series 为 $f(s)=\sum_{n=1}^{\infty}\frac{w(n)}{n^s}$。

相信很多人对这个级数形式一点都不陌生，黎曼 ζ 函数最常见的定义就是 Dirichlet series：$\zeta(s)=\sum_{n=1}^{\infty}\frac{1}{n^s}(s>1)$。

当然，如果你不喜欢 Dirichlet series 这个名字，你也可以称其为 Dirichlet Generating Function。

Dirichlet Convolution 与 Dirichlet series

我们可以写出一些常见的 Dirichlet Convolution 结果：

Dirichlet Convolution 与 Dirichlet series

我们可以写出一些常见的 Dirichlet Convolution 结果：

$\mu * 1 = \epsilon \qquad id_k * 1 = \sigma_k \qquad \varphi * 1 = id_1$

Dirichlet Convolution 与 Dirichlet series

我们可以写出一些常见的 Dirichlet Convolution 结果：

$\mu * 1 = \epsilon \qquad id_k * 1 = \sigma_k \qquad \varphi * 1 = id_1$

我们还可以写出这些函数的 Dirichlet Series：

Dirichlet Convolution 与 Dirichlet series

我们可以写出一些常见的 Dirichlet Convolution 结果：

$\mu * 1 = \epsilon \qquad id_k * 1 = \sigma_k \qquad \varphi * 1 = id_1$

我们还可以写出这些函数的 Dirichlet Series：

$\epsilon : 1$

$1 : \sum_{n=1}^{\infty} \frac{1}{n^s} = \zeta(s)$

$id_k : \sum_{n=1}^{\infty} \frac{1}{n^{s-k}} = \zeta(s-k)$

Dirichlet Convolution 与 Dirichlet series

我们可以写出一些常见的 Dirichlet Convolution 结果：

$\mu * 1 = \epsilon \qquad id_k * 1 = \sigma_k \qquad \varphi * 1 = id_1$

我们还可以写出这些函数的 Dirichlet Series：

$\epsilon : 1$

$1 : \sum_{n=1}^{\infty} \frac{1}{n^s} = \zeta(s)$

$id_k : \sum_{n=1}^{\infty} \frac{1}{n^{s-k}} = \zeta(s-k)$

$\mu : \sum_{n=1}^{\infty} \frac{\mu(n)}{n^s} = \prod_p (1 - \frac{1}{p^s}) = \frac{1}{\prod_p (\sum_{e \geq 0} p^{-es})} = \frac{1}{\zeta(s)}$

$\varphi : \sum_{n=1}^{\infty} \frac{\varphi(n)}{n^s} = \prod_p (\frac{1}{p} + \frac{p-1}{p}(1 - \frac{1}{p^{s-1}})) = \prod_p (1 - \frac{1}{p^{s-1}} + \frac{1}{p^s}) = \frac{\zeta(s-1)}{\zeta(s)}$

Dirichlet Convolution 与 Dirichlet series

我们可以写出一些常见的 Dirichlet Convolution 结果：

$\mu * 1 = \epsilon \qquad id_k * 1 = \sigma_k \qquad \varphi * 1 = id_1$

我们还可以写出这些函数的 Dirichlet Series：

$\epsilon : 1$

$1 : \sum_{n=1}^{\infty} \frac{1}{n^s} = \zeta(s)$

$id_k : \sum_{n=1}^{\infty} \frac{1}{n^{s-k}} = \zeta(s-k)$

$\mu : \sum_{n=1}^{\infty} \frac{\mu(n)}{n^s} = \prod_p (1 - \frac{1}{p^s}) = \frac{1}{\prod_p (\sum_{e \geq 0} p^{-es})} = \frac{1}{\zeta(s)}$

$\varphi : \sum_{n=1}^{\infty} \frac{\varphi(n)}{n^s} = \prod_p (\frac{1}{p} + \frac{p-1}{p}(1 - \frac{1}{p^{s-1}})) = \prod_p (1 - \frac{1}{p^{s-1}} + \frac{1}{p^s}) = \frac{\zeta(s-1)}{\zeta(s)}$

$\sigma_k : \sum_{n=1}^{\infty} \frac{\sigma_k(n)}{n^s} = \prod_p (\sum_{e \geq 0} \frac{\sum_{0 \leq r \leq e} p^{kr}}{p^{es}}) =$
$\prod_p (\sum_{e \geq 0} p^{-es})(\sum_{e \geq 0} p^{-e(s-k)}) = \zeta(s)\zeta(s-k)$

Dirichlet Convolution 与 Dirichlet series

容易发现两个数论函数进行 Dirichlet Convolution 就相当于将其 Dirichlet series 相乘：对比新的 Dirichlet series 的 n^{-s} 的系数，可以得到该结论。

Dirichlet Convolution 与 Dirichlet series

容易发现两个数论函数进行 Dirichlet Convolution 就相当于将其 Dirichlet series 相乘：对比新的 Dirichlet series 的 n^{-s} 的系数，可以得到该结论。

注意到我们可以将积性函数的 Dirichlet series 写成 $\prod_p \left(1+\sum_{e\geq 1}\frac{f(p^e)}{p^{es}}\right)$ 的形式，因此我们可以将积性函数的狄利克雷卷积看作对每个素数的一个多项式的卷积 (高维多项式卷积)，此时有个显而易见的结论就是积性函数的狄利克雷卷积仍是积性函数。

Dirichlet Convolution 与 Dirichlet series

容易发现两个数论函数进行 Dirichlet Convolution 就相当于将其 Dirichlet series 相乘：对比新的 Dirichlet series 的 n^{-s} 的系数，可以得到该结论。

注意到我们可以将积性函数的 Dirichlet series 写成 $\prod_p \left(1+\sum_{e\geq 1}\frac{f(p^e)}{p^{es}}\right)$ 的形式，因此我们可以将积性函数的狄利克雷卷积看作对每个素数的一个多项式的卷积 (高维多项式卷积)，此时有个显而易见的结论就是积性函数的狄利克雷卷积仍是积性函数。

考虑一种特殊的情形：在计算一个数论函数与某一个积性函数的卷积时，我们可以首先将积性函数分解成以上形式，并分别卷积。如果要求前 n 项，可以做到 $\mathcal{O}(n \log\log n)$。

Project Euler 484: Arithmetic Derivative

我们对一个正整数定义求导算子 $'$：

Project Euler 484: Arithmetic Derivative

我们对一个正整数定义求导算子 $'$：

1. 当 p 为素数时 $p' = 1$;

Project Euler 484: Arithmetic Derivative

我们对一个正整数定义求导算子 $'$：

1. 当 p 为素数时 $p' = 1$；
2. 对任意正整数 a, b 有 $(ab)' = a'b + ab'$。

Project Euler 484: Arithmetic Derivative

我们对一个正整数定义求导算子 $'$：

1. 当 p 为素数时 $p' = 1$；
2. 对任意正整数 a, b 有 $(ab)' = a'b + ab'$。

求 $\sum_{i=2}^{5\times10^{15}} \gcd(i, i')$。

Arithmetic Derivative

不妨假设 $x = \sum_{i=1}^{m} p_i^{e_i}$ ，我们来考虑 $\gcd(x, x')$ 的值。

Arithmetic Derivative

不妨假设 $x = \sum_{i=1}^{m} p_i^{e_i}$ ，我们来考虑 $\gcd(x, x')$ 的值。

首先考虑 $m = 1$ 的情况，根据性质 2 ，总有 $(p^e)' = p^{e-1} + p \times (p^{e-1})' = ep^{e-1}$ ，于是 $\gcd(p^e, (p^e)') = p^{e-[e \bmod p \neq 0]}$ 。

Arithmetic Derivative

不妨假设 $x = \sum_{i=1}^{m} p_i^{e_i}$ ，我们来考虑 $\gcd(x, x')$ 的值。

首先考虑 $m = 1$ 的情况，根据性质 2 ，总有 $(p^e)' = p^{e-1} + p \times (p^{e-1})' = ep^{e-1}$ ，于是 $\gcd(p^e, (p^e)') = p^{e-[e \bmod p \neq 0]}$ 。

接下来我们考虑 $m > 1$ 的情况，容易发现 $f(x)$ 中 p_i 的次数恰为 $e_i - [e_i \bmod p \neq 0]$ ，我们只要选择 $a = p_i^{e_i}$ 即可得到该结论。

Arithmetic Derivative

我们的任务就是求一个积性函数 w 的前缀和，其中 $w(p^e) = p^{e-[e \bmod p \neq 0]}$，但数据范围达到了 10^{15} 。

Arithmetic Derivative

我们的任务就是求一个积性函数 w 的前缀和，其中 $w(p^e) = p^{e-[e \bmod p \neq 0]}$，但数据范围达到了 10^{15} 。

但我们发现 $w(p) = 1$，这和恒等函数 $I(x)$ 非常相似，我们尝试计算 $Q = w/I$，其中 $/$ 是狄利克雷除法。容易发现对所有素数 p 存在 $Q(p) = 0$。

Arithmetic Derivative

我们的任务就是求一个积性函数 w 的前缀和，其中 $w(p^e) = p^{e-[e \bmod p \neq 0]}$，但数据范围达到了 10^{15} 。

但我们发现 $w(p) = 1$，这和恒等函数 $I(x)$ 非常相似，我们尝试计算 $Q = w/I$，其中 $/$ 是狄利克雷除法。容易发现对所有素数 p 存在 $Q(p) = 0$。

注意到 $\sum_{i=1}^{n} w(i) = \sum_{ij \leq n} Q(i)I(j) = \sum_{i=1}^{n} Q(i)\lfloor \frac{n}{i} \rfloor$，我们只要考虑 $Q(i)$ 不为 0 的位置。由于 Q 仍是积性函数，因此一个数 x 若满足 $Q(x) \neq 0$，则 x 满足所有素因子的次数均不小于 2 。

Powerful Number

我们称这样的素因子的次数均不小于 2 的数为 powerful number，而一个 powerful number 一定能被表示成 a^2b^3 的形式，其中 b 无平方因子，这告诉我们不超过 n 的 powerful number 个数是 $\mathcal{O}(\sqrt{n})$ 的，这样我们之前的这个问题可以在 $\mathcal{O}(\sqrt{n})$ 的时间复杂度内解决。

Powerful Number

我们称这样的素因子的次数均不小于 2 的数为 powerful number，而一个 powerful number 一定能被表示成 a^2b^3 的形式，其中 b 无平方因子，这告诉我们不超过 n 的 powerful number 个数是 $\mathcal{O}(\sqrt{n})$ 的，这样我们之前的这个问题可以在 $\mathcal{O}(\sqrt{n})$ 的时间复杂度内解决。

利用 powerful number 这一性质，我们可以优化许多关于数论函数的问题。

简单的函数

有一个积性函数 $f(p^e) = p \oplus e$，求 f 的前缀和，其中 $\oplus$ 为异或操作，现在要求函数 f 的前缀和。

$n \leq 10^{10}$

简单的函数

有一个积性函数 $f(p^e) = p \oplus e$，求 f 的前缀和，其中 $\oplus$ 为异或操作，现在要求函数 f 的前缀和。

$n \leq 10^{10}$

~~洲阁筛~~

简单的函数

有一个积性函数 $f(p^e) = p \oplus e$，求 f 的前缀和，其中 $\oplus$ 为异或操作，现在要求函数 f 的前缀和。

$n \leq 10^{10}$

~~洲阁筛~~

~~min25 筛~~

简单的函数

我们同样关注所有的 $f(p)$ 的值，容易发现对于 $p > 2$ 始终有 $f(p) = p - 1$，这和欧拉函数 φ 非常相似。同样计算 $Q = f/\varphi$，那么对所有奇素数 p 有 $Q(p) = 0$。

简单的函数

我们同样关注所有的 $f(p)$ 的值，容易发现对于 $p > 2$ 始终有 $f(p) = p - 1$，这和欧拉函数 φ 非常相似。同样计算 $Q = f/\varphi$，那么对所有奇素数 p 有 $Q(p) = 0$。

同样爆搜所有 powerful number 计算，注意此时 2 的因子无需再进行限制，这一部分的复杂度是 $\mathcal{O}(\sqrt{n})$。注意到我们现在还要处理 φ 在 $\lfloor \frac{n}{i} \rfloor$ 位置的前缀和，这个直接杜教筛就可以了，时间复杂度 $\mathcal{O}(n^{2/3})$。

简单的函数

我们同样关注所有的 $f(p)$ 的值，容易发现对于 $p > 2$ 始终有 $f(p) = p - 1$，这和欧拉函数 φ 非常相似。同样计算 $Q = f/\varphi$，那么对所有奇素数 p 有 $Q(p) = 0$。

同样爆搜所有 powerful number 计算，注意此时 2 的因子无需再进行限制，这一部分的复杂度是 $\mathcal{O}(\sqrt{n})$。注意到我们现在还要处理 φ 在 $\lfloor \frac{n}{i} \rfloor$ 位置的前缀和，这个直接杜教筛就可以了，时间复杂度 $\mathcal{O}(n^{2/3})$。

~~都 9102 年了还有人不会杜教筛?~~

杜教筛

考虑我们现在要计算函数 f 在所有 $\lfloor \frac{n}{i} \rfloor$ 上的前缀和，我们可以构造 g 满足 $g(1) = 1$ 使得 $f = h/g$，那么有：

$$\sum_{i=1}^{n} f(i) = \sum_{i=1}^{n} h(i) - \sum_{i=1}^{n} f(i) \sum_{j=2}^{\lfloor \frac{n}{i} \rfloor} g(j)$$

杜教筛

考虑我们现在要计算函数 f 在所有 $\lfloor \frac{n}{i} \rfloor$ 上的前缀和，我们可以构造 g 满足 $g(1) = 1$ 使得 $f = h/g$，那么有：

$$\sum_{i=1}^{n} f(i) = \sum_{i=1}^{n} h(i) - \sum_{i=1}^{n} f(i) \sum_{j=2}^{\lfloor \frac{n}{i} \rfloor} g(j)$$

如果 h 和 g 的前缀和都很好算的话，我们就可以预处理 $n^{2/3}$ 以内的值，其余位置根号统计，总的时间复杂度 $\mathcal{O}(n^{2/3})$。

杜教筛

考虑我们现在要计算函数 f 在所有 $\lfloor \frac{n}{i} \rfloor$ 上的前缀和，我们可以构造 g 满足 $g(1) = 1$ 使得 $f = h/g$，那么有：

$$\sum_{i=1}^{n} f(i) = \sum_{i=1}^{n} h(i) - \sum_{i=1}^{n} f(i) \sum_{j=2}^{\lfloor \frac{n}{i} \rfloor} g(j)$$

如果 h 和 g 的前缀和都很好算的话，我们就可以预处理 $n^{2/3}$ 以内的值，其余位置根号统计，总的时间复杂度 $\mathcal{O}(n^{2/3})$。

~~不过这玩意儿已经成了时代的眼泪了（其实好写还是优点的~~

局限性

以上的方法具有较大的局限性：必须满足存在一个容易计算的函数和 $f(p)$ 拟合得很好，我们的计算才能继续进行，而这样的函数往往并不容易寻找：我们通常只能在之前的常见积性函数表中寻找可能性，我们迫切需要更为通用的方法。

求

$$\sum_{lcm_{i=1}^{k} A_i \le n} \left(lcm_{i=1}^{k} A_i\right)^p f\left(gcd_{i=1}^{k} A_i\right) \pmod{10^9+7}$$

其中 $n \le 5 \times 10^{12}, 0 \le p \le 20$，$f$ 是一个不超过 20 次的多项式。

时间限制在 20 秒左右。

Chef and Working Plan

f 仍是一个数论函数——即便其是一个多项式，于是有：

$$
\begin{aligned}
&\sum_{lcm_{i=1}^{k} A_i \le n} \left(lcm_{i=1}^{k} A_i\right)^p f\left(gcd_{i=1}^{k} A_i\right) \\
=&\sum_{lcm_{i=1}^{k} A_i \le n} \left(lcm_{i=1}^{k} A_i\right)^p (f * \mu * 1)\left(gcd_{i=1}^{k} A_i\right) \\
=&\sum_{lcm_{i=1}^{k} A_i \le n} \left(lcm_{i=1}^{k} A_i\right)^p \sum_{d \mid gcd_{i=1}^{k} A_i} (f * \mu)(d) \\
=&\sum_{d=1}^{n} ((f * \mu) \circ id^p)(d) \sum_{lcm_{i=1}^{k} A_i \le \left\lfloor \frac{n}{d} \right\rfloor} \left(lcm_{i=1}^{k} A_i\right)^p
\end{aligned}
$$

不妨令 $w(x) = x^p \sum_{lcm_{i=1}^{k} A_i = x} 1$，容易发现 w 是一个完全积性函数和一个积性函数的点积，显然仍是一个积性函数，原式即为：

$$\sum_{xy \le n} ((f * \mu) \circ id^p)(x) x^p w(y)$$

$$= \sum_{x=1}^{n} (((f * \mu) \circ id^p) * w)(x)$$

Chef and Working Plan

不妨令 $w(x) = x^p \sum_{lcm_{i=1}^{k} A_i = x} 1$，容易发现 w 是一个完全积性函数和一个积性函数的点积，显然仍是一个积性函数，原式即为：

$$\sum_{xy \le n} ((f * \mu) \circ id^p)(x) x^p w(y)$$

$$= \sum_{x=1}^{n} (((f * \mu) \circ id^p) * w)(x)$$

id^p 是一个完全积性函数，这告诉我们 $(f * \mu) \circ id^p = (f \circ id^p) * (\mu \circ id^p)$。于是我们的问题是求 $(f \circ id^p) * ((\mu \circ id^p) * w)$ 的前缀和。

Chef and Working Plan

不妨令 $E = (\mu \circ id^p) * w, F(x) = \sum_{i=1}^{x} (f * id^p)(i), G(x) = \sum_{i=1}^{x} E(x)$，显然答案就是

$$\sum_{x=1}^{n} (F(x) - F(x-1)) G\left(\left\lfloor \frac{n}{x} \right\rfloor\right)$$

Chef and Working Plan

不妨令 $E = (\mu \circ id^p) * w, F(x) = \sum_{i=1}^{x} (f * id^p)(i), G(x) = \sum_{i=1}^{x} E(x)$，显然答案就是

$$\sum_{x=1}^{n} (F(x) - F(x-1)) G\left(\left\lfloor \frac{n}{x} \right\rfloor\right)$$

F 中有用的值只有所有 $\lfloor \frac{n}{x} \rfloor$ 位置的值，这一部分可以在 $\mathcal{O}((p+q)\sqrt{n})$ 内用插值解决，故略去。

Chef and Working Plan

不妨令 $E = (\mu \circ id^p) * w, F(x) = \sum_{i=1}^{x} (f * id^p)(i), G(x) = \sum_{i=1}^{x} E(x)$，显然答案就是

$$\sum_{x=1}^{n} (F(x) - F(x-1)) G\left(\left\lfloor \frac{n}{x} \right\rfloor\right)$$

F 中有用的值只有所有 $\lfloor \frac{n}{x} \rfloor$ 位置的值，这一部分可以在 $\mathcal{O}((p+q)\sqrt{n})$ 内用插值解决，故略去。

我们定义 E 的前缀和集合为 $V(E) = \{(i, \sum_{1 \le j \le i} E(j)) | 1 \le i \le \sqrt{n}\} \cup \{(\lfloor \frac{n}{i} \rfloor, \sum_{1 \le j \le \lfloor \frac{n}{i} \rfloor} E(j)) | 1 \le i \le \sqrt{n}\}$。我们实际上就是要求 $V(E)$。

Chef and Working Plan

考虑前缀和集合的性质，我们尝试将 $V(A)$ 和 $V(B)$ 合并为 $V(A * B)$，其中 $*$ 为狄利克雷卷积，记为 $V(A) \circ V(B)$。我们可以采用类似杜教筛的方法做到 $\mathcal{O}(n^{2/3+\epsilon})$。

Chef and Working Plan

考虑前缀和集合的性质，我们尝试将 $V(A)$ 和 $V(B)$ 合并为 $V(A*B)$，其中 $*$ 为狄利克雷卷积，记为 $V(A)\circ V(B)$。我们可以采用类似杜教筛的方法做到 $\mathcal{O}(n^{2/3+\epsilon})$。

显然仅仅使用上述方法并不能满足我们。不妨假设 $V(A)\circ V(B)=V(C)$，我们考虑 $V(A)$ 和 $V(B)$ 对 $V(C)$ 的贡献方式。我们用 ssum 表示 V 中不超过 $\sqrt{n}$ 的部分，lsum 表示另一部分，那么贡献方式有以下几种：

Chef and Working Plan

考虑前缀和集合的性质，我们尝试将 $V(A)$ 和 $V(B)$ 合并为 $V(A * B)$，其中 * 为狄利克雷卷积，记为 $V(A) \circ V(B)$。我们可以采用类似杜教筛的方法做到 $\mathcal{O}(n^{2/3+\epsilon})$。

显然仅仅使用上述方法并不能满足我们。不妨假设 $V(A) \circ V(B) = V(C)$，我们考虑 $V(A)$ 和 $V(B)$ 对 $V(C)$ 的贡献方式。我们用 ssum 表示 V 中不超过 $\sqrt{n}$ 的部分，lsum 表示另一部分，那么贡献方式有以下几种：

1. $\text{ssum} \circ \text{ssum} \to \text{ssum}$

Chef and Working Plan

考虑前缀和集合的性质，我们尝试将 $V(A)$ 和 $V(B)$ 合并为 $V(A * B)$，其中 * 为狄利克雷卷积，记为 $V(A) \circ V(B)$。我们可以采用类似杜教筛的方法做到 $\mathcal{O}(n^{2/3+\epsilon})$。

显然仅仅使用上述方法并不能满足我们。不妨假设 $V(A) \circ V(B) = V(C)$，我们考虑 $V(A)$ 和 $V(B)$ 对 $V(C)$ 的贡献方式。我们用 ssum 表示 V 中不超过 $\sqrt{n}$ 的部分，lsum 表示另一部分，那么贡献方式有以下几种：

1. $\text{ssum} \circ \text{ssum} \to \text{ssum}$
2. $\text{ssum} \circ \text{lsum} \to \text{lsum}$

Chef and Working Plan

考虑前缀和集合的性质，我们尝试将 $V(A)$ 和 $V(B)$ 合并为 $V(A * B)$，其中 * 为狄利克雷卷积，记为 $V(A) \circ V(B)$。我们可以采用类似杜教筛的方法做到 $\mathcal{O}(n^{2/3+\epsilon})$。

显然仅仅使用上述方法并不能满足我们。不妨假设 $V(A) \circ V(B) = V(C)$，我们考虑 $V(A)$ 和 $V(B)$ 对 $V(C)$ 的贡献方式。我们用 ssum 表示 V 中不超过 $\sqrt{n}$ 的部分，lsum 表示另一部分，那么贡献方式有以下几种：

1. $\text{ssum} \circ \text{ssum} \to \text{ssum}$
2. $\text{ssum} \circ \text{lsum} \to \text{lsum}$
3. $\text{ssum} \circ \text{ssum} \to \text{lsum}$

Chef and Working Plan

考虑前缀和集合的性质，我们尝试将 $V(A)$ 和 $V(B)$ 合并为 $V(A * B)$，其中 * 为狄利克雷卷积，记为 $V(A) \circ V(B)$。我们可以采用类似杜教筛的方法做到 $\mathcal{O}(n^{2/3+\epsilon})$。

显然仅仅使用上述方法并不能满足我们。不妨假设 $V(A) \circ V(B) = V(C)$，我们考虑 $V(A)$ 和 $V(B)$ 对 $V(C)$ 的贡献方式。我们用 ssum 表示 V 中不超过 $\sqrt{n}$ 的部分，lsum 表示另一部分，那么贡献方式有以下几种：

1. $\text{ssum} \circ \text{ssum} \to \text{ssum}$
2. $\text{ssum} \circ \text{lsum} \to \text{lsum}$
3. $\text{ssum} \circ \text{ssum} \to \text{lsum}$

对于前两种情况，直接暴力处理是 $\mathcal{O}(\sqrt{n} \log n)$ 的，实际上牵制我们计算的只有最后一种情况。

Chef and Working Plan

对于 $\mathrm{ssum} \circ \mathrm{ssum} \to \mathrm{lsum}$ 的部分，由于贡献可以表示成 $f(m) = \sum_{x=\frac{m}{\sqrt{n}}}^{\sqrt{n}} a(x) b\left(\lfloor \frac{m}{x} \rfloor\right)$ 的形式，通常采用的办法是 $\mathcal{O}(\sqrt{m})$ 枚举。

Chef and Working Plan

对于 $\mathrm{ssum} \circ \mathrm{ssum} \to \mathrm{lsum}$ 的部分，由于贡献可以表示成 $f(m) = \sum_{x=\frac{m}{\sqrt{n}}}^{\sqrt{n}} a(x) b\left(\lfloor \frac{m}{x} \rfloor\right)$ 的形式，通常采用的办法是 $\mathcal{O}(\sqrt{m})$ 枚举。

但我们发现，有时 a 和 b 中的元素个数较为稀疏，这时我们跳过 a 和 b 中，值为 0 的部分。可以证明，若 a 和 b 中元素的出现概率近似为 $\frac{1}{p}$ 和 $\frac{1}{q}$，那么这样枚举的复杂度大约为 $\mathcal{O}\left(\sqrt{\frac{m}{pq}}\right)$。

回到原来的问题，由 $E(x)$ 是一个积性函数，我们有：

$$V(E) = \bigcirc_{p \text{ is a prime}} V(E_p)$$

Chef and Working Plan

回到原来的问题，由 $E(x)$ 是一个积性函数，我们有：

$$V(E) = \bigcirc_{p \text{ is a prime}} V(E_p)$$

其中

$$E_p(x) = [x \text{ is a power of } p]E(x)$$

Chef and Working Plan

我们将原式分成三部分处理：

$$
\begin{aligned}
V(E) = &\bigcirc_{1 \le p \le n^{\alpha} \text{ is a prime}} V(E_p) \circ \\
&\bigcirc_{n^{\alpha} < p \le \sqrt{n} \text{ is a prime}} V(E_p) \circ \\
&\bigcirc_{p > \sqrt{n} \text{ is a prime}} V(E_p)
\end{aligned}
$$

Chef and Working Plan

我们将原式分成三部分处理：

$$\begin{aligned} V(E) = &\bigcirc_{1\le p\le n^{\alpha} \text{ is a prime}} V(E_p) \circ \\ &\bigcirc_{n^{\alpha}<p\le\sqrt{n} \text{ is a prime}} V(E_p) \circ \\ &\bigcirc_{p>\sqrt{n} \text{ is a prime}} V(E_p) \end{aligned}$$

对于第一部分，合并上单个 $V(E_p)$ 可以做到 $\mathcal{O}(\sqrt{n}\log_p(n))$，因此 α 足够小时我们不必关心复杂度；

Chef and Working Plan

我们将原式分成三部分处理：

$$V(E) = \bigcirc_{1\le p\le n^{\alpha} \text{ is a prime}} V(E_p) \circ \bigcirc_{n^{\alpha}<p\le\sqrt{n} \text{ is a prime}} V(E_p) \circ \bigcirc_{p>\sqrt{n} \text{ is a prime}} V(E_p)$$

对于第一部分，合并上单个 $V(E_p)$ 可以做到 $\mathcal{O}(\sqrt{n}\log_p(n))$，因此 α 足够小时我们不必关心复杂度；

最后一部分没有 ssum，因此这一部分容易 $\mathcal{O}(\sqrt{n}\log n)$ 合并，接下来我们讨论这一部分的 $E(p)$ 是一个项数不多次数不高的多项式的情形。也就是说，我们要求素数的 s 次幂前缀和。

Chef and Working Plan

我们将原式分成三部分处理：

$$V(E) = \bigcirc_{1\le p\le n^{\alpha} \text{ is a prime}} V(E_p) \circ \bigcirc_{n^{\alpha}<p\le\sqrt{n} \text{ is a prime}} V(E_p) \circ \bigcirc_{p>\sqrt{n} \text{ is a prime}} V(E_p)$$

对于第一部分，合并上单个 $V(E_p)$ 可以做到 $\mathcal{O}(\sqrt{n}\log_p(n))$，因此 α 足够小时我们不必关心复杂度；

最后一部分没有 ssum，因此这一部分容易 $\mathcal{O}(\sqrt{n}\log n)$ 合并，接下来我们讨论这一部分的 $E(p)$ 是一个项数不多次数不高的多项式的情形。也就是说，我们要求素数的 s 次幂前缀和。

在这种情况下，我们尝试将问题转化为第一和第二部分的问题。

Chef and Working Plan

我们构造函数 $E'(p^e) = [p \leq \sqrt{n}]p^{es}$，这个函数是没有 $p > \sqrt{n}$ 的部分的，我们可以对这个函数先求前缀和，这样我们求得了不包含 $> \sqrt{n}$ 的素数的所有数前缀和。

Chef and Working Plan

我们构造函数 $E'(p^e) = [p \le \sqrt{n}]p^{es}$，这个函数是没有 $p > \sqrt{n}$ 的部分的，我们可以对这个函数先求前缀和，这样我们求得了不包含 $> \sqrt{n}$ 的素数的所有数前缀和。

接下来考虑容斥，我们可以求出所有不是超过 $\sqrt{n}$ 的质数的数的 s 次幂前缀和：只要枚举不超过 $\sqrt{n}$ 的部分容斥即可。容斥的复杂度为 $\mathcal{O}(\sqrt{n}\log n)$。

于是我们只要关心如何求 $V(T) = \bigcirc_{n^\alpha < p \le \sqrt{n} \text{ is a prime}} V(E_p)$ 即可。

Chef and Working Plan

于是我们只要关心如何求 $V(T) = \bigcirc_{n^\alpha < p \le \sqrt{n} \text{ is a prime}} V(E_p)$ 即可。

我们可以采用类似洲阁筛的方法，从后往前逐一合并质数，这样的复杂度是 $\mathcal{O}(\frac{n^{3/4}}{\log n})$ 的，但我们并不满足于这一点，我们希望探索更为优秀的方法。

Chef and Working Plan

于是我们只要关心如何求 $V(T) = \bigcirc_{n^\alpha < p \le \sqrt{n} \text{ is a prime}} V(E_p)$ 即可。

我们可以采用类似洲阁筛的方法，从后往前逐一合并质数，这样的复杂度是 $\mathcal{O}(\frac{n^{3/4}}{\log n})$ 的，但我们并不满足于这一点，我们希望探索更为优秀的方法。

容易发现，这一做法的瓶颈在于需要同时考虑不同素数间的大小关系，而我们期待的是可以不考虑因子的顺序，直接枚举任意一个因子来转移。为了做到这一点，我们首先观察在什么情况下可以对所有素因子一起转移。

Chef and Working Plan

令 $Q(x) = [x \text{ is a power of any } n^{\alpha} < p \leq \sqrt{n}]E(x)$，我们期待的是

$$V(T) = \sum_{i \geq 0} V(Q)^i$$

Chef and Working Plan

令 $Q(x) = [x \text{ is a power of any } n^{\alpha} < p \leq \sqrt{n}]E(x)$，我们期待的是

$$V(T) = \sum_{i \geq 0} V(Q)^i$$

但我们发现这里我们是乱序枚举所有素因子的，因此该式不可能成立。对于一个恰有 k 个不同素因子的数，它会被统计 $k!$ 次，我们需要在最终的贡献中除去这一部分的贡献，即：

$$V(T) = \sum_{i \geq 0} \frac{V(Q)^i}{i!}$$

Chef and Working Plan

令 $Q(x) = [x \text{ is a power of any } n^{\alpha} < p \leq \sqrt{n}]E(x)$，我们期待的是

$$V(T) = \sum_{i \geq 0} V(Q)^i$$

但我们发现这里我们是乱序枚举所有素因子的，因此该式不可能成立。对于一个恰有 k 个不同素因子的数，它会被统计 $k!$ 次，我们需要在最终的贡献中除去这一部分的贡献，即：

$$V(T) = \sum_{i \geq 0} \frac{V(Q)^i}{i!}$$

在这样的条件下，我们讨论 $E(p^e)$ 与 $E(p)$ 的关系。

Chef and Working Plan

对于 $n = \prod_{i=1}^{m} p_i^{e_i}$，$n$ 的函数值会被统计 $\frac{(\sum e_i)!}{\prod e_i!}$ 次，统计到的函数值为 $\prod E(p_i)^{e_i}$，因此只要 $E(p^e) = \frac{E(p)^e}{e!}$，就存在

$$V(T) = \sum_{i \geq 0} \frac{V(Q)^i}{i!}$$

Chef and Working Plan

对于 $n = \prod_{i=1}^{m} p_i^{e_i}$，$n$ 的函数值会被统计 $\frac{(\sum e_i)!}{\prod e_i!}$ 次，统计到的函数值为 $\prod E(p_i)^{e_i}$，因此只要 $E(p^e) = \frac{E(p)^e}{e!}$，就存在

$$V(T) = \sum_{i \geq 0} \frac{V(Q)^i}{i!}$$

而该式存在意味着，我们得到了一个非常优秀的积性函数求和方法：注意到 n 以内最小素因子不小于 n^{α} 的数的个数是 $\frac{n}{\log n}$ 的，我们可以认为函数有值的概率约为 $\frac{1}{\log n}$。我们仍然采用类似杜教筛的方法，对不超过 $n^{2/3}$ 的部分预处理，对超过的部分暴力处理，而质因子个数不会超过 $\frac{1}{\alpha}$ 个，因此我们这一步计算的复杂度是 $\mathcal{O}\left(\frac{n^{2/3}}{\log n}\right)$ 的。

Chef and Working Plan

但实际上，$E(p^e) = \frac{E(p)^e}{e!}$ 这一条件几乎不可能被满足，但我们可以类比之前利用 powerful number 求值的方法：只要构造函数 D 满足所有 $D(p) = E(p)$ 即可。

Chef and Working Plan

但实际上，$E(p^e) = \frac{E(p)^e}{e!}$ 这一条件几乎不可能被满足，但我们可以类比之前利用 powerful number 求值的方法：只要构造函数 D 满足所有 $D(p) = E(p)$ 即可。

令函数 D 满足 $D(p^e) = \frac{E(p)^e}{e!}$，我们计算 $H = E/D$，那么求 E 的前缀和就可以转化为求 $H * D$ 的前缀和，其中 H 只有 powerful number 处值非零。

Chef and Working Plan

但实际上，$E(p^e) = \frac{E(p)^e}{e!}$ 这一条件几乎不可能被满足，但我们可以类比之前利用 powerful number 求值的方法：只要构造函数 D 满足所有 $D(p) = E(p)$ 即可。

令函数 D 满足 $D(p^e) = \frac{E(p)^e}{e!}$，我们计算 $H = E/D$，那么求 E 的前缀和就可以转化为求 $H * D$ 的前缀和，其中 H 只有 powerful number 处值非零。

考虑对于每个 $\lfloor \frac{n}{d} \rfloor$ 暴力卷积，单次计算的复杂度是 $\mathcal{O}(n^{1/3}d^{-1/3})$ 的，总的复杂度是 $\mathcal{O}(n^{2/3})$。

Chef and Working Plan

但实际上，$E(p^e) = \frac{E(p)^e}{e!}$ 这一条件几乎不可能被满足，但我们可以类比之前利用 powerful number 求值的方法：只要构造函数 D 满足所有 $D(p) = E(p)$ 即可。

令函数 D 满足 $D(p^e) = \frac{E(p)^e}{e!}$，我们计算 $H = E/D$，那么求 E 的前缀和就可以转化为求 $H * D$ 的前缀和，其中 H 只有 powerful number 处值非零。

考虑对于每个 $\lfloor \frac{n}{d} \rfloor$ 暴力卷积，单次计算的复杂度是 $\mathcal{O}(n^{1/3}d^{-1/3})$ 的，总的复杂度是 $\mathcal{O}(n^{2/3})$。

暴力卷积不超过 $n^{0.6}$ 的部分即可，这样这一部分并不会影响总体的复杂度，这样我们在 $\mathcal{O}\left(\frac{n^{2/3}}{\log n}\right)$ 的复杂度内求出了

$$V(E) = \bigcirc_{1 \le p \le n^{\alpha} \text{ is a prime}} V(E_p) \circ \bigcirc_{n^{\alpha} < p \le \sqrt{n} \text{ is a prime}} V(E_p)$$

Chef and Working Plan

但实际上，$E(p^e) = \frac{E(p)^e}{e!}$ 这一条件几乎不可能被满足，但我们可以类比之前利用 powerful number 求值的方法：只要构造函数 D 满足所有 $D(p) = E(p)$ 即可。

令函数 D 满足 $D(p^e) = \frac{E(p)^e}{e!}$，我们计算 $H = E/D$，那么求 E 的前缀和就可以转化为求 $H * D$ 的前缀和，其中 H 只有 powerful number 处值非零。

考虑对于每个 $\lfloor \frac{n}{d} \rfloor$ 暴力卷积，单次计算的复杂度是 $\mathcal{O}(n^{1/3}d^{-1/3})$ 的，总的复杂度是 $\mathcal{O}(n^{2/3})$。

暴力卷积不超过 $n^{0.6}$ 的部分即可，这样这一部分并不会影响总体的复杂度，这样我们在 $\mathcal{O}\left(\frac{n^{2/3}}{\log n}\right)$ 的复杂度内求出了

$$V(E) = \bigcirc_{1 \le p \le n^{\alpha} \text{ is a prime}} V(E_p) \circ \bigcirc_{n^{\alpha} < p \le \sqrt{n} \text{ is a prime}} V(E_p)$$

于是我们的问题得到解决。

其他的方法

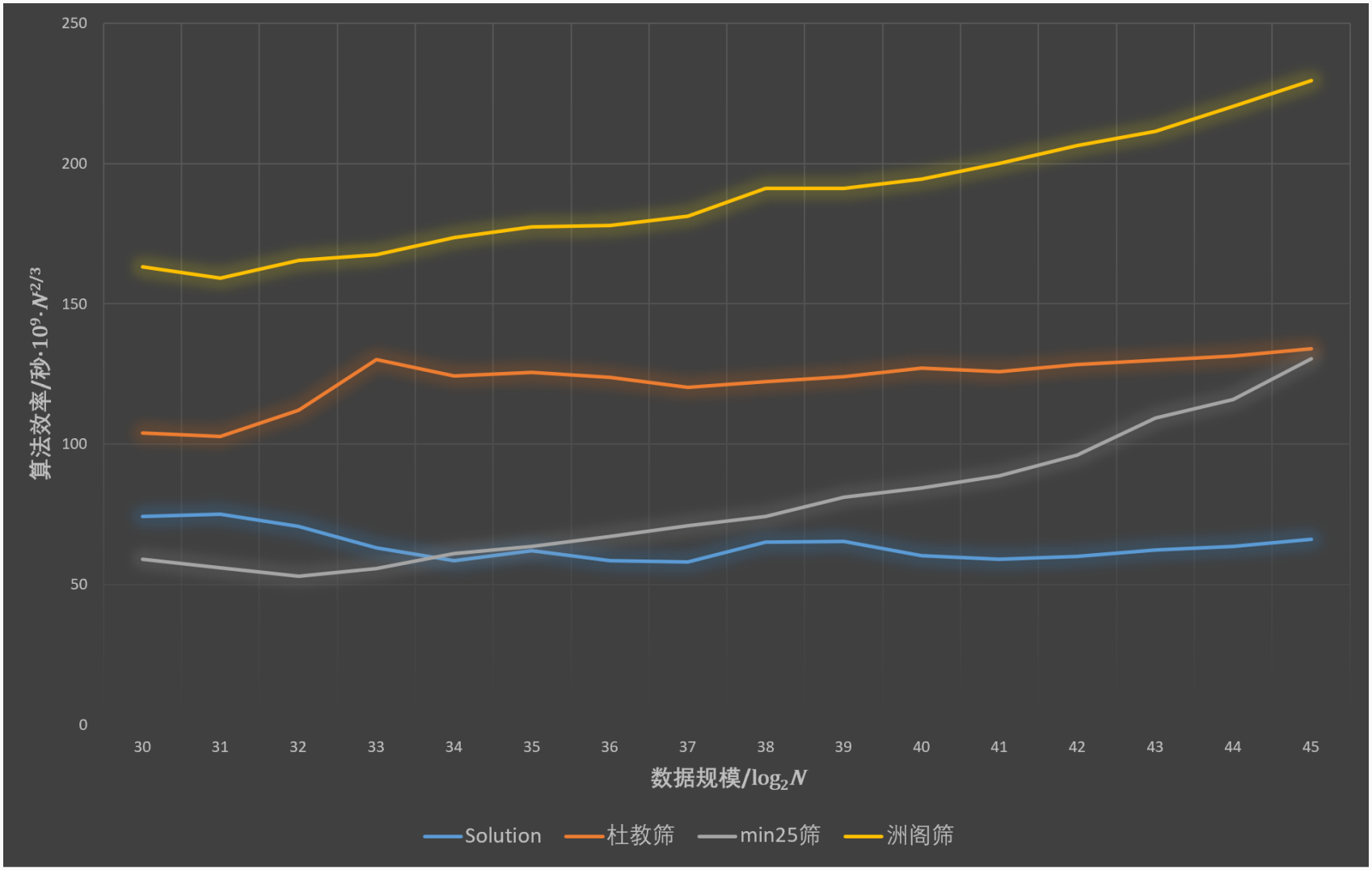
在 min-25 的博客上，提到了一种复杂度为 $\mathcal{O}(n^{2/3})$ 的方法：大致是在 min25 筛的基础上，增加上对较小的值进行预处理，使用树状数组维护以去掉最小质因子的限制，是可以做到 $\mathcal{O}(n^{2/3})$ 的。

其他的方法

在 min-25 的博客上，提到了一种复杂度为 $\mathcal{O}(n^{2/3})$ 的方法：大致是在 min25 筛的基础上，增加上对较小的值进行预处理，使用树状数组维护以去掉最小质因子的限制，是可以做到 $\mathcal{O}(n^{2/3})$ 的。

考后 @peehs_moorhsum 提到了另一种方法：我们尝试在 min25 筛的递归过程中，加入当 $p^3 > n$ 就停止递归的剪枝。那么我们需要得到含有恰好两个素因子的最小素因子不超过 p 的函数和。如果 $f(pq)$ 总是 pq 和 p 相关的项数较少的多项式，我们是可以大力递推出所有这些值的：我们可以将所有数都看成 pq 的形式统计贡献。

效率对比


250
200
150
100
50
0
算法效率/秒·10^9·$N^{2/3}$
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
数据规模/$\log_2 N$
Solution
杜教筛
min25筛
洲阁筛

同余方程

求 $ax \equiv b \pmod{p}$ 的任意一组整数解。

同余方程

求 $ax \equiv b \pmod{p}$ 的任意一组整数解。

这种普及组题大家肯定都会

同余方程

求 $ax \equiv b \pmod{p}$ 的任意一组整数解。

这种普及组题大家肯定都会

当 p 为素数时，由于 $x \equiv a^{\varphi(p)-1}b \pmod{p}$，使用快速幂求解即可。

同余方程

求 $ax \equiv b \pmod{p}$ 的任意一组整数解。

这种普及组题大家肯定都会

当 p 为素数时，由于 $x \equiv a^{\varphi(p)-1}b \pmod{p}$，使用快速幂求解即可。

当 p 不一定为素数时，可以用扩展欧几里得算法求解。

同余方程组

给定同余方程组 $x \equiv r_i \pmod{m_i}$，求一个可行的 x。

同余方程组

给定同余方程组 $x \equiv r_i \pmod{m_i}$，求一个可行的 x。

考虑只有两个方程的情况，不妨假设 $d = \gcd(m_1, m_2)$，那么若 $r_1 \not\equiv r_2 \pmod{d}$，显然方程组无解；

同余方程组

给定同余方程组 $x \equiv r_i \pmod{m_i}$，求一个可行的 x。

考虑只有两个方程的情况，不妨假设 $d = \gcd(m_1, m_2)$，那么若 $r_1 \not\equiv r_2 \pmod{d}$，显然方程组无解；

只要求得一个可能的 $y = \lfloor \frac{x}{d} \rfloor$，同样满足两个同余方程组的限制，且模数互质；

同余方程组

给定同余方程组 $x \equiv r_i \pmod{m_i}$，求一个可行的 x。

考虑只有两个方程的情况，不妨假设 $d = \gcd(m_1, m_2)$，那么若 $r_1 \not\equiv r_2 \pmod{d}$，显然方程组无解；

只要求得一个可能的 $y = \lfloor \frac{x}{d} \rfloor$，同样满足两个同余方程组的限制，且模数互质；

当模数互质时，利用插值的想法处理即可。

阶和原根

对于两个互质的整数 a, p 来说，a 模 p 的阶就是最小的满足 $a^k \equiv 1 \pmod{p}$ 的 k，记为 $\delta_p(a) = k$。

显然有 $\delta_p(a) | \varphi(p)$ 。

阶和原根

对于两个互质的整数 a, p 来说，a 模 p 的阶就是最小的满足 $a^k \equiv 1 \pmod{p}$ 的 k，记为 $\delta_p(a) = k$。

显然有 $\delta_p(a) | \varphi(p)$ 。

当 $\delta_p(a) = \varphi(p)$ 时称 a 为 p 的一个原根。可以证明只有 $1, 2, 4, p^k, 2p^k$ 存在原根，其中 p 为奇素数。

容易发现，如果 p 存在原根，那么一定存在恰好 $\varphi(\varphi(p))$ 个，这实际上是非常多的。因此我们可以随机原根并枚举 $\varphi(p)$ 的每个素因子判断即可。

指标

设 g 为 mod$p$ 的一个原根，那么 $g^i(0 \leq i < \varphi(p))$ 构成一个 mod$p$ 的简化剩余系。设 $x \equiv g^t \pmod{p}$，那么 $ind(x) = t$。

指标

设 g 为 mod$p$ 的一个原根，那么 $g^i(0 \le i < \varphi(p))$ 构成一个 mod$p$ 的简化剩余系。设 $x \equiv g^t \pmod{p}$，那么 $ind(x) = t$。

可以利用 BSGS 来求指标，即预处理 $i < \lfloor\sqrt{\varphi(p)}\rfloor$ 的 g^i，将 g^t 拆分为 $g^{a\lfloor\sqrt{\varphi(p)}\rfloor+r}$，枚举 a 即可得到 t。

“同余方程”?

求 $a^x \equiv b \pmod{p}$ 的任意一组解 x，其中 p 为素数。

$p \leq 10^{12}$

“同余方程”?

求 $a^x \equiv b \pmod{p}$ 的任意一组解 x，其中 p 为素数。

$p \leq 10^{12}$

这是一个经典问题，我们称 $x = \log_a b$ 为 b 关于 a 在模 p 意义下的离散对数。

“同余方程”?

求 $a^x \equiv b \pmod{p}$ 的任意一组解 x，其中 p 为素数。

$p \leq 10^{12}$

这是一个经典问题，我们称 $x = \log_a b$ 为 b 关于 a 在模 p 意义下的离散对数。

直接 BSGS 求解的时间复杂度为 $\mathcal{O}(\sqrt{p})$。

二次剩余

对正整数 a，判定是否存在 $x^2 \equiv a \pmod{p}$，其中 p 为奇素数。

二次剩余

对正整数 a，判定是否存在 $x^2 \equiv a \pmod{p}$，其中 p 为奇素数。

容易知道，在模奇素数 p 意义下的一个正整数 a 是二次剩余当且仅当 $a^{\frac{p-1}{2}} \equiv 1 \pmod{p}$。

二次剩余

对正整数 a，求 $x^2 \equiv a \pmod{p}$ 的正整数解 x，其中 p 为奇素数。

二次剩余

对正整数 a，求 $x^2 \equiv a \pmod{p}$ 的正整数解 x，其中 p 为奇素数。

根据我们之前原根的知识，我们知道该方程对于 $[1, p-1]$ 中恰一半的 a 有解，且解的个数恰好为 2 个。

二次剩余

对正整数 a，求 $x^2 \equiv a \pmod{p}$ 的正整数解 x，其中 p 为奇素数。

根据我们之前原根的知识，我们知道该方程对于 $[1, p-1]$ 中恰一半的 a 有解，且解的个数恰好为 2 个。

这个问题的 $\mathcal{O}(\sqrt{p})$ 方法非常容易：只要求出任意一个原根，并求出 a 的指标，可以方便地得到这两组解。

Cipolla 算法

模奇素数下的二次剩余问题已经被很好地解决了：Cipolla 算法就是众多解决方案中的一种。

Cipolla 算法

模奇素数下的二次剩余问题已经被很好地解决了：Cipolla 算法就是众多解决方案中的一种。

算法描述大致如下：不妨假设 $w = b^2 - a$ 是模 p 意义下的二次非剩余，则 $x \equiv (b + \sqrt{w})^{\frac{p+1}{2}} \pmod{p}$ 为方程的解。

Cipolla 算法

模奇素数下的二次剩余问题已经被很好地解决了：Cipolla 算法就是众多解决方案中的一种。

算法描述大致如下：不妨假设 $w = b^2 - a$ 是模 p 意义下的二次非剩余，则 $x \equiv (b + \sqrt{w})^{\frac{p+1}{2}} \pmod{p}$ 为方程的解。

正确性?

Cipolla 算法

模奇素数下的二次剩余问题已经被很好地解决了：Cipolla 算法就是众多解决方案中的一种。

算法描述大致如下：不妨假设 $w = b^2 - a$ 是模 p 意义下的二次非剩余，则 $x \equiv (b + \sqrt{w})^{\frac{p+1}{2}} \pmod{p}$ 为方程的解。

正确性?

w 是模 p 意义下的二次非剩余，因此 $w^{\frac{p-1}{2}} \equiv -1 \pmod{p}$。结合二项式定理我们有：

Cipolla 算法

模奇素数下的二次剩余问题已经被很好地解决了：Cipolla 算法就是众多解决方案中的一种。

算法描述大致如下：不妨假设 $w = b^2 - a$ 是模 p 意义下的二次非剩余，则 $x \equiv (b + \sqrt{w})^{\frac{p+1}{2}} \pmod{p}$ 为方程的解。

正确性?

w 是模 p 意义下的二次非剩余，因此 $w^{\frac{p-1}{2}} \equiv -1 \pmod{p}$。结合二项式定理我们有：

$$(b + \sqrt{w})^p \equiv b^p + \sqrt{w}^p \equiv b - \sqrt{w} \pmod{p}$$

于是上式确实是 $x^2 \equiv a$ 的一组解，由拉格朗日定理，我们求得的 x 必然是整数，问题得到完美解决。不考虑整数乘法的情况下，时间复杂度为 $\mathcal{O}(\log p)$。

Tonelli-Shanks 算法

我们再介绍另一种求解二次剩余的方法，相比 Cipolla 算法，该算法更为自然，且可扩展性更强。

Tonelli-Shanks 算法

我们再介绍另一种求解二次剩余的方法，相比 Cipolla 算法，该算法更为自然，且可扩展性更强。

我们求解 $x^2 \equiv n \pmod{q}$，不妨令 $q = 2^m Q + 1$，其中 Q 是奇数，那么取 $y \equiv n^{\frac{Q+1}{2}} \pmod{q}$ 即有

$$y^2 \equiv n \times n^Q \pmod{q}$$

Tonelli-Shanks 算法

我们再介绍另一种求解二次剩余的方法，相比 Cipolla 算法，该算法更为自然，且可扩展性更强。

我们求解 $x^2 \equiv n \pmod{q}$，不妨令 $q = 2^m Q + 1$，其中 Q 是奇数，那么取 $y \equiv n^{\frac{Q+1}{2}} \pmod{q}$ 即有

$$y^2 \equiv n \times n^Q \pmod{q}$$

不妨令 $t \equiv n^Q \pmod{q}$，由于 n 存在二次剩余，$t^{2^{m-1}} \equiv 1 \pmod{q}$。

Tonelli-Shanks 算法

不妨假设已经有 $y^2 = nt$ 和 $t^{2^r} \equiv 1 \pmod{q}$，我们可以推出 $y_0^2 \equiv nt_0 \pmod{q}$ 和 $t_0^{2^{r-1}} \equiv 1 \pmod{q}$：

Tonelli-Shanks 算法

不妨假设已经有 $y^2 = nt$ 和 $t^{2^r} \equiv 1 \pmod{q}$，我们可以推出 $y_0^2 \equiv nt_0 \pmod{q}$ 和 $t_0^{2^{r-1}} \equiv 1 \pmod{q}$：

- 若 $t^{2^{r-1}} \equiv 1 \pmod{q}$，直接可得一组 y_0 和 t_0

Tonelli-Shanks 算法

不妨假设已经有 $y^2 = nt$ 和 $t^{2^r} \equiv 1 \pmod{q}$，我们可以推出 $y_0^2 \equiv nt_0 \pmod{q}$ 和 $t_0^{2^{r-1}} \equiv 1 \pmod{q}$：

- 若 $t^{2^{r-1}} \equiv 1 \pmod{q}$，直接可得一组 y_0 和 t_0
- 若 $t^{2^{r-1}} \equiv -1 \pmod{q}$，我们找到一个 z 满足 $z^{2^{m-1}Q} \equiv -1 \pmod{q}$，我们需要找到一个 w 满足 $w^{2^r} \equiv -1 \pmod{q}$ 从而 $(tw^2)^{2^{r-1}} \equiv 1 \pmod{q}$，容易发现 $w = z^{Q2^{m-r-1}}$ 符合要求。

Tonelli-Shanks 算法

不妨假设已经有 $y^2 = nt$ 和 $t^{2^r} \equiv 1 \pmod{q}$，我们可以推出 $y_0^2 \equiv nt_0 \pmod{q}$ 和 $t_0^{2^{r-1}} \equiv 1 \pmod{q}$：

- 若 $t^{2^{r-1}} \equiv 1 \pmod{q}$，直接可得一组 y_0 和 t_0
- 若 $t^{2^{r-1}} \equiv -1 \pmod{q}$，我们找到一个 z 满足 $z^{2^{m-1}Q} \equiv -1 \pmod{q}$，我们需要找到一个 w 满足 $w^{2^r} \equiv -1 \pmod{q}$ 从而 $(tw^2)^{2^{r-1}} \equiv 1 \pmod{q}$，容易发现 $w = z^{Q2^{m-r-1}}$ 符合要求。

直到 $r = 0$，我们得到了 $x^2 \equiv n \pmod{q}$ 的一组解，该算法的时间复杂度是 $\mathcal{O}(\log^2 p)$ 的。

模 q^e 下的二次剩余

即求 $x^2 \equiv a \pmod{q^e}$ 的整数解，首先我们将问题转化为 $\gcd(a, q) = 1$ 的情形。

模 q^e 下的二次剩余

即求 $x^2 \equiv a \pmod{q^e}$ 的整数解，首先我们将问题转化为 $\gcd(a, q) = 1$ 的情形。

类比 Tonelli-Shanks 算法，不妨令 $\varphi(q) = 2^m Q$，其中 Q 是奇数，那么取 $y \equiv n^{\frac{Q+1}{2}} \pmod{q^e}$ 即有

$$y^2 \equiv n \times n^Q \pmod{q^e}$$

模 q^e 下的二次剩余

即求 $x^2 \equiv a \pmod{q^e}$ 的整数解，首先我们将问题转化为 $\gcd(a, q) = 1$ 的情形。

类比 Tonelli-Shanks 算法，不妨令 $\varphi(q) = 2^m Q$，其中 Q 是奇数，那么取 $y \equiv n^{\frac{Q+1}{2}} \pmod{q^e}$ 即有

$$y^2 \equiv n \times n^Q \pmod{q^e}$$

不妨令 $t \equiv n^Q \pmod{q^e}$，由于 a 存在二次剩余，$t^{2^{m-1}} \equiv 1 \pmod{q^e}$。

模 q^e 下的二次剩余

不妨假设已经有 $y^2 = nt$ 和 $t^{2^r} \equiv 1 \pmod{q^e}$，我们可以推出 $y_0^2 \equiv nt_0 \pmod{q}$ 和 $t_0^{2^{r-1}} \equiv 1 \pmod{q^e}$：

模 q^e 下的二次剩余

不妨假设已经有 $y^2 = nt$ 和 $t^{2^r} \equiv 1 \pmod{q^e}$，我们可以推出 $y_0^2 \equiv nt_0 \pmod{q}$ 和 $t_0^{2^{r-1}} \equiv 1 \pmod{q^e}$：

- 若 $t^{2^{r-1}} \equiv 1 \pmod{q^e}$，直接可得一组 y_0 和 t_0

模 q^e 下的二次剩余

不妨假设已经有 $y^2 = nt$ 和 $t^{2^r} \equiv 1 \pmod{q^e}$，我们可以推出 $y_0^2 \equiv nt_0 \pmod{q}$ 和 $t_0^{2^{r-1}} \equiv 1 \pmod{q^e}$：

- 若 $t^{2^{r-1}} \equiv 1 \pmod{q^e}$，直接可得一组 y_0 和 t_0
- 若 $t^{2^{r-1}} \equiv -1 \pmod{q^e}$，我们找到一个 z 满足 $z^{2^{m-1}Q} \equiv -1 \pmod{q}$，我们需要找到一个 w 满足 $w^{2^r} \equiv -1 \pmod{q^e}$ 从而 $(tw^2)^{2^{r-1}} \equiv 1 \pmod{q^e}$，容易发现 $w = z^{Q2^{m-r-1}}$ 符合要求。

模 q^e 下的二次剩余

不妨假设已经有 $y^2 = nt$ 和 $t^{2^r} \equiv 1 \pmod{q^e}$，我们可以推出 $y_0^2 \equiv nt_0 \pmod{q}$ 和 $t_0^{2^{r-1}} \equiv 1 \pmod{q^e}$：

- 若 $t^{2^{r-1}} \equiv 1 \pmod{q^e}$，直接可得一组 y_0 和 t_0
- 若 $t^{2^{r-1}} \equiv -1 \pmod{q^e}$，我们找到一个 z 满足 $z^{2^{m-1}Q} \equiv -1 \pmod{q}$，我们需要找到一个 w 满足 $w^{2^r} \equiv -1 \pmod{q^e}$ 从而 $(tw^2)^{2^{r-1}} \equiv 1 \pmod{q^e}$，容易发现 $w = z^{Q2^{m-r-1}}$ 符合要求。

直到 $r = 0$，我们得到了 $x^2 \equiv n \pmod{q^e}$ 的一组解，是不是一模一样?

模 2^e 下的二次剩余

即求 $x^2 \equiv a \pmod{2^e}$ 的整数解，同样将问题转化为 a 为奇数的情形，首先考虑该式对哪些 a 的取值有解。

模 2^e 下的二次剩余

即求 $x^2 \equiv a \pmod{2^e}$ 的整数解，同样将问题转化为 a 为奇数的情形，首先考虑该式对哪些 a 的取值有解。

显然 x 为奇数，不妨设 $x = 2t + 1$，那么 $a \equiv 4t^2 + 4t + 1 \pmod{2^e}$，即总有 $a \equiv 1 \pmod{8}$，接下来我们尝试说明对于这样的 a 一定能找到 $x^2 \equiv a \pmod{2^e}$ 的解，接下来我前面只讨论 $e \geq 3$ 的情况：

模 2^e 下的二次剩余

- 当 $e = 3$，仅 $a = 1$ 时有解 $1, 3, 5, 7$。由于 $(x + 2^{e-1})^2 \equiv x^2 \pmod{2^e}$，我们只需保留两组解 1 和 3。

模 2^e 下的二次剩余

- 当 $e = 3$，仅 $a = 1$ 时有解 $1, 3, 5, 7$。由于 $(x + 2^{e-1})^2 \equiv x^2 \pmod{2^e}$，我们只需保留两组解 1 和 3。
- 当 $e > 3$，不妨假设我们求得了 $x^2 \equiv a \pmod{2^{e-1}}$ 的解，我们尝试推得 $x^2 \equiv a \pmod{2^e}$ 的解。不妨考虑在 $x^2 \equiv a \pmod{2^{e-1}}$ 的两个解 w 和 $w + 2^{e-2}$，容易发现 w^2 和 $(w + 2^{e-2})^2$ 在模 2^e 意义下恰好分别对应 a 和 $a + 2^{e-1}$，于是我们可以推得 $x^2 \equiv a \pmod{2^e}$ 下的四组解。

模 2^e 下的二次剩余

- 当 $e = 3$，仅 $a = 1$ 时有解 $1, 3, 5, 7$。由于 $(x + 2^{e-1})^2 \equiv x^2 \pmod{2^e}$，我们只需保留两组解 1 和 3。
- 当 $e > 3$，不妨假设我们求得了 $x^2 \equiv a \pmod{2^{e-1}}$ 的解，我们尝试推得 $x^2 \equiv a \pmod{2^e}$ 的解。不妨考虑在 $x^2 \equiv a \pmod{2^{e-1}}$ 的两个解 w 和 $w + 2^{e-2}$，容易发现 w^2 和 $(w + 2^{e-2})^2$ 在模 2^e 意义下恰好分别对应 a 和 $a + 2^{e-1}$，于是我们可以推得 $x^2 \equiv a \pmod{2^e}$ 下的四组解。

通过这个构造方法，我们可以证明 $a \equiv 1 \pmod 8$ 是 $x^2 \equiv a \pmod{2^e}$ 的充要条件，且当 $e \geq 3$ 总恰好有四组解，我们也可以直接推得这四组解。

模合数意义下的二次剩余

我们现在可以方便地解决求 $x^2 \equiv m$ 的所有解的问题了：只要将 m 因式分解，对每一部分求解后，再一一使用中国剩余定理合并即可。所有的子问题在上面都已经讨论过，这个问题得以在因式分解的时间复杂度内解决。

k 次剩余

接下来将问题拓展至 k 次剩余，即 $x^k \equiv a \pmod{p}$ 的问题，其中 p 为素数。

k 次剩余

接下来将问题拓展至 k 次剩余，即 $x^k \equiv a \pmod{p}$ 的问题，其中 p 为素数。

和二次剩余相似，我们仍有一个非常简单的解决办法：求原根、找指标、解同余方程。显然利用中国剩余定理，这个方法也可以拓展到 p 为奇合数的情况。

k 次剩余

接下来将问题拓展至 k 次剩余，即 $x^k \equiv a \pmod{p}$ 的问题，其中 p 为素数。

和二次剩余相似，我们仍有一个非常简单的解决办法：求原根、找指标、解同余方程。显然利用中国剩余定理，这个方法也可以拓展到 p 为奇合数的情况。

当 $p = 2^e$ 时，我们仍然采用之前类似的方法。

Root

给定素数 $p < q$，求 $x^p \equiv n \pmod q$ 的任意一组解。

$T \leq 10, p, q \leq 10^{18}$

我们继续扩展 Tonelli-Shanks 算法：考虑如何解决 $x^p \equiv n \pmod{q}$ 的问题，其中 p 和 q 均为素数。

Root

我们继续扩展 Tonelli-Shanks 算法：考虑如何解决 $x^p \equiv n \pmod{q}$ 的问题，其中 p 和 q 均为素数。

- 当 $q \not\equiv 1 \pmod{p}$ 时，只要取 p 在模 $q-1$ 意义下的逆元 w，即可得到 $x \equiv n^w \pmod{q}$，这一部分是 trival 的。

我们继续扩展 Tonelli-Shanks 算法：考虑如何解决 $x^p \equiv n \pmod{q}$ 的问题，其中 p 和 q 均为素数。

- 当 $q \not\equiv 1 \pmod{p}$ 时，只要取 p 在模 $q-1$ 意义下的逆元 w，即可得到 $x \equiv n^w \pmod{q}$，这一部分是 trival 的。
- 当 $q \equiv 1 \pmod{p}$ 时，不妨设 $q = p^m Q + 1$，其中 Q 模 p 不和 0 同余，令 w 为 $-Q$ 在模 p 意义下的逆元，令 $y = n^{\frac{wQ+1}{p}} \pmod{q}$ 则有

$$y^p \equiv n \times n^{wQ} \pmod{q}$$

我们继续扩展 Tonelli-Shanks 算法：考虑如何解决 $x^p \equiv n \pmod q$ 的问题，其中 p 和 q 均为素数。

- 当 $q \not\equiv 1 \pmod p$ 时，只要取 p 在模 $q-1$ 意义下的逆元 w，即可得到 $x \equiv n^w \pmod q$，这一部分是 trival 的。
- 当 $q \equiv 1 \pmod p$ 时，不妨设 $q = p^m Q + 1$，其中 Q 模 p 不和 0 同余，令 w 为 $-Q$ 在模 p 意义下的逆元，令 $y = n^{\frac{wQ+1}{p}} \pmod q$ 则有

$$y^p \equiv n \times n^{wQ} \pmod q$$

由于 n 存在 p 次剩余，因此 $n^{p^{m-1}Q} \equiv 1 \pmod q$。令 $t = n^{wQ} \pmod q$，则 $t^{p^{m-1}} \equiv 1 \pmod q$。

Root

假设已经有了 $y^p \equiv nt$ 和 $t^{p^r} \equiv 1 \pmod{q}$，我们要推出 $y_0^p \equiv nt_0$ 和 $t_0^{p^{r-1}} \equiv 1 \pmod{q}$。

Root

假设已经有了 $y^p \equiv nt$ 和 $t^{p^r} \equiv 1 \pmod{q}$，我们要推出 $y_0^p \equiv nt_0$ 和 $t_0^{p^{r-1}} \equiv 1 \pmod{q}$。

- 若 $t^{p^{r-1}} \equiv 1 \pmod{q}$，可以直接得到 y_0 和 t_0。

Root

假设已经有了 $y^p \equiv nt$ 和 $t^{p^r} \equiv 1 \pmod q$，我们要推出 $y_0^p \equiv nt_0$ 和 $t_0^{p^{r-1}} \equiv 1 \pmod q$。

- 若 $t^{p^{r-1}} \equiv 1 \pmod q$，可以直接得到 y_0 和 t_0。
- 否则，设 $t^{p^{r-1}} \equiv s \pmod q$，我们需要一个 w 满足 $w^{p^r} \equiv s^{-1} \pmod q$ 从而使得 $(tw^p)^{p^{r-1}} \equiv 1 \pmod q$，若存在一个 $z^{p^{m-1}Q} \equiv s^{-1}$ 即可取 $w = z^{Qp^{m-r-1}}$。

Root

假设已经有了 $y^p \equiv nt$ 和 $t^{p^r} \equiv 1 \pmod q$，我们要推出 $y_0^p \equiv nt_0$ 和 $t_0^{p^{r-1}} \equiv 1 \pmod q$。

- 若 $t^{p^{r-1}} \equiv 1 \pmod q$，可以直接得到 y_0 和 t_0。
- 否则，设 $t^{p^{r-1}} \equiv s \pmod q$，我们需要一个 w 满足 $w^{p^r} \equiv s^{-1} \pmod q$ 从而使得 $(tw^p)^{p^{r-1}} \equiv 1 \pmod q$，若存在一个 $z^{p^{m-1}Q} \equiv s^{-1}$ 即可取 $w = z^{Qp^{m-r-1}}$。
- 考虑如何处理这样的 z，不妨设原根为 g，则 $s^{-1} = g^{kp^{m-1}Q}$，取 $z = g^k$ 即可。

Root

假设已经有了 $y^p \equiv nt$ 和 $t^{p^r} \equiv 1 \pmod q$，我们要推出 $y_0^p \equiv nt_0$ 和 $t_0^{p^{r-1}} \equiv 1 \pmod q$。

- 若 $t^{p^{r-1}} \equiv 1 \pmod q$，可以直接得到 y_0 和 t_0。
- 否则，设 $t^{p^{r-1}} \equiv s \pmod q$，我们需要一个 w 满足 $w^{p^r} \equiv s^{-1} \pmod q$ 从而使得 $(tw^p)^{p^{r-1}} \equiv 1 \pmod q$，若存在一个 $z^{p^{m-1}Q} \equiv s^{-1}$ 即可取 $w = z^{Qp^{m-r-1}}$。
- 考虑如何处理这样的 z，不妨设原根为 g，则 $s^{-1} = g^{kp^{m-1}Q}$，取 $z = g^k$ 即可。

得到原根后，我们可以使用 BSGS 在 $\mathcal{O}(\sqrt{p})$ 的时间内求出 k 的值，因此复杂度为 $\mathcal{O}(\sqrt{p}\log q)$。

Root

假设已经有了 $y^p \equiv nt$ 和 $t^{p^r} \equiv 1 \pmod q$，我们要推出 $y_0^p \equiv nt_0$ 和 $t_0^{p^{r-1}} \equiv 1 \pmod q$。

- 若 $t^{p^{r-1}} \equiv 1 \pmod q$，可以直接得到 y_0 和 t_0。
- 否则，设 $t^{p^{r-1}} \equiv s \pmod q$，我们需要一个 w 满足 $w^{p^r} \equiv s^{-1} \pmod q$ 从而使得 $(tw^p)^{p^{r-1}} \equiv 1 \pmod q$，若存在一个 $z^{p^{m-1}Q} \equiv s^{-1}$ 即可取 $w = z^{Qp^{m-r-1}}$。
- 考虑如何处理这样的 z，不妨设原根为 g，则 $s^{-1} = g^{kp^{m-1}Q}$，取 $z = g^k$ 即可。

得到原根后，我们可以使用 BSGS 在 $\mathcal{O}(\sqrt{p})$ 的时间内求出 k 的值，因此复杂度为 $\mathcal{O}(\sqrt{p}\log q)$。

同样我们会发现，当 p 很大时我们可以使用 BSGS 在 $\mathcal{O}\left(\sqrt{\frac{q}{p}}\right)$ 的时间内求出任意一个 x，原根的判定需要对 q 做质因数分解，使用 Pollard-Rho 分解，总的复杂度为 $\mathcal{O}(q^{1/4}(\log q)^{1/2})$。

同样，该问题也可以拓展到模数为合数、指数无平方因子的问题。

整数分解问题

数论问题往往涉及到整数分解。为了进行整数分解，我们需要完成两件事：判定一个数是不是素数，以及对一个合数找到一个非平凡因子。接下来我们先对前半部分进行一些讨论。

素数测试: Fermat primality test

费马小定理告诉我们，对任意素数 p 和任意整数 a，总有 $a^p \equiv a \pmod{p}$。而我们可以发现这个定理的逆定理也“几乎成立”，于是我们就想要去随机一些 a 直接判断。

素数测试: Fermat primality test

费马小定理告诉我们，对任意素数 p 和任意整数 a，总有 $a^p \equiv a \pmod{p}$。而我们可以发现这个定理的逆定理也“几乎成立”，于是我们就想要去随机一些 a 直接判断。

但这个算法的正确性存在问题：由于存在无穷多个卡迈克尔数 p 使得对于任意 $0 < a < p, \gcd(a, p) = 1$ 都有 $a^p \equiv a \pmod{p}$。

素数测试: Miller-Rabin primality test

考虑改进 Fermat primality test 的方法：根据中国剩余定理，我们可以知道对于素数 p 来说 1 的二次剩余只有 ± 1，而对于非素数来说并非如此。

素数测试: Miller-Rabin primality test

考虑改进 Fermat primality test 的方法：根据中国剩余定理，我们可以知道对于素数 p 来说 1 的二次剩余只有 ± 1，而对于非素数来说并非如此。

我们将这个方法和刚刚的测试结合起来，随机检验 k 个底数的正确率可以达到 $1 - 4^{-k}$。我们将能通过某个底数的检测数叫做 strong Fermat pseudoprime。

素数测试: Lucas primality test

可以考虑将上述检测方式进行一些拓展。费马小定理考虑的是 n 的剩余系，考虑将其二次扩张到 $\mathbb{Q}(\sqrt{D})$。也就是说，在这个域中的每个元素都是 $a + b\sqrt{D}$。

可以考虑将上述检测方式进行一些拓展。费马小定理考虑的是 n 的剩余系，考虑将其二次扩张到 $\mathbb{Q}(\sqrt{D})$。也就是说，在这个域中的每个元素都是 $a + b\sqrt{D}$。

如果 p 是素数，易知：

$$(a + b\sqrt{D})^p \equiv a + b\left(\frac{D}{p}\right)\sqrt{D}$$

其中 $\left(\frac{D}{p}\right)$ 是二次剩余的 Jacobi 符号。

素数测试: Lucas primality test

可以利用 $a+b\sqrt{D}$ 和 $a-b\sqrt{D}$ 为特征根来构造二阶线性递推序列：不妨取一组 $P^2-4Q=D$，定义递推序列 U 满足 $U_0=0, U_1=1, U_n=PU_{n-1}-QU_{n-2}$，可以得到序列 U 的通项公式：

$$U_n=\frac{1}{\sqrt{D}}\left(\left(\frac{P+\sqrt{D}}{2}\right)^n-\left(\frac{P-\sqrt{D}}{2}\right)^n\right)$$

素数测试: Lucas primality test

可以利用 $a+b\sqrt{D}$ 和 $a-b\sqrt{D}$ 为特征根来构造二阶线性递推序列：不妨取一组 $P^2-4Q=D$，定义递推序列 U 满足 $U_0=0, U_1=1, U_n=PU_{n-1}-QU_{n-2}$，可以得到序列 U 的通项公式：

$$U_n=\frac{1}{\sqrt{D}}\left(\left(\frac{P+\sqrt{D}}{2}\right)^n-\left(\frac{P-\sqrt{D}}{2}\right)^n\right)$$

考虑 $\left(\frac{D}{p}\right)=-1$ 的情况，容易发现若 n 是素数，有：

$$U_{n+1}\equiv 0 \pmod{n}$$

素数测试: Lucas primality test

可以利用 $a+b\sqrt{D}$ 和 $a-b\sqrt{D}$ 为特征根来构造二阶线性递推序列：不妨取一组 $P^2-4Q=D$，定义递推序列 U 满足 $U_0=0, U_1=1, U_n=PU_{n-1}-QU_{n-2}$，可以得到序列 U 的通项公式：

$$U_n=\frac{1}{\sqrt{D}}\left(\left(\frac{P+\sqrt{D}}{2}\right)^n-\left(\frac{P-\sqrt{D}}{2}\right)^n\right)$$

考虑 $\left(\frac{D}{p}\right)=-1$ 的情况，容易发现若 n 是素数，有：

$$U_{n+1}\equiv 0 \pmod{n}$$

我们同样可以考虑利用该式进行检验。我们可以取 $P=1, D=5,-7,9,-11\ldots$ 来生成这样的序列。同样，该方法也会产生伪素数——我们可以称其为 Lucas pseudoprime。

素数测试: strong Lucas probable prime test

和 Fermat primality test 类似，该方法同样存在一个更为强大的方法，不妨令 a 和 b 是 $x^2 - Px - Q = 0$ 的根，我们有：

$$U_n = \frac{a^n - b^n}{\sqrt{D}}$$

素数测试: strong Lucas probable prime test

和 Fermat primality test 类似，该方法同样存在一个更为强大的方法，不妨令 a 和 b 是 $x^2 - Px - Q = 0$ 的根，我们有：

$$U_n = \frac{a^n - b^n}{\sqrt{D}}$$

我们可以考虑构造另一个二阶递推数列 V 满足：

$$V_n = a^n + b^n$$

素数测试: strong Lucas probable prime test

和 Fermat primality test 类似，该方法同样存在一个更为强大的方法，不妨令 a 和 b 是 $x^2 - Px - Q = 0$ 的根，我们有：

$$U_n = \frac{a^n - b^n}{\sqrt{D}}$$

我们可以考虑构造另一个二阶递推数列 V 满足：

$$V_n = a^n + b^n$$

那么 $U_n = U_{\frac{n}{2}} V_{\frac{n}{2}}$，我们判断 U_{n+1} 是否为 0，可以同样将 $n+1$ 分解成 $2^r \times Q$ 的形式，并对 U 和 V 一起检验。

素数测试: strong Lucas probable prime test

和 Fermat primality test 类似，该方法同样存在一个更为强大的方法，不妨令 a 和 b 是 $x^2 - Px - Q = 0$ 的根，我们有：

$$U_n = \frac{a^n - b^n}{\sqrt{D}}$$

我们可以考虑构造另一个二阶递推数列 V 满足：

$$V_n = a^n + b^n$$

那么 $U_n = U_{\frac{n}{2}} V_{\frac{n}{2}}$，我们判断 U_{n+1} 是否为 0，可以同样将 $n+1$ 分解成 $2^r \times Q$ 的形式，并对 U 和 V 一起检验。

对于仍然能够通过测试的数，我们可以称其为 strong Lucas pseudoprime。

素数测试: Baillie-PSW primality test

上面的两个方法都具有以下特点：存在伪素数，但伪素数分布比较稀少。我们自然可以想到将这两个方法拼在一起。Baillie-PSW primality test 就是做了这样的一个事情，我们检验 n 是否为素数，首先做一次 base = 2 的 Miller-Rabin primality test，再取 $P = 1, D = 5, -7, 9, -11 \ldots$ 中第一个满足 $\left(\frac{D}{p}\right) = -1$ 的 D 进行 strong Lucas probable prime test，如果均通过我们就认为该数是质数。

素数测试: Baillie-PSW primality test

上面的两个方法都具有以下特点：存在伪素数，但伪素数分布比较稀少。我们自然可以想到将这两个方法拼在一起。Baillie-PSW primality test 就是做了这样的一个事情，我们检验 n 是否为素数，首先做一次 base = 2 的 Miller-Rabin primality test，再取 $P = 1, D = 5, -7, 9, -11\ldots$ 中第一个满足 $\left(\frac{D}{p}\right) = -1$ 的 D 进行 strong Lucas probable prime test，如果均通过我们就认为该数是质数。

目前该方法的正确性仍然是一个 open problem。但该方法的一个弱化版本 (用 Fibonacci test 代替 strong Lucas probable prime test) 已经被证明存在反例——虽然仍未找到一个确实的反例。

整数分解: Pollard's Rho 算法

重新回到整数分解的问题，我们想要找到一个 n 的非平凡因子 $1 < p \leq \sqrt{n}$。这里 John Pollard 提出了一种比较为大家熟知的方法：

整数分解: Pollard's Rho 算法

重新回到整数分解的问题，我们想要找到一个 n 的非平凡因子 $1 < p \leq \sqrt{n}$。这里 John Pollard 提出了一种比较为大家熟知的方法：

考虑构造一个随机函数 f，使得 $f(x)$ 在 $\bmod p$ 意义下几乎随机，且有 $f(x) \equiv f(x+p) \pmod{p}$。

整数分解: Pollard's Rho 算法

重新回到整数分解的问题，我们想要找到一个 n 的非平凡因子 $1 < p \leq \sqrt{n}$。这里 John Pollard 提出了一种比较为大家熟知的方法：

考虑构造一个随机函数 f，使得 $f(x)$ 在 mod$p$ 意义下几乎随机，且有 $f(x) \equiv f(x+p) \pmod{p}$。

我们不断将 x 变成 $f(x)$，设 x 如此做了 m 次变换后的值为 $f_m(x)$，因为 $f_m(x) \bmod p$ 的值只有有限个，那么一定存在一个最小的 u，满足存在一个最小的 v，使得对任意 $y \geq u$ 有 $f_y(x) \equiv f_{y+v}(x) \pmod{p}$，也就是在 u 步之后走入了一个长为 v 的环中。

整数分解: Pollard's Rho 算法

重新回到整数分解的问题，我们想要找到一个 n 的非平凡因子 $1 < p \leq \sqrt{n}$。这里 John Pollard 提出了一种比较为大家熟知的方法：

考虑构造一个随机函数 f，使得 $f(x)$ 在 mod$p$ 意义下几乎随机，且有 $f(x) \equiv f(x+p) \pmod{p}$。

我们不断将 x 变成 $f(x)$，设 x 如此做了 m 次变换后的值为 $f_m(x)$，因为 $f_m(x) \bmod p$ 的值只有有限个，那么一定存在一个最小的 u，满足存在一个最小的 v，使得对任意 $y \geq u$ 有 $f_y(x) \equiv f_{y+v}(x) \pmod{p}$，也就是在 u 步之后走入了一个长为 v 的环中。

我们想要知道在 mod$p$ 意义下 $u+v$ 的期望，由于 $f_m(x)$ 几乎随机，可以看成 $u+v$ 个随机数中有一对相等时，$u+v$ 的期望。而这时有 $\frac{(u+v)(u+v-1)}{2}$ 对数，每一对相等的期望都是 $\frac{1}{p}$，因此我们可以知道，$u+v$ 在期望意义下是 $\mathcal{O}(\sqrt{p})$ 的。

整数分解: Pollard's Rho 算法

由于保证了 $f(x) \equiv f(x+p) \pmod{p}$，函数在 $\bmod p$ 和 $\bmod n$ 的意义下循环节是不同的，也就是说，我们可以找到一对数在 $\bmod p$ 下相等，但在 $\bmod n$ 下不等，于是问题解决。

整数分解: Pollard's Rho 算法

由于保证了 $f(x) \equiv f(x+p) \pmod{p}$，函数在 $\bmod p$ 和 $\bmod n$ 的意义下循环节是不同的，也就是说，我们可以找到一对数在 $\bmod p$ 下相等，但在 $\bmod n$ 下不等，于是问题解决。

在实际测试中我们通常取 $f(x) = x^2 + c$，于是根据我们之前的方法，我们可以得到一个 n 的非平凡因数 a 。在实现时，我们可以通过初始时相等的两个数 x 和 y，每次将 x 变为 $f(x)$，y 变为 $f(f(y))$，并检查 $\gcd(x-y, n)$；也可以采用倍增环长的方法。

整数分解: Pollard's Rho 算法

由于保证了 $f(x) \equiv f(x+p) \pmod p$，函数在 $\bmod p$ 和 $\bmod n$ 的意义下循环节是不同的，也就是说，我们可以找到一对数在 $\bmod p$ 下相等，但在 $\bmod n$ 下不等，于是问题解决。

在实际测试中我们通常取 $f(x) = x^2 + c$，于是根据我们之前的方法，我们可以得到一个 n 的非平凡因数 a 。在实现时，我们可以通过初始时相等的两个数 x 和 y，每次将 x 变为 $f(x)$，y 变为 $f(f(y))$，并检查 $\gcd(x-y, n)$；也可以采用倍增环长的方法。

由于找一个因数的复杂度是 $\mathcal{O}(\sqrt{a})$，而 n 的非最大质因子的所有数之和是 $\mathcal{O}(\sqrt{n})$ 的，于是我们一共会走 $\mathcal{O}(n^{\frac{1}{4}})$ 步，算上 gcd 的复杂度就是 $\mathcal{O}(n^{\frac{1}{4}} \log n)$。

整数分解: Pollard's Rho 算法

由于保证了 $f(x) \equiv f(x+p) \pmod p$，函数在 $\bmod p$ 和 $\bmod n$ 的意义下循环节是不同的，也就是说，我们可以找到一对数在 $\bmod p$ 下相等，但在 $\bmod n$ 下不等，于是问题解决。

在实际测试中我们通常取 $f(x) = x^2 + c$，于是根据我们之前的方法，我们可以得到一个 n 的非平凡因数 a 。在实现时，我们可以通过初始时相等的两个数 x 和 y，每次将 x 变为 $f(x)$，y 变为 $f(f(y))$，并检查 $\gcd(x-y, n)$；也可以采用倍增环长的方法。

由于找一个因数的复杂度是 $\mathcal{O}(\sqrt{a})$，而 n 的非最大质因子的所有数之和是 $\mathcal{O}(\sqrt{n})$ 的，于是我们一共会走 $\mathcal{O}(n^{\frac{1}{4}})$ 步，算上 gcd 的复杂度就是 $\mathcal{O}(n^{\frac{1}{4}} \log n)$。

目前的瓶颈在于过多次无谓的 gcd 操作。但我们可以发现 $\gcd(x, n) \mid \gcd(xy, n)$ ，因此我们在判断是否有非平凡因子时，可以多次合并在一起判断。

离散对数: Pollard's Rho 算法

使用类似的思想，我们也可以得到一个求离散对数的时间复杂度 $\mathcal{O}(\sqrt{n})$，空间复杂度 $\mathcal{O}(1)$ 的方法——相比传统的 BSGS 做法在空间上占据绝对优势。

离散对数: Pollard's Rho 算法

使用类似的思想，我们也可以得到一个求离散对数的时间复杂度 $\mathcal{O}(\sqrt{n})$，空间复杂度 $\mathcal{O}(1)$ 的方法——相比传统的 BSGS 做法在空间上占据绝对优势。

考虑我们现在有一个生成元 x 和 x 生成的元素 y，我们要求 $x^e = y$ 中的 e，同时我们知道这个群的阶 n。

离散对数: Pollard's Rho 算法

使用类似的思想，我们也可以得到一个求离散对数的时间复杂度 $\mathcal{O}(\sqrt{n})$，空间复杂度 $\mathcal{O}(1)$ 的方法——相比传统的 BSGS 做法在空间上占据绝对优势。

考虑我们现在有一个生成元 x 和 x 生成的元素 y，我们要求 $x^e = y$ 中的 e，同时我们知道这个群的阶 n。

不妨假设我们现在有了 $x^{\alpha_1} y^{\beta_1} = x^{\alpha_2} y^{\beta_2}$，我们就有很大概率能够求得 e 的值。因此问题在于如何求一组这样的解。

离散对数: Pollard's Rho 算法

同样构造一个随机函数 $f(t)$：我们可以将群中的元素随机分为 3 个集合，对第一个集合的元素 t 有 $f(t) = ty$，第二个集合有 $f(t) = t^2$，第三个集合有 $f(t) = tx$。那么集合中的元素可以看做随机生成，类比分解质因数的方法，我们可以得到这个方法的复杂度是 $\mathcal{O}(\sqrt{n})$ 的。

整数分解: 指数级到次指数级

重新回到整数分解问题，我们之前介绍的 Pollard Rho Algorithm 只是一个指数级算法。突破指数级的思路来自于整数仍具有其他的性质：n 以内的最大素因子不超过 $n^{\frac{1}{\alpha}}$ 的数 (smooth number) 的个数趋于 $n\alpha^{-\alpha}$。

整数分解: 指数级到次指数级

重新回到整数分解问题，我们之前介绍的 Pollard Rho Algorithm 只是一个指数级算法。突破指数级的思路来自于整数仍具有其他的性质：n 以内的最大素因子不超过 $n^{\frac{1}{\alpha}}$ 的数 (smooth number) 的个数趋于 $n\alpha^{-\alpha}$。

可以发现，若存在 $x^2 \equiv y^2 \pmod{n}$ 且 $x \pm y \not\equiv 0 \pmod{n}$，我们就找到了 n 的一个非平凡因子 $\gcd(x+y, n)$，问题是如何生成不那么平凡的 x 和 y。

整数分解: 指数级到次指数级

重新回到整数分解问题，我们之前介绍的 Pollard Rho Algorithm 只是一个指数级算法。突破指数级的思路来自于整数仍具有其他的性质：n 以内的最大素因子不超过 $n^{\frac{1}{\alpha}}$ 的数 (smooth number) 的个数趋于 $n\alpha^{-\alpha}$。

可以发现，若存在 $x^2 \equiv y^2 \pmod{n}$ 且 $x \pm y \not\equiv 0 \pmod{n}$，我们就找到了 n 的一个非平凡因子 $\gcd(x+y, n)$，问题是如何生成不那么平凡的 x 和 y。

一种简单的想法是检查 $x^2 \bmod n$ 是否为完全平方数，如果是我们就找到了一组这样的 y，但是 n 以内的完全平方数只有 $\sqrt{n}$ 个，那么我们随机找到一个完全平方数的概率不会超过 $\frac{1}{\sqrt{n}}$，显然不太行。

整数分解: 指数级到次指数级

不过我们可以退而求其次：我们可以保留一些比较 smooth 的 $x^2 \bmod n$，即最大质因子较小的这些 $x^2 \bmod n$，那么接下来我们可以将每个这样的数分解成 $\prod p_i^{b_i}$ 的形式，而我们接下来只要找一个序列 $\{x\}$ 使得 $\prod(x_i^2 \bmod n) = \prod p_i^{\sum b_{j,i}}$ 使得右侧是一个完全平方数即可，也即指数项均为偶数。

整数分解: 指数级到次指数级

不过我们可以退而求其次：我们可以保留一些比较 smooth 的 $x^2 \bmod n$，即最大质因子较小的这些 $x^2 \bmod n$，那么接下来我们可以将每个这样的数分解成 $\prod p_i^{b_i}$ 的形式，而我们接下来只要找一个序列 $\{x\}$ 使得 $\prod (x_i^2 \bmod n) = \prod p_i^{\sum b_{j,i}}$ 使得右侧是一个完全平方数即可，也即指数项均为偶数。

这是一个经典的线性代数问题，可以使用高斯消元法解决。接下来我们只要找到一种选出这些比较 smooth 的 $x^2 \bmod n$ 的方法就好了。

整数分解: 指数级到次指数级

不过我们可以退而求其次：我们可以保留一些比较 smooth 的 $x^2 \bmod n$，即最大质因子较小的这些 $x^2 \bmod n$，那么接下来我们可以将每个这样的数分解成 $\prod p_i^{b_i}$ 的形式，而我们接下来只要找一个序列 $\{x\}$ 使得 $\prod (x_i^2 \bmod n) = \prod p_i^{\sum b_{j,i}}$ 使得右侧是一个完全平方数即可，也即指数项均为偶数。

这是一个经典的线性代数问题，可以使用高斯消元法解决。接下来我们只要找到一种选出这些比较 smooth 的 $x^2 \bmod n$ 的方法就好了。

我们可以随机大量的 x，采用暴力分解的方法。虽然这个方法看起来很暴力，但经过一些简单的分析，可以得到该算法的复杂度是 $\mathcal{O}\left(\exp\left((2+o(1))\sqrt{\ln n \ln\ln n}\right)\right)$ 的。

整数分解: 指数级到次指数级

不过我们可以退而求其次：我们可以保留一些比较 smooth 的 $x^2 \bmod n$，即最大质因子较小的这些 $x^2 \bmod n$，那么接下来我们可以将每个这样的数分解成 $\prod p_i^{b_i}$ 的形式，而我们接下来只要找一个序列 $\{x\}$ 使得 $\prod(x_i^2 \bmod n) = \prod p_i^{\sum b_{j,i}}$ 使得右侧是一个完全平方数即可，也即指数项均为偶数。

这是一个经典的线性代数问题，可以使用高斯消元法解决。接下来我们只要找到一种选出这些比较 smooth 的 $x^2 \bmod n$ 的方法就好了。

我们可以随机大量的 x，采用暴力分解的方法。虽然这个方法看起来很暴力，但经过一些简单的分析，可以得到该算法的复杂度是 $\mathcal{O}\left(\exp\left((2+o(1))\sqrt{\ln n \ln\ln n}\right)\right)$ 的。

值得注意的是，该算法对于 $n = p^q$ 型的数是无效的，需要进行特殊处理。

整数分解: Quadratic Sieve

刚刚的方法的问题在于，最大素因子不超过 $n^{\frac{1}{\alpha}}$ 的数实在是太少了，同时对每个数逐一进行分解也会消耗很多的时间。我们需要对这样的部分进行优化。

整数分解: Quadratic Sieve

刚刚的方法的问题在于，最大素因子不超过 $n^{\frac{1}{\alpha}}$ 的数实在是太少了，同时对每个数逐一进行分解也会消耗很多的时间。我们需要对这样的部分进行优化。

容易发现，越小的数 smooth 的概率越大。由于这里 α 已经不是常数，这一部分的优化是必要的。

整数分解: Quadratic Sieve

刚刚的方法的问题在于，最大素因子不超过 $n^{\frac{1}{\alpha}}$ 的数实在是太少了，同时对每个数逐一进行分解也会消耗很多的时间。我们需要对这样的部分进行优化。

容易发现，越小的数 smooth 的概率越大。由于这里 α 已经不是常数，这一部分的优化是必要的。

令 $t = \lceil \sqrt{n} \rceil$，我们考虑 $f(x) = (t+x)^2 - n$，容易发现第 x 数的值是 xy 级别的。我们可以检查连续的若干个这样的数，这样值域被控制在了约 $\sqrt{n}$ 级别。

整数分解: Quadratic Sieve

刚刚的方法的问题在于，最大素因子不超过 $n^{\frac{1}{\alpha}}$ 的数实在是太少了，同时对每个数逐一进行分解也会消耗很多的时间。我们需要对这样的部分进行优化。

容易发现，越小的数 smooth 的概率越大。由于这里 α 已经不是常数，这一部分的优化是必要的。

令 $t = \lceil \sqrt{n} \rceil$，我们考虑 $f(x) = (t+x)^2 - n$，容易发现第 x 数的值是 xy 级别的。我们可以检查连续的若干个这样的数，这样值域被控制在了约 $\sqrt{n}$ 级别。

注意到 $p|f(i) \Rightarrow p|f(i+p)$，我们可以使用二次剩余和区间筛筛出所有 smooth 的数。算法实现时，可以取多个类似的二次函数 f，筛选后也可以将剩余公共部分相同的合并，可以提高一定的效率。

整数分解: Quadratic Sieve

刚刚的方法的问题在于，最大素因子不超过 $n^{\frac{1}{\alpha}}$ 的数实在是太少了，同时对每个数逐一进行分解也会消耗很多的时间。我们需要对这样的部分进行优化。

容易发现，越小的数 smooth 的概率越大。由于这里 α 已经不是常数，这一部分的优化是必要的。

令 $t = \lceil \sqrt{n} \rceil$，我们考虑 $f(x) = (t+x)^2 - n$，容易发现第 x 数的值是 xy 级别的。我们可以检查连续的若干个这样的数，这样值域被控制在了约 $\sqrt{n}$ 级别。

注意到 $p|f(i) \Rightarrow p|f(i+p)$，我们可以使用二次剩余和区间筛筛出所有 smooth 的数。算法实现时，可以取多个类似的二次函数 f，筛选后也可以将剩余公共部分相同的合并，可以提高一定的效率。

该算法的渐进时间复杂度为 $\mathcal{O}\left(\exp\left((1+o(1))\sqrt{\ln n \ln\ln n}\right)\right)$(理论上达到该复杂度需要使用一些稀疏矩阵高斯消元的方法)。

整数分解: Multi Polynomial Quadratic Sieve

为了提高刚刚算法的效率，我们尝试考虑多个多项式 $f(x) = Ax^2 + 2Bx + C$，对每个多项式筛选 $[-M, M]$ 之间的所有数。

整数分解: Multi Polynomial Quadratic Sieve

为了提高刚刚算法的效率，我们尝试考虑多个多项式 $f(x) = Ax^2 + 2Bx + C$，对每个多项式筛选 $[-M, M]$ 之间的所有数。

由于 $f(x) = A\left(x + \frac{B}{A}\right)^2 - \frac{B^2 - AC}{A}$，为了产生 $x^2 \equiv y^2$ 的左半部分，可以取 $B^2 \equiv N \pmod{A}$，即 N 在 mod$A$ 下的二次剩余来解出 C。我们也可以考虑让 $A = D^2$，则有：

$$f(x) = \left(Dx + \frac{B}{D}\right)^2$$

为了提高刚刚算法的效率，我们尝试考虑多个多项式 $f(x) = Ax^2 + 2Bx + C$，对每个多项式筛选 $[-M, M]$ 之间的所有数。

由于 $f(x) = A\left(x + \frac{B}{A}\right)^2 - \frac{B^2 - AC}{A}$，为了产生 $x^2 \equiv y^2$ 的左半部分，可以取 $B^2 \equiv N \pmod{A}$，即 N 在 mod$A$ 下的二次剩余来解出 C。我们也可以考虑让 $A = D^2$，则有：

$$f(x) = \left(Dx + \frac{B}{D}\right)^2$$

为了提高 $f(x) \equiv 0 \pmod{p}$ 的概率，我们可以将 D 设置为素数；同时，为了更快解决二次剩余问题，我们可以进一步设置 $D \equiv 3 \pmod{4}$，于是我们可以用 Tonelli-Shanks 算法直接推出答案。

这样做还有一个好处：因为 $B^2 - AC = N$，$Ax^2 + 2Bx + C \equiv 0 \pmod{p}$ 有解当且仅当 N 在模 p 下有二次剩余。因此我们的 Factor Base 大小会减小一半，同时在筛法过程中省去了二次剩余的计算。

这样做还有一个好处：因为 $B^2 - AC = N$，$Ax^2 + 2Bx + C \equiv 0 \pmod{p}$ 有解当且仅当 N 在模 p 下有二次剩余。因此我们的 Factor Base 大小会减小一半，同时在筛法过程中省去了二次剩余的计算。

我们尝试使得 $f(x)$ 的极值尽量小，容易发现 $A \approx \frac{\sqrt{2N}}{M}$ 时较优。

整数分解与离散对数: Index Calculus 算法

我们也可以利用上面类似的思想得到一个解决离散对数问题的次指数算法。

整数分解与离散对数: Index Calculus 算法

我们也可以利用上面类似的思想得到一个解决离散对数问题的次指数算法。

考虑在模 q^e 意义下的离散对数问题。

整数分解与离散对数: Index Calculus 算法

我们也可以利用上面类似的思想得到一个解决离散对数问题的次指数算法。

考虑在模 q^e 意义下的离散对数问题。

注意到

$$g^w \equiv \prod p_i^{t_i} \pmod{q^e} \Leftrightarrow w \equiv \sum t_i \log(p_i) \pmod{\varphi(q^e)}$$
$$xg^w \equiv \prod p_i^{t_i} \pmod{q^e} \Leftrightarrow x \equiv \sum t_i \log(p_i) - w \pmod{\varphi(q^e)}$$

整数分解与离散对数: Index Calculus 算法

我们也可以利用上面类似的思想得到一个解决离散对数问题的次指数算法。

考虑在模 q^e 意义下的离散对数问题。

注意到

$$g^w \equiv \prod p_i^{t_i} \pmod{q^e} \Leftrightarrow w \equiv \sum t_i \log(p_i) \pmod{\varphi(q^e)}$$
$$xg^w \equiv \prod p_i^{t_i} \pmod{q^e} \Leftrightarrow x \equiv \sum t_i \log(p_i) - w \pmod{\varphi(q^e)}$$

只要随机大量的 w 直到有解即可。

椭圆曲线群

实际上整数分解与模意义下的离散对数具有很大的相似性，这两个问题都是 BQP 的，因此未来这样的问题可能会变得容易解决。但这是由于模域的特殊性质所导致的。接下来我们来介绍一点椭圆曲线和椭圆曲线群的相关知识，并了解一些简单应用。

椭圆曲线群

实际上整数分解与模意义下的离散对数具有很大的相似性，这两个问题都是 BQP 的，因此未来这样的问题可能会变得容易解决。但这是由于模域的特殊性质所导致的。接下来我们来介绍一点椭圆曲线和椭圆曲线群的相关知识，并了解一些简单应用。

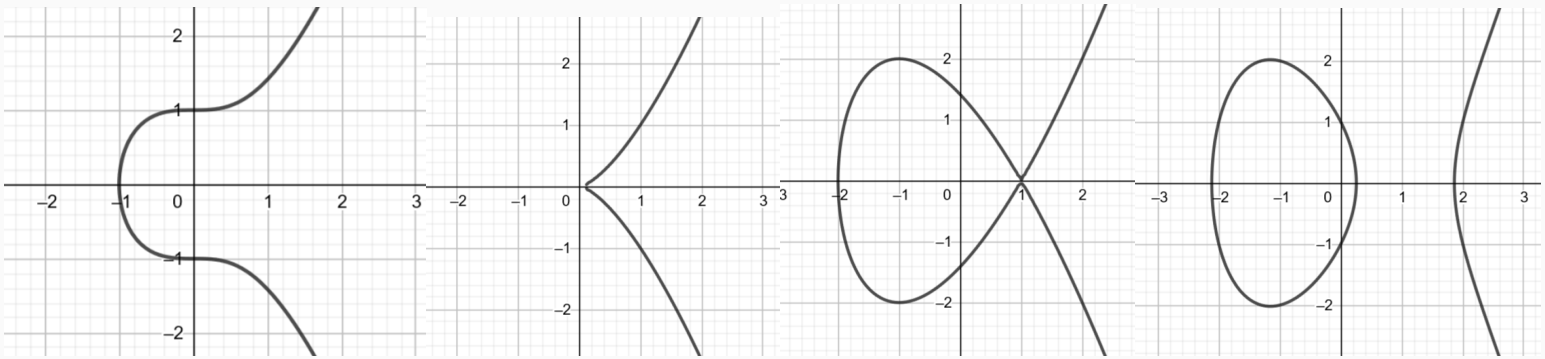
椭圆曲线是平面上的满足 $y^2 = x^3 + ax^2 + bx + c$ 的点 (x, y) 的集合。显然我们对 x 平移可以消去 x^2 项，因此我们下面讨论 $y^2 = x^3 + ax + b$ 的情形。

椭圆曲线群

令 $D = 4a^3 + 27b^2$，椭圆曲线 $y^2 = x^3 + ax + b$ 有以下性质：当 $D > 0$ 时，函数恰有一个零点；当 $D = 0$ 时，函数有重复零点（一个重复三次的零点或是一个重复两次、一个重复一次的零点）；当 $D < 0$ 时，函数有三个零点。

椭圆曲线群

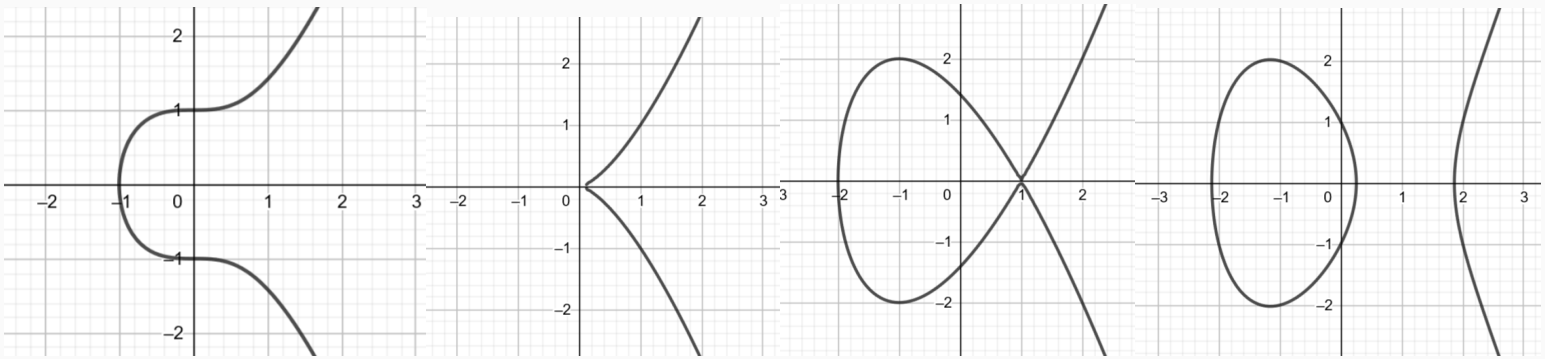
令 $D = 4a^3 + 27b^2$，椭圆曲线 $y^2 = x^3 + ax + b$ 有以下性质：当 $D > 0$ 时，函数恰有一个零点；当 $D = 0$ 时，函数有重复零点（一个重复三次的零点或是一个重复两次、一个重复一次的零点）；当 $D < 0$ 时，函数有三个零点。



上图为 $y^2 = x^3 + 1, y^2 = x^3, y^2 = x^3 - 3x + 2, y^2 = x^3 - 4x + 1$ 的图像。

椭圆曲线群

令 $D = 4a^3 + 27b^2$，椭圆曲线 $y^2 = x^3 + ax + b$ 有以下性质：当 $D > 0$ 时，函数恰有一个零点；当 $D = 0$ 时，函数有重复零点（一个重复三次的零点或是一个重复两次、一个重复一次的零点）；当 $D < 0$ 时，函数有三个零点。



上图为 $y^2 = x^3 + 1, y^2 = x^3, y^2 = x^3 - 3x + 2, y^2 = x^3 - 4x + 1$ 的图像。

接下来我们忽略 $D = 0$ 的情况，于是函数不会有重复零点。

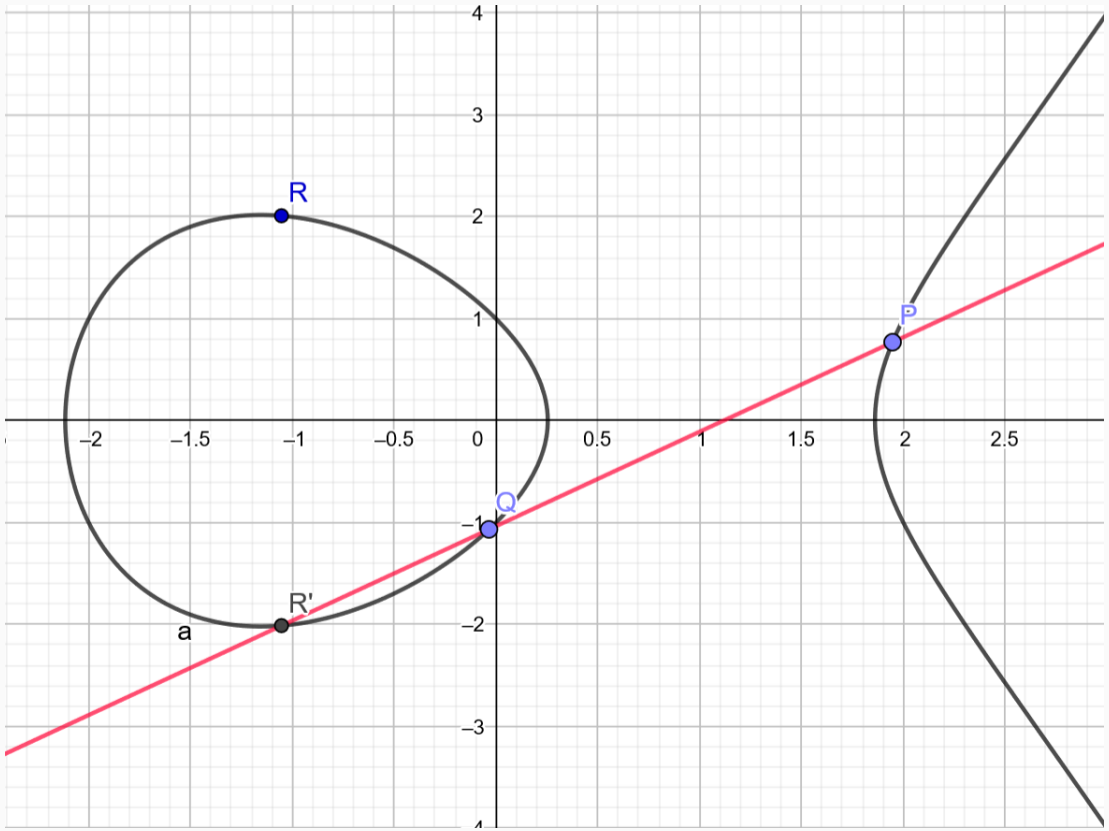
椭圆曲线群

接下来我们对集合 $S = \{(x, y) : y^2 = x^3 + ax + b\} \cup \{\infty\}$ 中的元素定义一种加法：首先我们对 $P \in S \backslash \{\infty\}$ 定义 $-P$ 为 P 在平面上的对称点，同时定义 $-\infty = \infty$。

椭圆曲线群

接下来我们对集合 $S = \{(x, y) : y^2 = x^3 + ax + b\} \cup \{\infty\}$ 中的元素定义一种加法：首先我们对 $P \in S \backslash \{\infty\}$ 定义 $-P$ 为 P 在平面上的对称点，同时定义 $-\infty = \infty$。

当 $P \neq Q$ 且 $P \neq -Q$ 时，我们可以如下图所示定义 $P + Q = R$：



当 $P = -Q$ 时，我们定义 $P + Q = \infty$。

当 $P = -Q$ 时，我们定义 $P + Q = \infty$。

我们定义 $\infty + P = P + \infty = P$。

当 $P = -Q$ 时，我们定义 $P + Q = \infty$。

我们定义 $\infty + P = P + \infty = P$。

当 $P = Q$ 或 PQ 和椭圆曲线没有交点时应该怎么办？当 $P = Q$ 时，我们作椭圆曲线上过点 P 的切线，将切线和曲线的交点定义为 R' 点。当 PQ 和椭圆曲线没有更多的交点，那么其一定是一条切线。我们将切线的切点定义为 R' 点。

椭圆曲线群

当 $P = -Q$ 时，我们定义 $P + Q = \infty$。

我们定义 $\infty + P = P + \infty = P$。

当 $P = Q$ 或 PQ 和椭圆曲线没有交点时应该怎么办？当 $P = Q$ 时，我们作椭圆曲线上过点 P 的切线，将切线和曲线的交点定义为 R' 点。当 PQ 和椭圆曲线没有更多的交点，那么其一定是一条切线。我们将切线的切点定义为 R' 点。

实际上我们计算椭圆曲线的加法时，就是已知三次方程的两个根，求另一个根的过程。

椭圆曲线群

我们说，集合 S 在运算 $+$ 下构成一个阿贝尔群：交换律、结合律、逆元、单位元。

椭圆曲线群

我们说，集合 S 在运算 $+$ 下构成一个阿贝尔群：交换律、结合律、逆元、单位元。

容易发现，若 P 和 Q 是有理点，那么 $P+Q$ 一定也是一个有理点。因此椭圆曲线上的所有有理点和一个我们定义的 ∞ 点同样在运算 $+$ 下构成一个阿贝尔群。

椭圆曲线群

我们说，集合 S 在运算 $+$ 下构成一个阿贝尔群：交换律、结合律、逆元、单位元。

容易发现，若 P 和 Q 是有理点，那么 $P+Q$ 一定也是一个有理点。因此椭圆曲线上的所有有理点和一个我们定义的 ∞ 点同样在运算 $+$ 下构成一个阿贝尔群。

于是我们可以尝试将其扩展到素数 p 模域下：对于模域下满足椭圆曲线方程的所有点和一个我们定义的 ∞ 点来说，它们同样在运算 $+$ 下构成一个阿贝尔群。

椭圆曲线群

我们说，集合 S 在运算 $+$ 下构成一个阿贝尔群：交换律、结合律、逆元、单位元。

容易发现，若 P 和 Q 是有理点，那么 $P+Q$ 一定也是一个有理点。因此椭圆曲线上的所有有理点和一个我们定义的 ∞ 点同样在运算 $+$ 下构成一个阿贝尔群。

于是我们可以尝试将其扩展到素数 p 模域下：对于模域下满足椭圆曲线方程的所有点和一个我们定义的 ∞ 点来说，它们同样在运算 $+$ 下构成一个阿贝尔群。

我们可以给椭圆曲线上的点定义一个数乘运算，就如同我们定义整数乘法一样：kP 即为 k 个 P 之和。

整数分解: Lenstra elliptic-curve factorization

那么椭圆曲线群是怎么和整数分解拉上关系的呢？这就要说到一个椭圆曲线的重要性质了：对于一个模素数 p 域下的椭圆曲线群来说，不妨设群内的点数为 M，那么总有：

$$p + 1 - 2\sqrt{p} \leq M \leq p + 1 + 2\sqrt{p}$$

整数分解: Lenstra elliptic-curve factorization

那么椭圆曲线群是怎么和整数分解拉上关系的呢？这就要说到一个椭圆曲线的重要性质了：对于一个模素数 p 域下的椭圆曲线群来说，不妨设群内的点数为 M，那么总有：

$$p + 1 - 2\sqrt{p} \leq M \leq p + 1 + 2\sqrt{p}$$

这个定理被称为 Hasse's Theorem。

整数分解: Lenstra elliptic-curve factorization

那么椭圆曲线群是怎么和整数分解拉上关系的呢？这就要说到一个椭圆曲线的重要性质了：对于一个模素数 p 域下的椭圆曲线群来说，不妨设群内的点数为 M，那么总有：

$$p + 1 - 2\sqrt{p} \leq M \leq p + 1 + 2\sqrt{p}$$

这个定理被称为 Hasse's Theorem。

在介绍利用椭圆曲线的分解方法之前，我们先介绍 Pollard's p-1 算法。

整数分解: Lenstra elliptic-curve factorization

考虑这样一个事实：不妨假设 p 是 n 的一个素因子，那么当 $\gcd(a, n) \neq 1$ 有：

$$a^{k\varphi(p)} \equiv 1 \pmod{p}$$

整数分解: Lenstra elliptic-curve factorization

考虑这样一个事实：不妨假设 p 是 n 的一个素因子，那么当 $\gcd(a, n) \neq 1$ 有：

$$a^{k\varphi(p)} \equiv 1 \pmod{p}$$

而此时若有

$$a^{k\varphi(p)} \not\equiv 1 \pmod{n}$$

那么只要计算 $\gcd(a^{k\varphi(p)} - 1, n)$ 就能得到 n 的一个非平凡因子。

整数分解: Lenstra elliptic-curve factorization

考虑这样一个事实：不妨假设 p 是 n 的一个素因子，那么当 $\gcd(a, n) \neq 1$ 有：

$$a^{k\varphi(p)} \equiv 1 \pmod{p}$$

而此时若有

$$a^{k\varphi(p)} \not\equiv 1 \pmod{n}$$

那么只要计算 $\gcd(a^{k\varphi(p)} - 1, n)$ 就能得到 n 的一个非平凡因子。

我们同样设定一个素数上限 $n^{\frac{1}{\alpha}}$，如果 $\varphi(p)$ 能分解成这些素数的乘积，而 $\varphi(n)$ 不能，我们就能够通过计算

$$\gcd\left(a^{\prod_p p^{\left\lfloor \frac{\ln \sqrt{n}}{\ln p} \right\rfloor}} - 1, n\right)$$

来得到一个 n 的非平凡因子。

整数分解: Lenstra elliptic-curve factorization

Lenstra 将以上算法从 n 的模域上搬到了模 n 的椭圆曲线群上：不妨假设模 n 的因子 p 域下的椭圆曲线群的元素个数为 L，那么对于该群中的任意元素 T，总有 $LT = \infty$——同样，在模 n 的模域下，这一性质同样存在 (我们通过加法得到 ∞ 意味着分母不存在逆元)。

整数分解: Lenstra elliptic-curve factorization

Lenstra 将以上算法从 n 的模域上搬到了模 n 的椭圆曲线群上：不妨假设模 n 的因子 p 域下的椭圆曲线群的元素个数为 L，那么对于该群中的任意元素 T，总有 $LT = \infty$——同样，在模 n 的模域下，这一性质同样存在 (我们通过加法得到 ∞ 意味着分母不存在逆元)。

而我们选取多条不同的椭圆曲线，只要有一条曲线满足 L 是 smooth 的，我们就可以成功分解。Lenstra 证明了，我们随机一条椭圆曲线，它的点数是在范围内非常随机的。

整数分解: Lenstra elliptic-curve factorization

Lenstra 将以上算法从 n 的模域上搬到了模 n 的椭圆曲线群上：不妨假设模 n 的因子 p 域下的椭圆曲线群的元素个数为 L，那么对于该群中的任意元素 T，总有 $LT = \infty$——同样，在模 n 的模域下，这一性质同样存在 (我们通过加法得到 ∞ 意味着分母不存在逆元)。

而我们选取多条不同的椭圆曲线，只要有一条曲线满足 L 是 smooth 的，我们就可以成功分解。Lenstra 证明了，我们随机一条椭圆曲线，它的点数是在范围内非常随机的。

这个做法的复杂度也是 $\mathcal{O}\left(\exp\left((1+o(1))\sqrt{\ln n \ln\ln n}\right)\right)$。

整数分解: Lenstra elliptic-curve factorization

Lenstra 将以上算法从 n 的模域上搬到了模 n 的椭圆曲线群上：不妨假设模 n 的因子 p 域下的椭圆曲线群的元素个数为 L，那么对于该群中的任意元素 T，总有 $LT = \infty$——同样，在模 n 的模域下，这一性质同样存在 (我们通过加法得到 ∞ 意味着分母不存在逆元)。

而我们选取多条不同的椭圆曲线，只要有一条曲线满足 L 是 smooth 的，我们就可以成功分解。Lenstra 证明了，我们随机一条椭圆曲线，它的点数是在范围内非常随机的。

这个做法的复杂度也是 $\mathcal{O}\left(\exp\left((1+o(1))\sqrt{\ln n \ln\ln n}\right)\right)$。

此外，我们还可以利用椭圆曲线来证明一个数是素数。

参考文献

D. H. Lehmer, “On the exact number of primes less than a given limit”, Illinois Journal of Mathematics, vol. 3, pp. 381–388, 1959.

J. C. Lagarias, V. S. Miller, and A. M. Odlyzko, “Computing $\pi(x)$: The Meissel-Lehmer method”, Mathematics of Computation, vol. 44, no. 170, pp. 537–560, Apr. 1985.

J. C. Lagarias and A. M. Odlyzko, “Computing $\pi(x)$: an analytic method ”, Journal of Algorithms, vol. 8, pp. 173–191, 1987.

Deléglise, Marc, and Joël Rivat. “Computing $\pi(x)$: the Meissel, Lehmer, Lagarias, Miller, Odlyzko method.” Mathematics of Computation of the American Mathematical Society 65.213 (1996): 235-245. Oliveira, Tomás. “Computing $\pi(x)$: the combinatorial method.” Electrónica e Telecomunicações 4.6 (2006): 759-768.

Kim Walisch, “https://github.com/kimwalisch/primecount ”

Min-25, “https://min-25.hatenablog.com/ ”

参考文献

任之洲, 《积性函数求和的几种方法》, 国家集训队 2016 论文集

朱震霆, 《一些特殊的数论函数求和问题》, 国家集训队 2018 论文集

Tornaría, Gonzalo. “Square roots modulo p.” Latin American Symposium on Theoretical Informatics. Springer, Berlin, Heidelberg, 2002.

Miskcoo’s Space, “http://blog.miskcoo.com/2014/08/quadratic-residue ”

Wikipedia, “https://en.wikipedia.org/wiki/Index_calculus_algorithmv ”

Wikipedia, “https://en.wikipedia.org/wiki/Elliptic_curve ”

Wikipedia, “https://en.wikipedia.org/wiki/Lenstra_elliptic-curve_factorization ”

Wiener ES, “https://www.zhihu.com/question/308322307 ”

参考文献

Silverman, Robert D. “The multiple polynomial quadratic sieve.” Mathematics of Computation 48.177 (1987): 329-339.

Wagstaff, Samuel S. The joy of factoring. Vol. 68. American Mathematical Soc., 2013.

zball, “https://loj.ac/article/863 ”

Fighting!

祝大家冬令营取得好成绩！