算法竞赛中的数论

1. 基础知识、简单素数筛法、与解线性 Diophantus 方程

陈亮舟
PinkRabbit

福建师范大学附属中学
The Affiliated High School of Fujian Normal University

2022 年 7 月 31 日
初版于 2022 年 1 月 26 日

关于

本系列课件的 PDF 版本和 LaTeX 源代码均托管于"算法竞赛中的数论 – 系列课件" – GitHub/GitPinkRabbit，后续更新将在此上传。

目录

1 基础知识

当前进度

整除性

定义 (整除、因数与倍数)

对于整数 a, b，且 $a \neq 0$，如果 $\frac{b}{a}$ 也是整数，则称 $\boldsymbol{b}$ **被** $\boldsymbol{a}$ **整除**，或称 $\boldsymbol{a}$ **整除** $\boldsymbol{b}$，记作 $\boldsymbol{a \mid b}$，否则称 a 不整除 b，记作 $\boldsymbol{a \nmid b}$。

若 a 整除 b，也称 b 是 a 的**倍数**，a 是 b 的**因数（约数）**。

整除性

定义 (整除、因数与倍数)

对于整数 a, b，且 $a \neq 0$，如果 $\frac{b}{a}$ 也是整数，则称 ***b* 被 *a* 整除**，或称 ***a* 整除 *b***，记作 $\boldsymbol{a \mid b}$，否则称 a 不整除 b，记作 $\boldsymbol{a \nmid b}$。
若 a 整除 b，也称 b 是 a 的**倍数**，a 是 b 的**因数（约数）**。

同时，我们约定 0 整除 0，但不整除其他任何整数。
即 $0 \mid 0$，但是对于所有 $n \neq 0$ 均有 $0 \nmid n$。

整除性 – 例子

例 (整除性)

整除性 – 例子

例 (整除性)

- 2 整除 6，因为 $\frac{6}{2} = 3$ 是整数。
- -6 整除 6，因为 $\frac{6}{-6} = -1$ 是整数。
- 5 整除 0，因为 $\frac{0}{5} = 0$ 是整数。
- -4 不整除 -6，因为 $\frac{-6}{-4} = 1.5$ 不是整数。

整除性 – 例子

例 (整除性)

- 2 整除 6，因为 $\frac{6}{2}=3$ 是整数。
- -6 整除 6，因为 $\frac{6}{-6}=-1$ 是整数。
- 5 整除 0，因为 $\frac{0}{5}=0$ 是整数。
- -4 不整除 -6，因为 $\frac{-6}{-4}=1.5$ 不是整数。
- 0 整除 0，因为我们约定如此。
- 0 不整除 -7，因为我们约定如此。

整除性 – 例子

例 (因数与倍数)

整除性 – 例子

例 (因数与倍数)

- 所以 6 是 2 的倍数，2 是 6 的因数。
- 6 是 -6 的倍数，-6 是 6 的因数。
- 0 是 5 的倍数，5 是 0 的因数。
- -6 不是 -4 的倍数，-4 不是 -6 的因数。

整除性 – 例子

例 (因数与倍数)

- 所以 6 是 2 的倍数，2 是 6 的因数。
- 6 是 -6 的倍数，-6 是 6 的因数。
- 0 是 5 的倍数，5 是 0 的因数。
- -6 不是 -4 的倍数，-4 不是 -6 的因数。
- 0 是 0 的倍数，0 是 0 的因数。
- -7 不是 0 的倍数，0 不是 -7 的因数。

整除性 – 基本性质

关于整除的一些基本性质：

整除性 – 基本性质

关于整除的一些基本性质：

- 对于任意整数 a，均有 $a \mid a$（自反性）以及 $a \mid (-a)$。
- 如果 $a \mid b$ 且 $b \mid c$，那么 $a \mid c$（传递性）。

整除性 – 基本性质

关于整除的一些基本性质：

- 对于任意整数 a，均有 $a \mid a$（自反性）以及 $a \mid (-a)$。
- 如果 $a \mid b$ 且 $b \mid c$，那么 $a \mid c$（传递性）。
- 如果 $a \mid b$ 且 $b \neq 0$，那么 $|a| \leq |b|$。
- 如果 $a \mid b$ 且 $b \mid a$，那么 $a = \pm b$。

整除性 – 基本性质

关于整除的一些基本性质：

- 对于任意整数 a，均有 $a \mid a$（自反性）以及 $a \mid (-a)$。
- 如果 $a \mid b$ 且 $b \mid c$，那么 $a \mid c$（传递性）。
- 如果 $a \mid b$ 且 $b \neq 0$，那么 $|a| \leq |b|$。
- 如果 $a \mid b$ 且 $b \mid a$，那么 $a = \pm b$。
- 如果 $a \mid b$ 且 k 为整数，那么 $a \mid (k \cdot b)$ 且 $(k \cdot a) \mid (k \cdot b)$。
- 如果 $d \mid a$ 且 $d \mid b$，那么 $d \mid (a \pm b)$。

整除性 – 基本性质

关于整除的一些基本性质：

- 对于任意整数 a，均有 $a \mid a$（自反性）以及 $a \mid (-a)$。
- 如果 $a \mid b$ 且 $b \mid c$，那么 $a \mid c$（传递性）。
- 如果 $a \mid b$ 且 $b \neq 0$，那么 $|a| \leq |b|$。
- 如果 $a \mid b$ 且 $b \mid a$，那么 $a = \pm b$。
- 如果 $a \mid b$ 且 k 为整数，那么 $a \mid (k \cdot b)$ 且 $(k \cdot a) \mid (k \cdot b)$。
- 如果 $d \mid a$ 且 $d \mid b$，那么 $d \mid (a \pm b)$。
- 一般地，如果有 n 个整数 $a_1, a_2, \ldots, a_n$，对于 $i = 1 \sim n$ 均有 $d \mid a_i$，且另给 n 个整数 $k_1, k_2, \ldots, k_n$，则有
$$d \mid (k_1 a_1 + k_2 a_2 + \cdots + k_n a_n)。$$

整除性 – 基本性质

关于整除的一些基本性质：

- 对于任意整数 a，均有 $a \mid a$（自反性）以及 $a \mid (-a)$。
- 如果 $a \mid b$ 且 $b \mid c$，那么 $a \mid c$（传递性）。
- 如果 $a \mid b$ 且 $b \neq 0$，那么 $|a| \leq |b|$。
- 如果 $a \mid b$ 且 $b \mid a$，那么 $a = \pm b$。
- 如果 $a \mid b$ 且 k 为整数，那么 $a \mid (k \cdot b)$ 且 $(k \cdot a) \mid (k \cdot b)$。
- 如果 $d \mid a$ 且 $d \mid b$，那么 $d \mid (a \pm b)$。
- 一般地，如果有 n 个整数 $a_1, a_2, \ldots, a_n$，对于 $i = 1 \sim n$ 均有 $d \mid a_i$，且另给 n 个整数 $k_1, k_2, \ldots, k_n$，则有
$$d \mid (k_1 a_1 + k_2 a_2 + \cdots + k_n a_n)。$$
其中 $k_1 a_1 + k_2 a_2 + \cdots + k_n a_n$ 被称作 $a_{1\sim n}$ 的**整系数线性组合**。

整除性 – 简单规律

关于整除的一些简单规律：

整除性 – 简单规律

关于整除的一些简单规律：

- 1 和 -1 整除所有整数。
  1 和 -1 是所有整数的因数，所有整数是 1 和 -1 的倍数。

整除性 – 简单规律

关于整除的一些简单规律：

- 1 和 -1 整除所有整数。
  1 和 -1 是所有整数的因数，所有整数是 1 和 -1 的倍数。
- 所有整数整除 0。
  0 是所有整数的倍数，所有整数是 0 的因数。

整除性 – 简单规律

关于整除的一些简单规律：

- 1 和 -1 整除所有整数。
  1 和 -1 是所有整数的因数，所有整数是 1 和 -1 的倍数。
- 所有整数整除 0。
  0 是所有整数的倍数，所有整数是 0 的因数。
- 对于正整数 a，a 是 a 的最大因数，也是 a 的最小正倍数。
  并且，$-a$ 是 a 的最小因数，也是 a 的最大负倍数。

整除性 – 简单规律

关于整除的一些简单规律：

- 1 和 -1 整除所有整数。
  1 和 -1 是所有整数的因数，所有整数是 1 和 -1 的倍数。
- 所有整数整除 0。
  0 是所有整数的倍数，所有整数是 0 的因数。
- 对于正整数 a，a 是 a 的最大因数，也是 a 的最小正倍数。
  并且，$-a$ 是 a 的最小因数，也是 a 的最大负倍数。

一般地，对于任意整数 a，一定有 $1, -1, a, -a$ 是 a 的因数，
也一定有 $0, a, -a$ 是 a 的倍数。

对于非 0 整数 a，称 $1, -1, a, -a$ 为 a 的**平凡因数**。

有余数的除法

定义 (有余数的除法)

对于整数 a 和正整数 m，存在且仅存在一对整数 $\langle q, r\rangle$ 满足

$$a = q \cdot m + r \text{ 且 } 0 \le r < m,$$

称 a 除以 m 的 **商** 为 q，
a 除以 m 的**余数**为 r。
记作 $a \div m = q \cdots\cdots r$。

有余数的除法

定义 (有余数的除法)

对于整数 a 和正整数 m，存在且仅存在一对整数 $\langle q, r\rangle$ 满足

$$a = q \cdot m + r \text{ 且 } 0 \le r < m,$$

称 a 除以 m 的 **商** 为 q，
a 除以 m 的**余数**为 r。
记作 $a \div m = q \cdots\cdots r$。

在计算上，很简单地有 $q = \left\lfloor \frac{a}{m} \right\rfloor$ 和 $r = a - q \cdot m = a - \left\lfloor \frac{a}{m} \right\rfloor \cdot m$。
这里 $\lfloor x \rfloor$ 指向下取整，例如 $\lfloor 3.8 \rfloor = 3$、$\lfloor -4.2 \rfloor = -5$、$\lfloor 7 \rfloor = 7$、$\lfloor -9 \rfloor = -9$。

有余数的除法 – 例子

例 (有余数的除法)

- 8 除以 3 的商为 2，余数为 2。
  因为 $2 \times 3 + 2 = 8$，且 $0 \le 2 < 3$。
- -9 除以 5 的商为 -2，余数为 1。
  因为 $(-2) \times 5 + 1 = -9$，且 $0 \le 1 < 5$。
- -12 除以 4 的商为 -3，余数为 0。
  因为 $(-3) \times 4 + 0 = -12$，且 $0 \le 0 < 4$。
- 10^6 除以 7 的商为 142857，余数为 1。
  因为 $142857 \times 7 + 1 = 10^6$，且 $0 \le 1 < 7$。

有余数的除法 – 例子

例 (有余数的除法)

- 8 除以 3 的商为 2，余数为 2。
  因为 $2 \times 3 + 2 = 8$，且 $0 \le 2 < 3$。
- -9 除以 5 的商为 -2，余数为 1。
  因为 $(-2) \times 5 + 1 = -9$，且 $0 \le 1 < 5$。
- -12 除以 4 的商为 -3，余数为 0。
  因为 $(-3) \times 4 + 0 = -12$，且 $0 \le 0 < 4$。
- 10^6 除以 7 的商为 142857，余数为 1。
  因为 $142857 \times 7 + 1 = 10^6$，且 $0 \le 1 < 7$。

注意到，如果 a 除以 m 的余数为 0，则 $m \mid a$，否则 $m \nmid a$。

C++ 中的除法和求余运算符

在 C++ 中，提供了除法运算符“/”和求余运算符“%”以支持 C++ 程序进行相关计算。它们的形式为：

- “左操作数 / 右操作数”，
- “左操作数 % 右操作数”。

这里我们针对两操作数均拥有整数类型或无作用域枚举类型的情况进行说明。假设两操作数已进行过整型提升和整型转换，并产生为 `int` 或 `long long` 及其无符号（unsigned）版本之一的公共类型。

C++ 中的除法和求余运算符

从 C++11 标准开始，规定了除法运算符的舍入方向：运算结果为第一操作数除以第二操作数的数值结果向零舍入得到的整数。

这意味着，如果数值结果 > 0，则运算结果 $\le$ 数值结果，
如果数值结果 < 0，则运算结果 $\ge$ 数值结果，
如果数值结果 $= 0$，则运算结果 $=$ 数值结果 $= 0$。

C++ 中的除法和求余运算符

从 C++11 标准开始，规定了除法运算符的舍入方向：运算结果为第一操作数除以第二操作数的数值结果向零舍入得到的整数。

这意味着，如果数值结果 > 0，则运算结果 $\leq$ 数值结果，
如果数值结果 < 0，则运算结果 $\geq$ 数值结果，
如果数值结果 $= 0$，则运算结果 $=$ 数值结果 $= 0$。

注意，C++ 中除法和求余运算符的第二操作数均可以为负数，这与前文中“有余数的除法”不同。

C++ 中的除法和求余运算符

从 C++11 标准开始，规定了除法运算符的舍入方向：运算结果为第一操作数除以第二操作数的数值结果向零舍入得到的整数。

这意味着，如果数值结果 > 0，则运算结果 $\leq$ 数值结果，
如果数值结果 < 0，则运算结果 $\geq$ 数值结果，
如果数值结果 $= 0$，则运算结果 $=$ 数值结果 $= 0$。

注意，C++ 中除法和求余运算符的第二操作数均可以为负数，这与前文中“有余数的除法”不同。

特别地，如果第二操作数为 0，则行为未定义（UB）。典型的编译器实现可能导致运行时错误（RE）。
若运算结果不能以结果类型表示，则行为未定义。

C++ 中的除法和求余运算符 – 除法运算符例子

例 (除法运算符)

C++ 中的除法和求余运算符 – 除法运算符例子

例 (除法运算符)

- 5 / 3 的结果为 1。
- 5 / −3 的结果为 −1。
- −5 / 3 的结果为 −1。
- −5 / −3 的结果为 1。

C++ 中的除法和求余运算符 – 除法运算符例子

例 (除法运算符)

- `5 /  3` 的结果为 `1`。
- `5 / -3` 的结果为 `-1`。
- `-5 /  3` 的结果为 `-1`。
- `-5 / -3` 的结果为 `1`。
- `-7 /  0` 是未定义行为。
- `0 /  0` 是未定义行为。

C++ 中的除法和求余运算符 – 除法运算符例子

例 (除法运算符)

- `5 /  3` 的结果为 `1`。
- `5 / -3` 的结果为 `-1`。
- `-5 /  3` 的结果为 `-1`。
- `-5 / -3` 的结果为 `1`。
- `-7 /  0` 是未定义行为。
- `0 /  0` 是未定义行为。
- `(int)(-2147483648LL) / -1` 是未定义行为（补码系统上）。

C++ 中的除法和求余运算符 – 求余运算符

对于求余运算符，C++ 保证了：若 $a \mathbin{\%} m = r$ 以及 $a \mathbin{/} m = q$，则一定有 $a = q * m + r$。（前提是 $a \mathbin{/} m$ 时不会触发未定义行为）

C++ 中的除法和求余运算符 – 求余运算符

对于求余运算符，C++ 保证了：若 $a \mathbin{\%} m = r$ 以及 $a \mathbin{/} m = q$，则一定有 $a = q * m + r$。（前提是 $a \mathbin{/} m$ 时不会触发未定义行为）

所以，对于非负数 a 和正数 m，C++ 产生的商和余数与前文“有余数的除法”中定义的相同。

C++ 中的除法和求余运算符 – 求余运算符例子

例 (求余运算符)

C++ 中的除法和求余运算符 – 求余运算符例子

例 (求余运算符)

- 5 % 3 的结果为 2。
- 5 % −3 的结果为 2。
- −5 % 3 的结果为 −2。
- −5 % −3 的结果为 −2。

C++ 中的除法和求余运算符 – 求余运算符例子

例 (求余运算符)

- `5 % 3` 的结果为 `2`。
- `5 % -3` 的结果为 `2`。
- `-5 % 3` 的结果为 `-2`。
- `-5 % -3` 的结果为 `-2`。
- `-7 % 0` 是未定义行为。
- `0 % 0` 是未定义行为。

C++ 中的除法和求余运算符 – 求余运算符例子

例 (求余运算符)

- `5 %  3` 的结果为 `2`。
- `5 % -3` 的结果为 `2`。
- `-5 %  3` 的结果为 `-2`。
- `-5 % -3` 的结果为 `-2`。
- `-7 %  0` 是未定义行为。
- `0 %  0` 是未定义行为。
- `(int)(-2147483648LL) % -1` 是未定义行为（补码系统上）。

C++ 中的除法和求余运算符 – 正负性提示

假设 a / $m = q$ 和 a % $m = r$。
则有 q 的正负性与 $a \cdot m$ 的正负性相同，
r 的正负性与 a 的正负性相同。

C++ 中的除法和求余运算符 – 正负性提示

假设 $a \mathbin{/} m = q$ 和 $a \mathbin{\%} m = r$。
则有 q 的正负性与 $a \cdot m$ 的正负性相同，
r 的正负性与 a 的正负性相同。

在操作数可能为负数或 0 时，需要特别注意。

素数与合数

定义 (素数)

恰好有 2 个正因数的正整数称为**素数（质数，prime number）**。
即，唯二的两个正因数为 1 和它本身。

素数与合数

定义 (素数)

恰好有 2 个正因数的正整数称为**素数（质数，prime number）**。即，唯二的两个正因数为 1 和它本身。

例 (素数)

- 7 是素数，因为 7 恰好有 2 个正因数 $1, 7$。
- 51 不是素数，因为 51 有 4 个正因数 $1, 3, 17, 51$。
- 1 不是素数，因为 1 只有 1 个正因数 1。
- -7 不是素数，因为 -7 不是正整数。

素数与合数

定义 (素数)

恰好有 2 个正因数的正整数称为**素数（质数，prime number）**。
即，唯一的两个正因数为 1 和它本身。

例 (素数)

- 7 是素数，因为 7 恰好有 2 个正因数 $1, 7$。
- 51 不是素数，因为 51 有 4 个正因数 $1, 3, 17, 51$。
- 1 不是素数，因为 1 只有 1 个正因数 1。
- -7 不是素数，因为 -7 不是正整数。

若无特殊说明，下文中的字母 p 均表示素数，$\mathbb{P}$ 为素数集。

素数与合数

这里列举一下前几个素数：

$$\mathbb{P}=\{2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,\ldots\}。$$

素数与合数

这里列举一下前几个素数：

$$\mathbb{P} = \{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, \ldots\}。$$

定义 (合数)

有多于 2 个正因数的正整数称为**合数**。
即，除了 1 和它本身还有别的正因数。
注意 1 既不是素数也不是合数。

素数与合数

这里列举一下前几个素数：

$$\mathbb{P} = \{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, \ldots\}。$$

定义 (合数)

有多于 2 个正因数的正整数称为**合数**。
即，除了 1 和它本身还有别的正因数。
注意 1 既不是素数也不是合数。

这里列举一下前几个合数：

$$\mathbb{Z}_{\geq 2} \setminus \mathbb{P} = \{4, 6, 8, 9, 10, 12, 14, 15, 16, 18, 20, 21, 22, 24, 25, 26, \ldots\}。$$

当前进度

同余

定义 (同余和模)

对于正整数 m 和整数 a, b，如果 a, b 的差被 m 整除，即 $m \mid (a-b)$，则称 **a 同余于 b 模 m**。
换句话说，a 除以 m 的余数等于 b 除以 m 的余数。
记作 $a \equiv b \pmod{m}$，并称 m 为**模数**。
如果 $m \nmid (a-b)$，则称 a 不同余于 b 模 m，记作 $a \not\equiv b \pmod{m}$。

同余

定义 (同余和模)

对于正整数 m 和整数 a, b，如果 a, b 的差被 m 整除，即 $m \mid (a-b)$，则称 **a 同余于 b 模 m**。
换句话说，a 除以 m 的余数等于 b 除以 m 的余数。
记作 $\boldsymbol{a \equiv b \pmod{m}}$，并称 m 为**模数**。
如果 $m \nmid (a-b)$，则称 a 不同余于 b 模 m，记作 $a \not\equiv b \pmod{m}$。

固定模 m 下的同余关系是一种等价关系，即如果 $a \equiv b$ 且 $b \equiv c$，则 $a \equiv c$。还可以记作 $a \equiv b \equiv c \pmod{m}$。

同余

定义 (同余和模)

对于正整数 m 和整数 a, b，如果 a, b 的差被 m 整除，即 $m \mid (a-b)$，则称 **a 同余于 b 模 m**。
换句话说，a 除以 m 的余数等于 b 除以 m 的余数。
记作 $\boldsymbol{a \equiv b \pmod{m}}$，并称 m 为**模数**。
如果 $m \nmid (a-b)$，则称 a 不同余于 b 模 m，记作 $a \not\equiv b \pmod{m}$。

固定模 m 下的同余关系是一种等价关系，即如果 $a \equiv b$ 且 $b \equiv c$，则 $a \equiv c$。还可以记作 $a \equiv b \equiv c \pmod{m}$。

若 a 除以 m 的余数为 r，也可以称 **a 对 m 的模为 r**，或称 **a 模 m 等于 r**。记作 $\boldsymbol{a \bmod m = r}$。

同余 – 例子

例 (同余)

同余 – 例子

例 (同余)

- $-2 \equiv 4 \pmod 3$ 并且 $6 \not\equiv 7 \pmod 3$。

同余 – 例子

例 (同余)

- $-2 \equiv 4 \pmod 3$ 并且 $6 \not\equiv 7 \pmod 3$。
- 如果 $\mathcal{A} \equiv \mathcal{B} \pmod m$，且 $\mathcal{C} \equiv \mathcal{D} \pmod m$，则
  $$(\mathcal{A}) \pm (\mathcal{C}) \equiv (\mathcal{B}) \pm (\mathcal{D}) \pmod m，以及$$
  $$(\mathcal{A}) \cdot (\mathcal{C}) \equiv (\mathcal{B}) \cdot (\mathcal{D}) \pmod m。$$
  其中 $\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D}$ 为任意取值为整数的数学表达式。

同余 – 例子

例 (同余)

- $-2 \equiv 4 \pmod 3$ 并且 $6 \not\equiv 7 \pmod 3$。
- 如果 $\mathcal{A} \equiv \mathcal{B} \pmod m$，且 $\mathcal{C} \equiv \mathcal{D} \pmod m$，则
  $$(\mathcal{A}) \pm (\mathcal{C}) \equiv (\mathcal{B}) \pm (\mathcal{D}) \pmod m$$，以及
  $$(\mathcal{A}) \cdot (\mathcal{C}) \equiv (\mathcal{B}) \cdot (\mathcal{D}) \pmod m$$。

  其中 $\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D}$ 为任意取值为整数的数学表达式。
- 如果 $\mathcal{A} \equiv \mathcal{B} \pmod m$，则对于任意正整数 k，有
  $$k \cdot (\mathcal{A}) \equiv k \cdot (\mathcal{B}) \pmod{k \cdot m}$$。

同余 – 例子

例 (同余)

- $-2 \equiv 4 \pmod 3$ 并且 $6 \not\equiv 7 \pmod 3$。
- 如果 $\mathcal{A} \equiv \mathcal{B} \pmod m$，且 $\mathcal{C} \equiv \mathcal{D} \pmod m$，则
  $$(\mathcal{A}) \pm (\mathcal{C}) \equiv (\mathcal{B}) \pm (\mathcal{D}) \pmod m$$，以及
  $$(\mathcal{A}) \cdot (\mathcal{C}) \equiv (\mathcal{B}) \cdot (\mathcal{D}) \pmod m$$。
  其中 $\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D}$ 为任意取值为整数的数学表达式。
- 如果 $\mathcal{A} \equiv \mathcal{B} \pmod m$，则对于任意正整数 k，有
  $$k \cdot (\mathcal{A}) \equiv k \cdot (\mathcal{B}) \pmod{k \cdot m}$$。
- 如果 $\mathcal{A} \equiv \mathcal{B} \pmod m$，且 $k \mid \mathcal{A}$、$k \mid \mathcal{B}$、$k \mid m$，则
  $$\frac{\mathcal{A}}{k} \equiv \frac{\mathcal{B}}{k} \pmod{\frac{m}{k}}$$。

最大公因数与最小公倍数 – 公因数

定义 (公因数与最大公因数)

对于整数 a, b，若存在整数 d 使得 $d \mid a$ 且 $d \mid b$，则称 d 是 a, b 的一个**公因数**。a, b 的公因数中的最大值称为 a, b 的**最大公因数**（greatest common divisor），记作 $\gcd(a, b)$。
特别地，约定 0 与 0 的最大公因数为 $\gcd(0, 0) = 0$。

最大公因数与最小公倍数 – 公因数

定义 (公因数与最大公因数)

对于整数 a, b，若存在整数 d 使得 $d \mid a$ 且 $d \mid b$，则称 d 是 a, b 的一个**公因数**。a, b 的公因数中的最大值称为 a, b 的**最大公因数 (greatest common divisor)**，记作 $\gcd(a, b)$。
特别地，约定 0 与 0 的最大公因数为 $\gcd(0, 0) = 0$。

对于 n 个整数 $a_1, a_2, \ldots, a_n$，若存在整数 d 使得对于 $i = 1 \sim n$ 均有 $d \mid a_i$，则称 d 是 $a_1, a_2, \ldots, a_n$ 的一个公因数。
$a_1, a_2, \ldots, a_n$ 的公因数中的最大值称为 $a_1, a_2, \ldots, a_n$ 的最大公因数，记作 $\gcd(a_1, a_2, \ldots, a_n)$。
特别地，若 a_i 全部为 0，约定最大公因数为 $\gcd(0, 0, \ldots, 0) = 0$。

最大公因数与最小公倍数 – 公因数例子

例 (公因数与最大公因数)

最大公因数与最小公倍数 – 公因数例子

例 (公因数与最大公因数)

- 60 与 96 的公因数有 $\{\pm1, \pm2, \pm3, \pm4, \pm6, \pm12\}$。
- -60 与 -96 的最大公因数为 $\gcd(-60, -96) = 12$。

最大公因数与最小公倍数 – 公因数例子

例 (公因数与最大公因数)

- 60 与 96 的公因数有 $\{\pm 1, \pm 2, \pm 3, \pm 4, \pm 6, \pm 12\}$。
- -60 与 -96 的最大公因数为 $\gcd(-60, -96) = 12$。
- 0 与 0 的最大公因数为 $\gcd(0, 0) = 0$。
- $6, 10, 15$ 的最大公因数为 $\gcd(6, 10, 15) = 1$。

最大公因数与最小公倍数 – 公因数例子

例 (公因数与最大公因数)

- 60 与 96 的公因数有 $\{\pm 1, \pm 2, \pm 3, \pm 4, \pm 6, \pm 12\}$。
- -60 与 -96 的最大公因数为 $\gcd(-60, -96) = 12$。
- 0 与 0 的最大公因数为 $\gcd(0, 0) = 0$。
- $6, 10, 15$ 的最大公因数为 $\gcd(6, 10, 15) = 1$。

定义 (互素)

如果 $\gcd(a, b) = 1$，那么称 **a 与 b 互素**。记作 **$a \perp b$**。
如果 $\gcd(a_1, a_2, \ldots, a_n) = 1$，那么称 $a_1, a_2, \ldots, a_n$ 互素，但未必两两互素。

最大公因数与最小公倍数 – 公倍数

定义 (公倍数与最小公倍数)

对于整数 a, b，若存在整数 m 使得 $a \mid m$ 且 $b \mid m$，则称 m 是 a, b 的一个**公倍数**。a, b 的公倍数中的最小正值称为 a, b 的**最小公倍数（least common multiple）**，记作 $\operatorname{lcm}(a, b)$。
特别地，约定任何整数 a 与 0 的最小公倍数为 0。

最大公因数与最小公倍数 – 公倍数

定义 (公倍数与最小公倍数)

对于整数 a, b，若存在整数 m 使得 $a \mid m$ 且 $b \mid m$，则称 m 是 a, b 的一个**公倍数**。a, b 的公倍数中的最小正值称为 a, b 的**最小公倍数 (least common multiple)**，记作 $\operatorname{lcm}(a, b)$。
特别地，约定任何整数 a 与 0 的最小公倍数为 0。

对于 n 个整数 $a_1, a_2, \ldots, a_n$，若存在整数 m 使得对于 $i = 1 \sim n$ 均有 $a_i \mid m$，则称 m 是 $a_1, a_2, \ldots, a_n$ 的一个公倍数。
$a_1, a_2, \ldots, a_n$ 的公倍数中的最小正值称为 $a_1, a_2, \ldots, a_n$ 的最小公倍数，记作 $\operatorname{lcm}(a_1, a_2, \ldots, a_n)$。
特别地，若 a_i 中至少有一个为 0，约定最小公倍数为 0。

最大公因数与最小公倍数 – 公倍数例子

例 (公倍数与最小公倍数)

最大公因数与最小公倍数 – 公倍数例子

例 (公倍数与最小公倍数)

- 60 与 96 的公倍数有 $\{\ldots,-960,-480,0,480,960,\ldots\}$。
- -60 与 -96 的最小公倍数为 $\operatorname{lcm}(-60,-96)=480$。

最大公因数与最小公倍数 – 公倍数例子

例 (公倍数与最小公倍数)

- 60 与 96 的公倍数有 $\{\ldots, -960, -480, 0, 480, 960, \ldots\}$。
- -60 与 -96 的最小公倍数为 $\operatorname{lcm}(-60, -96) = 480$。
- -60 与 0 的最小公倍数为 $\operatorname{lcm}(-60, 0) = 0$。
- $6, 10, 15$ 的最小公倍数为 $\operatorname{lcm}(6, 10, 15) = 30$。

最大公因数与最小公倍数 – 公倍数例子

例 (公倍数与最小公倍数)

- 60 与 96 的公倍数有 $\{\dots,-960,-480,0,480,960,\dots\}$。
- -60 与 -96 的最小公倍数为 $\operatorname{lcm}(-60,-96)=480$。
- -60 与 0 的最小公倍数为 $\operatorname{lcm}(-60,0)=0$。
- $6,10,15$ 的最小公倍数为 $\operatorname{lcm}(6,10,15)=30$。
- 如果 $a\perp b$，那么 $\operatorname{lcm}(a,b)=|a|\cdot|b|$。
  否则，也有 $\operatorname{lcm}(a,b)=\frac{|a|\cdot|b|}{\gcd(a,b)}$（$a,b$ 不能同时为 0）。

当前进度

算术基本定理

定理 (算术基本定理（fundamental theorem of arithmetic）)

对于任意正整数 n，n 可以表示成有限个素数的乘积（允许重复）。即 $n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}$，其中 $p_1 < p_2 < \cdots < p_k$ 为递增素数，$\alpha_1, \alpha_2, \ldots, \alpha_k$ 为正整数。并且这种表示方法是唯一的，称为 n 的**标准分解式**。
当 $n = 1$ 时，可视作 $k = 0$，即 1 可以表示成 0 个素数的乘积（我们约定 0 个数的乘积为 1）。

算术基本定理

定理 (算术基本定理（fundamental theorem of arithmetic）)

对于任意正整数 n，n 可以表示成有限个素数的乘积（允许重复）。即 $n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}$，其中 $p_1 < p_2 < \cdots < p_k$ 为递增素数，$\alpha_1, \alpha_2, \ldots, \alpha_k$ 为正整数。并且这种表示方法是唯一的，称为 n 的**标准分解式**。

当 $n = 1$ 时，可视作 $k = 0$，即 1 可以表示成 0 个素数的乘积（我们约定 0 个数的乘积为 1）。

例 (标准分解式)

- $10725 = 3^1 \cdot 5^2 \cdot 11^1 \cdot 13^1$
- $998244352 = 2^{23} \cdot 7^1 \cdot 17^1$

算术基本定理 – 证明（存在性）

算术基本定理 – 存在性.

如果 $n = 1$ 或 n 是素数，则显然存在表示方法。
否则 n 是合数，存在非 1 或其本身的正因数 a，令 $b = \frac{n}{a}$，则 $a, b < n$，可以递归地求出 a, b 的表示方法：$a = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}$ 和 $b = q_1^{\beta_1} q_2^{\beta_2} \cdots q_l^{\beta_l}$，将两个表示方法进行归并即可得到 n 的表示方法。 $\square$

算术基本定理 – 证明（唯一性）

算术基本定理 – 唯一性.

假设存在正整数拥有多于一种表示方法，令其中最小的数为 n。
显然，满足条件的 n 必须为合数。

算术基本定理 – 证明（唯一性）

算术基本定理 – 唯一性.

假设存在正整数拥有多于一种表示方法，令其中最小的数为 n。
显然，满足条件的 n 必须为合数。
设 n 的两种表示方法为 $p_1^{\alpha_1}p_2^{\alpha_2}\cdots p_k^{\alpha_k}$ 和 $q_1^{\beta_1}q_2^{\beta_2}\cdots q_l^{\beta_l}$。

算术基本定理 – 证明（唯一性）

算术基本定理 – 唯一性.

假设存在正整数拥有多于一种表示方法，令其中最小的数为 n。
显然，满足条件的 n 必须为合数。
设 n 的两种表示方法为 $p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}$ 和 $q_1^{\beta_1} q_2^{\beta_2} \cdots q_l^{\beta_l}$。

令 $P = p_1^{\alpha_1 - 1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}$ 以及 $Q = q_1^{\beta_1 - 1} q_2^{\beta_2} \cdots q_l^{\beta_l}$。
那么 $p_1 P = q_1 Q = n$，且 $2 \le P, Q < n$。

算术基本定理 – 证明（唯一性）

算术基本定理 – 唯一性.

假设存在正整数拥有多于一种表示方法，令其中最小的数为 n。
显然，满足条件的 n 必须为合数。
设 n 的两种表示方法为 $p_1^{\alpha_1}p_2^{\alpha_2}\cdots p_k^{\alpha_k}$ 和 $q_1^{\beta_1}q_2^{\beta_2}\cdots q_l^{\beta_l}$。

令 $P = p_1^{\alpha_1-1}p_2^{\alpha_2}\cdots p_k^{\alpha_k}$ 以及 $Q = q_1^{\beta_1-1}q_2^{\beta_2}\cdots q_l^{\beta_l}$。
那么 $p_1P = q_1Q = n$，且 $2 \le P, Q < n$。

- 如果 $p_1 = q_1$，那么 $P = Q < n$ 拥有两种表示方法，与 n 的最小性矛盾。

算术基本定理 – 证明（唯一性）

算术基本定理 – 唯一性.

假设存在正整数拥有多于一种表示方法，令其中最小的数为 n。
显然，满足条件的 n 必须为合数。
设 n 的两种表示方法为 $p_1^{\alpha_1}p_2^{\alpha_2}\cdots p_k^{\alpha_k}$ 和 $q_1^{\beta_1}q_2^{\beta_2}\cdots q_l^{\beta_l}$。

令 $P = p_1^{\alpha_1-1}p_2^{\alpha_2}\cdots p_k^{\alpha_k}$ 以及 $Q = q_1^{\beta_1-1}q_2^{\beta_2}\cdots q_l^{\beta_l}$。
那么 $p_1P = q_1Q = n$，且 $2 \le P, Q < n$。

- 如果 $p_1 = q_1$，那么 $P = Q < n$ 拥有两种表示方法，与 n 的最小性矛盾。
- 否则 $p_1 \neq q_1$，不失一般性，设 $p_1 < q_1$，则 $Q < P < n$。

（续……）

算术基本定理 – 证明（唯一性）

算术基本定理 – 唯一性（续）.

- 否则 $p_1 \neq q_1$，不失一般性，设 $p_1 < q_1$，则 $Q < P < n$。

算术基本定理 – 证明（唯一性）

算术基本定理 – 唯一性（续）.

- 否则 $p_1 \neq q_1$，不失一般性，设 $p_1 < q_1$，则 $Q < P < n$。
  令 $s = p_1 \cdot (P - Q)$，则 $2 \le s < n$，然而同时 $s = (q_1 - p_1) \cdot Q$。

算术基本定理 – 证明（唯一性）

算术基本定理 – 唯一性（续）.

- 否则 $p_1 \neq q_1$，不失一般性，设 $p_1 < q_1$，则 $Q < P < n$。令 $s = p_1 \cdot (P - Q)$，则 $2 \leq s < n$，然而同时 $s = (q_1 - p_1) \cdot Q$。由 $s < n$ 可知 s 的表示方法唯一。注意到 $p_1 \cdot (P - Q)$ 中包含 p_1，这说明 $q_1 - p_1$ 或 Q 中必然包含 p_1。

算术基本定理 – 证明（唯一性）

算术基本定理 – 唯一性（续）.

- 否则 $p_1 \neq q_1$，不失一般性，设 $p_1 < q_1$，则 $Q < P < n$。
  令 $s = p_1 \cdot (P - Q)$，则 $2 \leq s < n$，然而同时 $s = (q_1 - p_1) \cdot Q$。
  由 $s < n$ 可知 s 的表示方法唯一。注意到 $p_1 \cdot (P - Q)$ 中包含 p_1，这说明 $q_1 - p_1$ 或 Q 中必然包含 p_1。
  然而 $p_1 \mid (q_1 - p_1) \iff p_1 \mid q_1$ 是不可能的，因为 q_1 是素数。

算术基本定理 – 证明（唯一性）

算术基本定理 – 唯一性（续）.

- 否则 $p_1 \neq q_1$，不失一般性，设 $p_1 < q_1$，则 $Q < P < n$。
  令 $s = p_1 \cdot (P - Q)$，则 $2 \leq s < n$，然而同时 $s = (q_1 - p_1) \cdot Q$。
  由 $s < n$ 可知 s 的表示方法唯一。注意到 $p_1 \cdot (P - Q)$ 中包含 p_1，这说明 $q_1 - p_1$ 或 Q 中必然包含 p_1。
  然而 $p_1 \mid (q_1 - p_1) \iff p_1 \mid q_1$ 是不可能的，因为 q_1 是素数。
  并且 $p_1 \mid Q$ 也是不可能的，因为由 $Q < n$ 可知 Q 的表示方法唯一，而所有 q_j 均大于 p_1（$1 \leq j \leq l$），故 $p_1 \nmid Q$。

算术基本定理 – 证明（唯一性）

算术基本定理 – 唯一性（续）.

- 否则 $p_1 \neq q_1$，不失一般性，设 $p_1 < q_1$，则 $Q < P < n$。
  令 $s = p_1 \cdot (P - Q)$，则 $2 \leq s < n$，然而同时 $s = (q_1 - p_1) \cdot Q$。
  由 $s < n$ 可知 s 的表示方法唯一。注意到 $p_1 \cdot (P - Q)$ 中包含 p_1，这说明 $q_1 - p_1$ 或 Q 中必然包含 p_1。
  然而 $p_1 \mid (q_1 - p_1) \iff p_1 \mid q_1$ 是不可能的，因为 q_1 是素数。
  并且 $p_1 \mid Q$ 也是不可能的，因为由 $Q < n$ 可知 Q 的表示方法唯一，而所有 q_j 均大于 p_1（$1 \leq j \leq l$），故 $p_1 \nmid Q$。

至此，所有可能性已被排除，故假设有误，不存在拥有多于一种表示方法的正整数。 $\square$

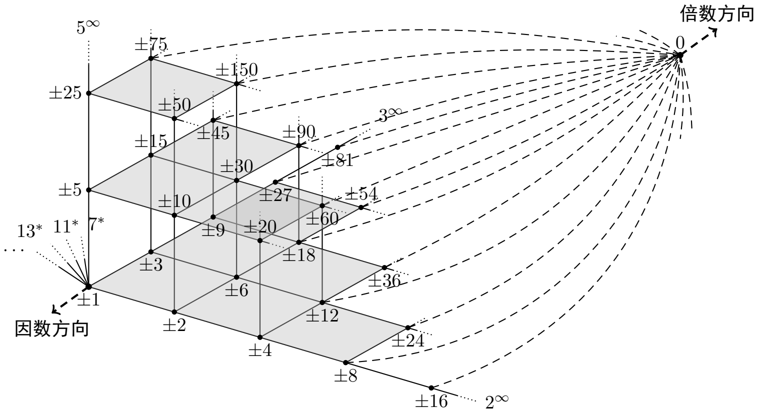
算术基本定理 – 更多例子

当指数为 1 时，可以省略指数。
对于 n 为负数的情况，若允许 -1 作为因子出现不超过 1 次，负数的标准分解显然也是唯一的。

- $1 = 1$
- $4 = 2^2$
- $6 = 2 \cdot 3$
- $8 = 2^3$
- $9 = 3^2$
- $10 = 2 \cdot 5$
- $12 = 2^2 \cdot 3$
- $15 = 3 \cdot 5$
- $18 = 2 \cdot 3^2$
- $20 = 2^2 \cdot 5$
- $21 = 3 \cdot 7$
- $24 = 2^3 \cdot 3$
- $-60 = (-1) \cdot 2^2 \cdot 3 \cdot 5$
- $-117047 = (-1) \cdot 7 \cdot 23 \cdot 727$

当前进度

整除结构 – 直接上图


倍数方向
0
5∞
±75
±150
±25
±50
±45
±90
3∞
±15
±30
±81
±5
±27
±54
±10
±60
13*
11*
7*
±9
±20
···
±18
±3
±36
±6
±1
±12
因数方向
±2
±24
±4
±8
±16
2∞

标准分解式与 p 进赋值序列

定义

对于正整数 n，设其标准分解式为 $n=p_1^{\alpha_1}p_2^{\alpha_2}\cdots p_k^{\alpha_k}$，对于给定素数 p，定义 n 的 **p 进阶数** $\nu_p(n)=\begin{cases}\alpha_i & \text{如果存在 } p_i=p, \\ 0 & \text{不存在 } p_i=p。\end{cases}$

即，如果 n 的 p 进阶数 $\nu_p(n)=\alpha$，则 $p^\alpha \mid n$ 但 $p^{\alpha+1}\nmid n$，此时也记作 $p^\alpha \parallel n$，如果 $p^\alpha \parallel n$ 不成立，记作 $p^\alpha \nparallel n$。

标准分解式与 p 进赋值序列

定义

对于正整数 n，设其标准分解式为 $n=p_1^{\alpha_1}p_2^{\alpha_2}\cdots p_k^{\alpha_k}$，对于给定素数 p，定义 n 的 **p 进阶数** $\nu_p(n)=\begin{cases}\alpha_i & \text{如果存在 } p_i=p\text{，}\\ 0 & \text{不存在 } p_i=p\text{。}\end{cases}$

即，如果 n 的 p 进阶数 $\nu_p(n)=\alpha$，则 $p^\alpha\mid n$ 但 $p^{\alpha+1}\nmid n$，此时也记作 $p^\alpha\parallel n$，如果 $p^\alpha\parallel n$ 不成立，记作 $p^\alpha\nparallel n$。

同时，定义 n 的 **p 进赋值序列**为 $\langle\nu_2(n),\nu_3(n),\nu_5(n),\ldots\rangle$。

标准分解式与 p 进赋值序列

定义

对于正整数 n，设其标准分解式为 $n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}$，对于给定素数 p，定义 n 的 **p 进阶数** $\nu_p(n) = \begin{cases} \alpha_i & \text{如果存在 } p_i = p\text{，} \\ 0 & \text{不存在 } p_i = p\text{。} \end{cases}$

即，如果 n 的 p 进阶数 $\nu_p(n) = \alpha$，则 $p^\alpha \mid n$ 但 $p^{\alpha+1} \nmid n$，此时也记作 $p^\alpha \parallel n$，如果 $p^\alpha \parallel n$ 不成立，记作 $p^\alpha \nparallel n$。

同时，定义 n 的 **p 进赋值序列**为 $\langle \nu_2(n), \nu_3(n), \nu_5(n), \ldots \rangle$。

例 (p 进赋值序列)

- $1 \mapsto \langle 0, 0, 0, \ldots \rangle$
- $126 \mapsto \langle 1, 2, 0, 1, 0, \ldots \rangle$

标准分解式与 p 进赋值序列 – 性质

由算术基本定理可知，每个正整数与其 p 进赋值序列一一对应。

标准分解式与 p 进赋值序列 – 性质

由算术基本定理可知，每个正整数与其 p 进赋值序列一一对应。

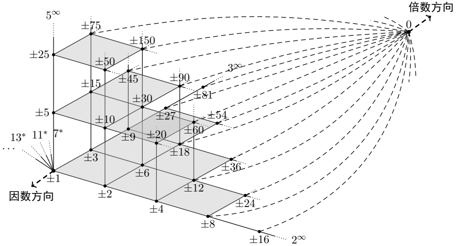
反之，每个只有有限项非零的非负整数序列，在看作 p 进赋值序列后，可以对应到所有正整数。

标准分解式与 p 进赋值序列 – 性质

由算术基本定理可知，每个正整数与其 p 进赋值序列一一对应。

反之，每个只有有限项非零的非负整数序列，在看作 p 进赋值序列后，可以对应到所有正整数。

所以，可以使用 p 进赋值序列来表示正整数，在 p 进赋值序列的视角下，每个正整数均位于无穷维空间的一个整点上，而每个维度均对应一个素数。



标准分解式与 p 进赋值序列 – 结论

对于 p 进阶数和 p 进赋值序列，有如下结论：

标准分解式与 p 进赋值序列 – 结论

对于 p 进阶数和 p 进赋值序列，有如下结论：

- $\nu_p(a \cdot b) = \nu_p(a) + \nu_p(b)$。
- $a \mid b$ 等价于对于所有素数 p 均有 $\nu_p(a) \le \nu_p(b)$。
- $\nu_p(\gcd(a, b)) = \min(\nu_p(a), \nu_p(b))$。
- $\nu_p(\operatorname{lcm}(a, b)) = \max(\nu_p(a), \nu_p(b))$。

标准分解式与 p 进赋值序列 – 结论

对于 p 进阶数和 p 进赋值序列，有如下结论：

- $\nu_p(a \cdot b) = \nu_p(a) + \nu_p(b)$。
- $a \mid b$ 等价于对于所有素数 p 均有 $\nu_p(a) \le \nu_p(b)$。
- $\nu_p(\gcd(a, b)) = \min(\nu_p(a), \nu_p(b))$。
- $\nu_p(\operatorname{lcm}(a, b)) = \max(\nu_p(a), \nu_p(b))$。
- 于是，$\gcd(a, b) \cdot \operatorname{lcm}(a, b) = a \cdot b$ 是显然的。

标准分解式与 p 进赋值序列 – 结论

对于 p 进阶数和 p 进赋值序列，有如下结论：

- $\nu_p(a \cdot b) = \nu_p(a) + \nu_p(b)$。
- $a \mid b$ 等价于对于所有素数 p 均有 $\nu_p(a) \le \nu_p(b)$。
- $\nu_p(\gcd(a, b)) = \min(\nu_p(a), \nu_p(b))$。
- $\nu_p(\operatorname{lcm}(a, b)) = \max(\nu_p(a), \nu_p(b))$。
- 于是，$\gcd(a, b) \cdot \operatorname{lcm}(a, b) = a \cdot b$ 是显然的。

其中，a, b 为非 0 正整数，p 为任意素数。

标准分解式与 p 进赋值序列 – 结论

对于 p 进阶数和 p 进赋值序列，有如下结论：

- $\nu_p(a \cdot b) = \nu_p(a) + \nu_p(b)$。
- $a \mid b$ 等价于对于所有素数 p 均有 $\nu_p(a) \leq \nu_p(b)$。
- $\nu_p(\gcd(a, b)) = \min(\nu_p(a), \nu_p(b))$。
- $\nu_p(\operatorname{lcm}(a, b)) = \max(\nu_p(a), \nu_p(b))$。
- 于是，$\gcd(a, b) \cdot \operatorname{lcm}(a, b) = a \cdot b$ 是显然的。

其中，a, b 为非 0 正整数，p 为任意素数。

据此，可以证明 a, b 的全体公因数就是 $\gcd(a, b)$ 的全体因数，以及 a, b 的全体公倍数就是 $\operatorname{lcm}(a, b)$ 的全体倍数。

标准分解式与 p 进赋值序列 – 完整图景

对于 0 和负整数，可以如下定义它们的 p 进阶数和 p 进赋值序列：

标准分解式与 p 进赋值序列 – 完整图景

对于 0 和负整数，可以如下定义它们的 p 进阶数和 p 进赋值序列：

- 对于任意素数 p，0 的 p 进阶数为 $\nu_p(0) = \infty$。
  0 的 p 进赋值序列为 $\langle\infty, \infty, \infty, \ldots\rangle$。
- 对于任意素数 p，负整数 $-n$ 的 p 进阶数为 $\nu_p(-n) = \nu_p(n)$。
  $-n$ 的 p 进赋值序列与 n 的 p 进赋值序列相同。

如此，前述的 gcd 和 lcm 相关公式便可以拓展到 0 和负整数。

标准分解式与 p 进赋值序列 – 完整图景

对于 0 和负整数，可以如下定义它们的 p 进阶数和 p 进赋值序列：

- 对于任意素数 p，0 的 p 进阶数为 $\nu_p(0)=\infty$。
  0 的 p 进赋值序列为 $\langle\infty,\infty,\infty,\dots\rangle$。
- 对于任意素数 p，负整数 $-n$ 的 p 进阶数为 $\nu_p(-n)=\nu_p(n)$。
  $-n$ 的 p 进赋值序列与 n 的 p 进赋值序列相同。

如此，前述的 $\gcd$ 和 lcm 相关公式便可以拓展到 0 和负整数。

并且，每个只有有限项非零的非负整数序列，在看作 p 进赋值序列后，恰好对应一正一负两个整数。而序列 $\langle\infty,\infty,\infty,\dots\rangle$ 对应在无穷维空间的无穷远点上的 0。
于是，整除偏序在无穷维空间上的嵌入的结构便是清晰的了。

2

基础算法

当前进度

素性测试

素性测试，即判断一个正整数 n 是否是素数。

素性测试

素性测试，即判断一个正整数 n 是否是素数。

算法竞赛中，常用的素性测试算法有试除法和 Miller–Rabin 算法。Miller–Rabin 算法是一个多项式时间的素性测试算法，即它的最坏时间复杂度关于 $\log n$ 呈多项式增长。

素性测试

素性测试，即判断一个正整数 n 是否是素数。

算法竞赛中，常用的素性测试算法有试除法和 Miller–Rabin 算法。Miller–Rabin 算法是一个多项式时间的素性测试算法，即它的最坏时间复杂度关于 $\log n$ 呈多项式增长。

一般的 Miller–Rabin 算法实现是非确定性的，有概率将合数报告为素数。对于确定性算法，有 Agrawal–Kayal–Saxena 算法同样是多项式时间的。

本课件中只介绍试除法。

试除法

试除法是素性测试的一种最简单的算法。

试除法

试除法是素性测试的一种最简单的算法。

试除法枚举所有在 2 到 $n-1$ 之间的整数 d，并通过计算 $n \bmod d$ 是否为 0 判断 d 是否为 n 的因数。如果 $n \neq 1$ 并且不存在这样的因数，则可以确认 n 是素数。

试除法

试除法是素性测试的一种最简单的算法。

试除法枚举所有在 2 到 $n-1$ 之间的整数 d，并通过计算 $n \bmod d$ 是否为 0 判断 d 是否为 n 的因数。如果 $n \neq 1$ 并且不存在这样的因数，则可以确认 n 是素数。试除法的 C++ 代码：

```cpp
bool primality_test(int n) {
  for (int d = 2; d <= n - 1; ++d)
    if (n % d == 0)
      return false;
  return n != 1;
}
```

可以看出，此算法的时间复杂度为 $\mathcal{O}(n)$。

试除法

试除法是素性测试的一种最简单的算法。

试除法枚举所有在 2 到 $\lfloor\sqrt{n}\rfloor$ 之间的整数 d，并通过计算 $n \bmod d$ 是否为 0 判断 d 是否为 n 的因数。如果 $n \neq 1$ 并且不存在这样的因数，则可以确认 n 是素数。试除法的 C++ 代码：

```
bool primality_test(int n) {
  for (int d = 2; d * d <= n; ++d)
    if (n % d == 0)
      return false;
  return n != 1;
}
```

可以看出，此算法的时间复杂度为 $\mathcal{O}(\sqrt{n})$。
此优化成立的原因是：如果 n 拥有非平凡因数，则其最小非平凡正因数 $\leq \sqrt{n}$。

当前进度

1 基础知识

2 基础算法
- 素性测试
- 素因数分解
- 求最大公因数

3 素数筛法

4 解线性 Diophantus 方程

5 附录

素因数分解

素因数分解，即给出一个正整数 n 的标准分解式。

素因数分解

素因数分解，即给出一个正整数 n 的标准分解式。

算法竞赛中，常用的素因数分解算法有试除法和 Pollard's rho 算法。Pollard's rho 算法首先对 n 执行任意素性测试算法，如果 n 为合数，设 n 的最小素因数为 p，该算法可以在 $\tilde{O}(\sqrt{p})$ 的时间复杂度内求出 n 的一个非平凡因数 a。对 a 和 $\frac{n}{a}$ 递归执行 Pollard's rho 算法即可完成素因数分解。

素因数分解

素因数分解，即给出一个正整数 n 的标准分解式。

算法竞赛中，常用的素因数分解算法有试除法和 Pollard's rho 算法。Pollard's rho 算法首先对 n 执行任意素性测试算法，如果 n 为合数，设 n 的最小素因数为 p，该算法可以在 $\tilde{O}(\sqrt{p})$ 的时间复杂度内求出 n 的一个非平凡因数 a。对 a 和 $\frac{n}{a}$ 递归执行 Pollard's rho 算法即可完成素因数分解。

素因数分解在经典计算模型下是否有多项式时间算法还未知，目前最高效的算法为普通数域筛法。
对于量子计算模型，有多项式时间的 Shor 算法。

本课件中只介绍试除法。

试除法

试除法是素因数分解的一种最简单的算法。

试除法

试除法是素因数分解的一种最简单的算法。

与素性测试类似，素因数分解的试除法枚举所有 2 到 $\lfloor\sqrt{n}\rfloor$ 之间的整数 d，并判断 d 是否为 n 的因数。
如果 $d \mid n$，则 $d = p$ 必然为素数，然后不断从 n 中除去素因数 p 直到 $p \nmid n$，从 n 中除去 p 的次数即为 $\nu_p(n)$。
注意，此时 d 的枚举上限，即 $\lfloor\sqrt{n}\rfloor$，是会随着 n 的减小而减小的，所以最终 d 可能将不会枚举到原数的平方根处。
最后，当由于 $d > \lfloor\sqrt{n}\rfloor$ 而退出循环后，如果 $n \neq 1$，则 n 为原数的标准分解式中的最后一个素数。

试除法 – C++ 代码

试除法的 C++ 代码：

```cpp
std::vector<std::pair<int, int>> prime_factorization(int n) {
  std::vector<std::pair<int, int>> ret;
  for (int d = 2; d * d <= n; ++d)
    if (n % d == 0) {
      int p = d, v = 0;
      while (n % p == 0)
        n /= p, ++v;
      ret.push_back({p, v});
    }
  if (n != 1)
    ret.push_back({n, 1});
  return ret;
}
```

可以看出，此算法的时间复杂度为 $\mathcal{O}(\sqrt{n})$。

当前进度

求最大公因数 – Euclid 算法

求最大公因数，即给定两个非负整数 a, b，计算 $\gcd(a, b)$，是数论中一个很特别的问题，这是由于解决它的算法，即 **Euclid 算法**，十分简单，而且有着非常优秀的时间复杂度。

注意，前文提到过，求非平凡因数在经典计算模型下是否有多项式时间算法还未知。事实上，包括 Pollard's rho 算法在内的许多数论算法都以 Euclid 算法为基础。

Euclid 算法

Euclid 算法基于如下结论：

- 当 $b = 0$ 时，$\gcd(a, b) = a$。
- 当 $b \neq 0$ 时，$\gcd(a, b) = \gcd(b, a \bmod b)$。

Euclid 算法

Euclid 算法基于如下结论：

- 当 $b = 0$ 时，$\gcd(a, b) = a$。
- 当 $b \neq 0$ 时，$\gcd(a, b) = \gcd(b, a \bmod b)$。

证明（当 $b = 0$ 时）.

由前文结论可知 $\gcd(a, b) = \gcd(a, 0) = a$。 （续……）

Euclid 算法

- 当 $b \neq 0$ 时，$\gcd(a, b) = \gcd(b, a \bmod b)$。

Euclid 算法

- 当 $b \neq 0$ 时，$\gcd(a, b) = \gcd(b, a \bmod b)$。

证明（当 $b \neq 0$ 时）.

在取 d 为 b 的任一因数的大前提下，注意到：

Euclid 算法

- 当 $b \neq 0$ 时，$\gcd(a, b) = \gcd(b, a \bmod b)$。

证明（当 $b \neq 0$ 时）.

在取 d 为 b 的任一因数的大前提下，注意到：
对于任意整数 x，如果 $d \mid x$，则 $d \mid (x + k \cdot b)$ 对于任意整数 k 均成立。形式化地说，即 $d \mid x \implies d \mid (x + k \cdot b)$。

Euclid 算法

- 当 $b \neq 0$ 时，$\gcd(a, b) = \gcd(b, a \bmod b)$。

证明（当 $b \neq 0$ 时）.

在取 d 为 b 的任一因数的大前提下，注意到：
对于任意整数 x，如果 $d \mid x$，则 $d \mid (x + k \cdot b)$ 对于任意整数 k 均成立。形式化地说，即 $d \mid x \implies d \mid (x + k \cdot b)$。

此时，我们令 $q = \left\lfloor \frac{a}{b} \right\rfloor$ 和 $r = a \bmod b$。注意 $r = a - q \cdot b$。

Euclid 算法

- 当 $b \neq 0$ 时，$\gcd(a,b) = \gcd(b, a \bmod b)$。

证明（当 $b \neq 0$ 时）.

在取 d 为 b 的任一因数的大前提下，注意到：
对于任意整数 x，如果 $d \mid x$，则 $d \mid (x + k \cdot b)$ 对于任意整数 k 均成立。形式化地说，即 $d \mid x \implies d \mid (x + k \cdot b)$。

此时，我们令 $q = \left\lfloor \frac{a}{b} \right\rfloor$ 和 $r = a \bmod b$。注意 $r = a - q \cdot b$。
请注意此处 $a - q \cdot b = r$ 和 $r + q \cdot b = a$ 的对称性，基于此我们有

$$d \mid a \implies d \mid (a - q \cdot b) \text{ 和 } d \mid r \implies d \mid (r + q \cdot b)$$

的双向关系。换句话说，即 $d \mid a \iff d \mid r$。

Euclid 算法

- 当 $b \neq 0$ 时，$\gcd(a, b) = \gcd(b, a \bmod b)$。

证明（当 $b \neq 0$ 时）.

考虑到 $d \mid b$ 的大前提，还可以形式化地写作

$$(d \mid b) \land (d \mid a) \iff (d \mid b) \land (d \mid a \bmod b)。$$

Euclid 算法

- 当 $b \neq 0$ 时，$\gcd(a, b) = \gcd(b, a \bmod b)$。

证明（当 $b \neq 0$ 时）.

考虑到 $d \mid b$ 的大前提，还可以形式化地写作

$$(d \mid b) \land (d \mid a) \iff (d \mid b) \land (d \mid a \bmod b)\text{。}$$

结合前文已证的 $(d \mid b) \land (d \mid a) \iff d \mid \gcd(b, a)$，可得 $d \mid \gcd(b, a) \iff d \mid \gcd(b, a \bmod b)$。

Euclid 算法

- 当 $b \neq 0$ 时，$\gcd(a, b) = \gcd(b, a \bmod b)$。

证明（当 $b \neq 0$ 时）.

考虑到 $d \mid b$ 的大前提，还可以形式化地写作

$$(d \mid b) \wedge (d \mid a) \iff (d \mid b) \wedge (d \mid a \bmod b)。$$

结合前文已证的 $(d \mid b) \wedge (d \mid a) \iff d \mid \gcd(b, a)$，可得 $d \mid \gcd(b, a) \iff d \mid \gcd(b, a \bmod b)$。
由于 $\gcd(b, a)$ 和 $\gcd(a \bmod b, b)$ 都是正整数，得到最终结论 $\gcd(a, b) = \gcd(b, a) = \gcd(b, a \bmod b)$。 □

Euclid 算法 – C++ 代码实现

Euclid 算法基于如下结论：

- 当 $b=0$ 时，$\gcd(a,b)=a$。
- 当 $b\neq 0$ 时，$\gcd(a,b)=\gcd(b,a \bmod b)$。

Euclid 算法 – C++ 代码实现

Euclid 算法基于如下结论：

- 当 $b = 0$ 时，$\gcd(a, b) = a$。
- 当 $b \neq 0$ 时，$\gcd(a, b) = \gcd(b, a \bmod b)$。

直接利用如上结论，可以给出 Euclid 算法的 C++ 代码：

```cpp
int gcd_Euclidean(int a, int b) {
  return b ? gcd_Euclidean(b, a % b) : a;
}
```

Euclid 算法 – C++ 代码实现

Euclid 算法基于如下结论：

- 当 $b = 0$ 时，$\gcd(a, b) = a$。
- 当 $b \neq 0$ 时，$\gcd(a, b) = \gcd(b, a \bmod b)$。

直接利用如上结论，可以给出 Euclid 算法的 C++ 代码：

```
int gcd_Euclidean(int a, int b) {
  return b ? gcd_Euclidean(b, a % b) : a;
}
```

我们将证明 Euclid 算法的时间复杂度为 $\mathcal{O}\left(\log \frac{\min(a,b)}{\gcd(a,b)}\right)$。
不过在此之前，我们先看 Euclid 算法的一个运行实例。

Euclid 算法 – 例子

假设我们需要计算 $\gcd(46, 240)$，则 Euclid 算法经过如下递归得到结果：

$$\begin{aligned}
\gcd(46,240) &= \gcd(240,46) && (46 \bmod 240 = 46) \\
&= \gcd(46,10) && (240 \bmod 46 = 10) \\
&= \gcd(10,6) && (46 \bmod 10 = 6) \\
&= \gcd(6,4) && (10 \bmod 6 = 4) \\
&= \gcd(4,2) && (6 \bmod 4 = 2) \\
&= \gcd(2,0) && (4 \bmod 2 = 0) \\
&= 2 && (\gcd(a,0) = a)
\end{aligned}$$

Euclid 算法 – 时间复杂度证明

称 Euclid 算法的一次递归为一步。

Euclid 算法 – 时间复杂度证明

称 Euclid 算法的一次递归为一步。

如果给定的 $\langle a, b\rangle$ 有其中之一为 0，则算法将在 2 步内结束。接下来假定 $a, b \neq 0$。
如果 $a < b$，则算法的第 1 步将交换 a, b，接下来假定 $a \geq b$。

Euclid 算法 – 时间复杂度证明

称 Euclid 算法的一次递归为一步。

如果给定的 $\langle a, b\rangle$ 有其中之一为 0，则算法将在 2 步内结束。接下来假定 $a, b \neq 0$。
如果 $a < b$，则算法的第 1 步将交换 a, b，接下来假定 $a \geq b$。

有此结论：对于正整数 a, b，如果 $a \geq b$，则 $a \bmod b < \frac{a}{2}$。
证明：如果 $b \leq a < 2 \cdot b$，则 $a \bmod b = a - b < a - \frac{a}{2} = \frac{a}{2}$，
否则 $a \geq 2 \cdot b$，则 $a \bmod b < b \leq \frac{a}{2}$。

Euclid 算法 – 时间复杂度证明

称 Euclid 算法的一次递归为一步。

如果给定的 $\langle a, b\rangle$ 有其中之一为 0，则算法将在 2 步内结束。接下来假定 $a, b \neq 0$。
如果 $a < b$，则算法的第 1 步将交换 a, b，接下来假定 $a \geq b$。

有此结论：对于正整数 a, b，如果 $a \geq b$，则 $a \bmod b < \frac{a}{2}$。
证明：如果 $b \leq a < 2 \cdot b$，则 $a \bmod b = a - b < a - \frac{a}{2} = \frac{a}{2}$，
否则 $a \geq 2 \cdot b$，则 $a \bmod b < b \leq \frac{a}{2}$。

所以，在算法接下来的每一步，a, b 两数中均有一个会变为原来的至多一半。总步数将被控制在 $\mathcal{O}(\log a + \log b)$ 内。

Euclid 算法 – 时间复杂度证明（续）

又注意到，如果 $a \geq b$，则第 1 步后两数为 $\langle b, a \bmod b\rangle$，两数均小于等于 b，接下来的时间复杂度为 $\mathcal{O}(\log b)$。所以，总步数将被控制在 $\mathcal{O}(\log \min(a, b))$ 内。

Euclid 算法 – 时间复杂度证明（续）

又注意到，如果 $a \geq b$，则第 1 步后两数为 $\langle b, a \bmod b\rangle$，两数均小于等于 b，接下来的时间复杂度为 $\mathcal{O}(\log b)$。所以，总步数将被控制在 $\mathcal{O}(\log \min(a, b))$ 内。

又由于 Euclid 算法在计算 $\langle a, b\rangle$ 时的过程和 $\left\langle \frac{a}{\gcd(a,b)}, \frac{b}{\gcd(a,b)} \right\rangle$ 完全一致（后者在计算中的值乘 $\gcd(a, b)$ 即得到前者）。
所以，Euclid 算法的时间复杂度为 $\mathcal{O}\left(\log \frac{\min(a,b)}{\gcd(a,b)}\right)$。

3

素数筛法

素数筛法 – 引入

接下来我们关心的是与素数分布有关的问题：

- 如何求出第 n 个素数?
- 如何求出小于等于 n 的最大素数?
- 如何求出大于等于 n 的最小素数?
- 在 $1 \sim n$ 的正整数中，一共有多少素数?
- ……

素数筛法 – 引入

接下来我们关心的是与素数分布有关的问题：

- 如何求出第 n 个素数?
- 如何求出小于等于 n 的最大素数?
- 如何求出大于等于 n 的最小素数?
- 在 $1 \sim n$ 的正整数中，一共有多少素数?
- ……

素数筛法可以解决这些问题，本课件中，我们介绍 Eratosthenes 筛法和 Euler 筛法。

当前进度

小学五年级数学

让我们复习一下五年级下册数学（人教版）。

小学五年级数学

一个数，如果只有 1 和它本身两个因数，这样的数叫做**质数**（或**素数**）。如 2，3，5，7 都是质数。

一个数，如果除了 1 和它本身还有别的因数，这样的数叫做**合数**。如 4，6，15，49 都是合数。

1 找出 100 以内的质数，做一个质数表。

1✗	2√	3√	~~4~~	5√	~~6~~	7√	~~8~~	~~9~~	~~10~~
11√	~~12~~	13√	~~14~~	~~15~~	~~16~~	17√	~~18~~	19√	~~20~~
~~21~~	~~22~~	23√	~~24~~	~~25~~	~~26~~	~~27~~	~~28~~	29√	~~30~~
31√	~~32~~	~~33~~	~~34~~	~~35~~	~~36~~	37√	~~38~~	~~39~~	~~40~~
41√	~~42~~	43√	~~44~~	~~45~~	~~46~~	47√	~~48~~	~~49~~	~~50~~
~~51~~	~~52~~	53√	~~54~~	~~55~~	~~56~~	~~57~~	~~58~~	59√	~~60~~
61√	~~62~~	~~63~~	~~64~~	~~65~~	~~66~~	67√	~~68~~	~~69~~	~~70~~
71√	~~72~~	73√	~~74~~	~~75~~	~~76~~	~~77~~	~~78~~	79√	~~80~~
~~81~~	~~82~~	83√	~~84~~	~~85~~	~~86~~	~~87~~	~~88~~	89√	~~90~~
~~91~~	~~92~~	~~93~~	~~94~~	~~95~~	~~96~~	97√	~~98~~	~~99~~	~~100~~

Eratosthenes 筛法

小明描述的算法相当于对 $1 \sim n$ 中每个数都进行一次素性测试。

Eratosthenes 筛法

小明描述的算法相当于对 $1 \sim n$ 中每个数都进行一次素性测试。
小红描述的过程就是 **Eratosthenes 筛法**。

Eratosthenes 筛法

小明描述的算法相当于对 $1 \sim n$ 中每个数都进行一次素性测试。
小红描述的过程就是 **Eratosthenes 筛法**。
小天使的问题“划到几的倍数就可以了?”的回答是“划到 $\lfloor\sqrt{n}\rfloor$ 的倍数就可以了”。这与试除法中只对不超过 $\lfloor\sqrt{n}\rfloor$ 的数进行因数判断原因相同。

Eratosthenes 筛法 – 算法流程

让我们再描述一次算法流程：

Eratosthenes 筛法 – 算法流程

让我们再描述一次算法流程：

1. 维护一个下标范围在 $2 \sim n$ 的 Boolean 数组，表示每个数是否被划去，初始时所有数都未被划去。

Eratosthenes 筛法 – 算法流程

让我们再描述一次算法流程：

1. 维护一个下标范围在 $2 \sim n$ 的 Boolean 数组，表示每个数是否被划去，初始时所有数都未被划去。
2. 从上一个找到的素数开始，找到数组中下一个未被划去的下标，将其标记为素数，并将其所有非本身的正倍数划去。

Eratosthenes 筛法 – 算法流程

让我们再描述一次算法流程：

1. 维护一个下标范围在 $2 \sim n$ 的 Boolean 数组，表示每个数是否被划去，初始时所有数都未被划去。
2. 从上一个找到的素数开始，找到数组中下一个未被划去的下标，将其标记为素数，并将其所有非本身的正倍数划去。
3. 重复步骤 2，直到不存在下一个未被划去的下标或此下标大于 $\lfloor\sqrt{n}\rfloor$ 即可停止，这意味着 $2 \sim n$ 中的所有素数均已被找到。

Eratosthenes 筛法 – 算法流程

让我们再描述一次算法流程：

1. 维护一个下标范围在 $2 \sim n$ 的 Boolean 数组，表示每个数是否被划去，初始时所有数都未被划去。
2. 从上一个找到的素数开始，找到数组中下一个未被划去的下标，将其标记为素数，并将其所有非本身的正倍数划去。
3. 重复步骤 2，直到不存在下一个未被划去的下标或此下标大于 $\lfloor \sqrt{n} \rfloor$ 即可停止，这意味着 $2 \sim n$ 中的所有素数均已被找到。
4. 将数组中剩余的未被划去的下标标记为素数。

Eratosthenes 筛法 – 算法流程

让我们再描述一次算法流程：

1. 维护一个下标范围在 $2 \sim n$ 的 Boolean 数组，表示每个数是否被划去，初始时所有数都未被划去。
2. 从上一个找到的素数开始，找到数组中下一个未被划去的下标，将其标记为素数，并将其所有非本身的正倍数划去。
3. 重复步骤 2，直到不存在下一个未被划去的下标或此下标大于 $\lfloor \sqrt{n} \rfloor$ 即可停止，这意味着 $2 \sim n$ 中的所有素数均已被找到。
4. 将数组中剩余的未被划去的下标标记为素数。

算法结束后，所有的素数按照从小到大的顺序被找到，并且 Boolean 数组记录了每个数的素性：如果被划去则是合数，否则是素数。

Eratosthenes 筛法 – C++ 实现

以下是 Eratosthenes 筛法的 C++ 代码：

```
bool is_composite[MaxN];
std::vector<int> sieve_of_Eratosthenes(int n) {
  std::vector<int> primes;
  for (int i = 2; i <= n; ++i)
    if (!is_composite[i]) {
      primes.push_back(i);
      if ((long long)i * i <= n)
        for (int j = 2 * i; j <= n; j += i)
          is_composite[j] = true;
    }
  return primes;
}
```

Eratosthenes 筛法 – C++ 实现

以下是 Eratosthenes 筛法的 C++ 代码：

```cpp
bool is_composite[MaxN];
std::vector<int> sieve_of_Eratosthenes(int n) {
  std::vector<int> primes;
  for (int i = 2; i <= n; ++i)
    if (!is_composite[i]) {
      primes.push_back(i);
      if ((long long)i * i <= n)
        for (int j = 2 * i; j <= n; j += i)
          is_composite[j] = true;
    }
  return primes;
}
```

接下来将解释 Eratosthenes 筛法的时间复杂度为 $\mathcal{O}(n \log \log n)$。

Eratosthenes 筛法 – 时间复杂度

Eratosthenes 筛法的时间复杂度由内层循环体的执行次数确定。即

$$\text{复杂度} \sim \sum_{p\le\sqrt{n}} \left\lfloor \frac{n}{p} \right\rfloor \sim n \sum_{p\le\sqrt{n}} \frac{1}{p} \sim n \ln\ln\left(\sqrt{n}\right) = \mathcal{O}(n \log\log n)。$$

其中，基于一结论 $\sum_{p\le n} \frac{1}{p} = \ln\ln n + \mathcal{O}(1)$。[1]

[1] 此结论即 Mertens 第二定理

Eratosthenes 筛法 – 时间复杂度

Eratosthenes 筛法的时间复杂度由内层循环体的执行次数确定。即

$$\text{复杂度} \sim \sum_{p \leq \sqrt{n}} \left\lfloor \frac{n}{p} \right\rfloor \sim n \sum_{p \leq \sqrt{n}} \frac{1}{p} \sim n \ln \ln(\sqrt{n}) = \mathcal{O}(n \log \log n)\text{。}$$

其中，基于一结论 $\sum_{p \leq n} \frac{1}{p} = \ln \ln n + \mathcal{O}(1)$。[1]

运行在一般个人电脑上时，如前述朴素实现的 Eratosthenes 筛法可以在 1 秒内处理出 $2 \sim n = 3 \times 10^7$ 甚至更大范围内的所有素数。

[1] 此结论即 Mertens 第二定理

区间筛法

问题 (区间求素数)

给定整数 l, r，求出 $l \sim r$ 内的所有素数。

- 数据范围：$1 \le l \le r \le 10^{14}$，$r - l \le 10^7$。

区间筛法

问题 (区间求素数)

给定整数 l, r，求出 $l \sim r$ 内的所有素数。

- 数据范围：$1 \le l \le r \le 10^{14}$，$r - l \le 10^7$。

可以对 Eratosthenes 筛法进行拓展以解决此问题。
考虑使用 Eratosthenes 筛法求出 $1 \sim \lfloor\sqrt{r}\rfloor$ 内的所有素数。
根据试除法中的结论，仅使用这些素数即可划去区间 $l \sim r$ 内的所有合数。与前文类似，时间复杂度可以分析为

$$\begin{aligned} & \mathcal{O}\left(\sqrt{r}\log\log r\right) + \mathcal{O}((r - l + 1)\log\log r) \\ = {} & \mathcal{O}\left(\left(r - l + \sqrt{r}\right)\log\log r\right)。\end{aligned}$$

当前进度

1 基础知识

2 基础算法

3 素数筛法

- Eratosthenes 筛法
- Euler 筛法

4 解线性 Diophantus 方程

5 附录

对 Eratosthenes 筛法进行改造

注意到，在 Eratosthenes 筛法中，每个合数被其每个素因数均划去了一次。实际上，若使用 $\omega(n)$ 表示正整数 n 的不同素因数个数，则 Eratosthenes 筛法的复杂度说明了 $\sum_{i=1}^{n} \omega(i) = \mathcal{O}(n \log \log n)$。

对 Eratosthenes 筛法进行改造

注意到，在 Eratosthenes 筛法中，每个合数被其每个素因数均划去了一次。实际上，若使用 $\omega(n)$ 表示正整数 n 的不同素因数个数，则 Eratosthenes 筛法的复杂度说明了 $\sum_{i=1}^{n} \omega(i) = \mathcal{O}(n \log \log n)$。

这启发我们改造 Eratosthenes 筛法，使得每个合数仅被其最小素因数划去。也即，每个素数 p 不会划去最小素因数不为它的合数。

对 Eratosthenes 筛法进行改造

注意到，在 Eratosthenes 筛法中，每个合数被其每个素因数均划去了一次。实际上，若使用 $\omega(n)$ 表示正整数 n 的不同素因数个数，则 Eratosthenes 筛法的复杂度说明了 $\displaystyle\sum_{i=1}^{n}\omega(i)=\mathcal{O}(n\log\log n)$。

这启发我们改造 Eratosthenes 筛法，使得每个合数仅被其最小素因数划去。也即，每个素数 p 不会划去最小素因数不为它的合数。

换句话说，考虑该合数为 $k=p\cdot i$，则 i 应为一个最小素因数大于等于 p 的正整数。

对 Eratosthenes 筛法进行改造

注意到，在 Eratosthenes 筛法中，每个合数被其每个素因数均划去了一次。实际上，若使用 $\omega(n)$ 表示正整数 n 的不同素因数个数，则 Eratosthenes 筛法的复杂度说明了 $\sum_{i=1}^{n}\omega(i)=\mathcal{O}(n\log\log n)$。

这启发我们改造 Eratosthenes 筛法，使得每个合数仅被其最小素因数划去。也即，每个素数 p 不会划去最小素因数不为它的合数。

换句话说，考虑该合数为 $k=p\cdot i$，则 i 应为一个最小素因数大于等于 p 的正整数。接下来，我们将展示改造后的算法的具体细节。

Euler 筛法

Eratosthenes 筛法的此改造得名于 (Euler, 1737)，即 **Euler 筛法**。

对于正整数 $n \geq 2$，记 $\operatorname{lpd}(n)$ 为 n 的最小素因数。
根据前文结果，在使用素数 p 筛去其倍数 $k = p \cdot i$ 时，需要确保 $\operatorname{lpd}(i)$ 大于等于 p。

Euler 筛法

Eratosthenes 筛法的此改造得名于 (Euler, 1737)，即 **Euler 筛法**。

对于正整数 $n \geq 2$，记 $\operatorname{lpd}(n)$ 为 n 的最小素因数。
根据前文结果，在使用素数 p 筛去其倍数 $k = p \cdot i$ 时，需要确保 $\operatorname{lpd}(i)$ 大于等于 p。为了在对 p 和 i 的双重枚举中确保这一点，有以下两种实现手段：

1. 外层枚举素数 p，内层枚举乘数 i，内层需要确保 $\operatorname{lpd}(i) \geq p$。
2. 外层枚举乘数 i，内层枚举素数 p，内层需要确保 $p \leq \operatorname{lpd}(i)$。

Euler 筛法

Eratosthenes 筛法的此改造得名于 (Euler, 1737)，即 **Euler 筛法**。

对于正整数 $n \geq 2$，记 $\operatorname{lpd}(n)$ 为 n 的最小素因数。
根据前文结果，在使用素数 p 筛去其倍数 $k = p \cdot i$ 时，需要确保 $\operatorname{lpd}(i)$ 大于等于 p。为了在对 p 和 i 的双重枚举中确保这一点，有以下两种实现手段：

1. 外层枚举素数 p，内层枚举乘数 i，内层需要确保 $\operatorname{lpd}(i) \geq p$。
2. 外层枚举乘数 i，内层枚举素数 p，内层需要确保 $p \leq \operatorname{lpd}(i)$。

在算法竞赛中，常见的是第 2 种实现手段。其原因是自然的：
枚举 $p \leq \operatorname{lpd}(i)$ 时，只需要从小到大枚举目前求出的素数 p，直到发现 $i \bmod p = 0$，即 $p \mid i$ 时跳出循环即可。

Euler 筛法 – C++ 实现

以下是 Euler 筛法的 C++ 代码：

```
bool is_composite[MaxN];
std::vector<int> sieve_of_Euler(int n) {
  std::vector<int> primes;
  for (int i = 2; i <= n; ++i) {
    if (!is_composite[i])
      primes.push_back(i);
    for (int p : primes) {
      int k = p * i;
      if (k > n) break;
      is_composite[k] = true;
      if (i % p == 0) break;
    }
  }
  return primes;
}
```

4

解线性 Diophantus 方程

当前进度

1 基础知识

2 基础算法

3 素数筛法

4 解线性 Diophantus 方程
- 问题引入
  - 线性 Diophantus 方程
  - 线性同余方程与有理数取模
- 问题解决

5 附录

Diophantus 方程

定义

Diophantus 方程指多元整系数方程，求解只在整数范围内进行。

数论中的许多经典问题与 Diophantus 方程有关，我们通常关心特定形式的 Diophantus 方程是否有解，若有解则试图判断解数是否有限，若有无限组解则希望求出通解形式。

Diophantus 方程 – 例子

例 (Diophantus 方程)

Diophantus 方程 – 例子

例 (Diophantus 方程)

- 关于 x, y, z 的方程 $x^n + y^n = z^n$ 为 Diophantus 方程。
已知 $n = 1, 2$ 时，此方程的通解形式已得到清晰理解。[2]
$n \geq 3$ 时，Fermat 猜想此方程仅有平凡解 $(0, 0, 0)$。[3]

[2] Pythagorean 三元组
[3] Fermat 猜想 (∼1637)，Fermat 大定理 (Wiles, 1995)

Diophantus 方程 – 例子

例 (Diophantus 方程)

- 关于 x, y, z 的方程 $x^n + y^n = z^n$ 为 Diophantus 方程。
  已知 $n = 1, 2$ 时，此方程的通解形式已得到清晰理解。[2]
  $n \geq 3$ 时，Fermat 猜想此方程仅有平凡解 $(0, 0, 0)$。[3]
- 关于 🍎, 🍌, 🍍 的方程 $\frac{🍎}{🍌 + 🍍} + \frac{🍌}{🍎 + 🍍} + \frac{🍍}{🍎 + 🍌} = 4$
  通分后可以变为 Diophantus 方程的形式。
  据说有 99% 的人无法找到正整数解。[4]

---

[2] Pythagorean 三元组

[3] Fermat 猜想 (∼1637)，Fermat 大定理 (Wiles, 1995)

[4] 一组正整数解为 $\langle$🍎, 🍌, 🍍$\rangle = \langle 15447680210874616644195131501991983748566432$

线性 Diophantus 方程

定义

关于 x, y 的方程 $ax + by = c$ 称为一个**线性 Diophantus 方程**。
其中 a, b, c 均为整数，并要求 a, b 不能同时为 0。

线性 Diophantus 方程

定义

关于 x, y 的方程 $ax + by = c$ 称为一个**线性 Diophantus 方程**。
其中 a, b, c 均为整数，并要求 a, b 不能同时为 0。

例 (线性 Diophantus 方程)

- $x + 2y = 0$ 为线性 Diophantus 方程。
- $6x - 10y = 4$ 为线性 Diophantus 方程。
- $6x - 10y = 3$ 为线性 Diophantus 方程。
- $0x + 0y = 0$ 不为线性 Diophantus 方程。

线性 Diophantus 方程 – 解集形式

例 (线性 Diophantus 方程的解集)

- 线性 Diophantus 方程 $x + 2y = 0$ 的解集为 $\langle x = -2k, y = k \rangle$ $(k \in \mathbb{Z})$。
- 线性 Diophantus 方程 $6x - 10y = 4$ 的解集为 $\langle x = 4 + 5k, y = 2 + 3k \rangle$ $(k \in \mathbb{Z})$。
- 线性 Diophantus 方程 $6x - 10y = 3$ 的解集为 $\varnothing$。

线性 Diophantus 方程 – 解集形式

可以看出，线性 Diophantus 方程的解集形式是以下二者之一：

- $\langle x_0 - kb', y_0 + ka' \rangle$（$\langle x_0, y_0 \rangle$ 为一组特解，$k \in \mathbb{Z}$）
- $\varnothing$

我们需要对某个特定方程的解集分类，并给出具体形式。

当前进度

1 基础知识

2 基础算法

3 素数筛法

4 解线性 Diophantus 方程
- 问题引入
  - 线性 Diophantus 方程
  - 线性同余方程与有理数取模
- 问题解决

5 附录

线性同余方程

定义

线性同余方程指关于 x 的形如 $ax \equiv b \pmod{m}$ 的同余方程。
其中 m 为正整数，a, b 为整数。

线性同余方程

定义

线性同余方程指关于 x 的形如 $ax \equiv b \pmod{m}$ 的同余方程。
其中 m 为正整数，a, b 为整数。

例 (线性同余方程)

- $x \equiv 0 \pmod{2}$ 为线性同余方程。
- $6x \equiv 4 \pmod{10}$ 为线性同余方程。
- $6x \equiv 3 \pmod{10}$ 为线性同余方程。

线性同余方程 – 解集形式

例 (线性同余方程的解集)

- 线性同余方程 $x \equiv 0 \pmod 2$ 的形式已足够简洁，解集为 $x \equiv 0 \pmod 2$。
- 线性同余方程 $6x \equiv 4 \pmod{10}$ 的解集为 $x \equiv 4 \pmod 5$。
- 线性同余方程 $6x \equiv 3 \pmod{10}$ 的解集为 $\varnothing$。

线性同余方程 – 解集形式

例 (线性同余方程的解集)

- 线性同余方程 $x \equiv 0 \pmod 2$ 的形式已足够简洁，解集为 $x \equiv 0 \pmod 2$。
- 线性同余方程 $6x \equiv 4 \pmod{10}$ 的解集为 $x \equiv 4 \pmod 5$。
- 线性同余方程 $6x \equiv 3 \pmod{10}$ 的解集为 $\varnothing$。

可以看出，线性同余方程的解集形式是以下二者之一：

- $x \equiv x_0 \pmod{m'}$
- $\varnothing$

我们需要对某个特定方程的解集分类，并给出具体形式。

线性同余方程 – 与线性 Diophantus 方程的联系

事实上，有线性同余方程 $ax \equiv b \pmod{m}$ 与 $m \mid (ax - b)$ 等价，并进一步等价于存在 k 使得 $km = ax - b$ 成立，令 $y = -k$，移项后变为 $ax + my = b$，可以看作一个关于 x, y 的线性 Diophantus 方程。

线性同余方程 – 与线性 Diophantus 方程的联系

事实上，有线性同余方程 $ax \equiv b \pmod{m}$ 与 $m \mid (ax - b)$ 等价，并进一步等价于存在 k 使得 $km = ax - b$ 成立，令 $y = -k$，移项后变为 $ax + my = b$，可以看作一个关于 x, y 的线性 Diophantus 方程。

这意味着，只需要解决 $ax + my = b$ 这一线性 Diophantus 方程，即可解决原线性同余方程。

有理数取模

目前我们已经定义整数对另一正整数的模，也即进行有余数的除法后得到的余数。

从同余等价的角度考虑，对于有理数来说，我们也希望通过同余关系定义有理数对一正整数的模，并期望如此定义的模保持运算性质。

有理数取模

定义 (有理数取模)

给定正整数 m，定义 $(* \bmod_{\mathbb{Q}} m) : \mathbb{Q} \to \{0, 1, \ldots, m-1\}$，保持：

- 对于整数 n，$n \bmod_{\mathbb{Q}} m = n \bmod m$
- $((a \bmod_{\mathbb{Q}} m) \pm (b \bmod_{\mathbb{Q}} m)) \bmod m = (a \pm b) \bmod_{\mathbb{Q}} m$
- $((a \bmod_{\mathbb{Q}} m) \cdot (b \bmod_{\mathbb{Q}} m)) \bmod m = (a \cdot b) \bmod_{\mathbb{Q}} m$

其中，a, b 为任意有理数。

有理数取模

定义 (有理数取模)

给定正整数 m，定义 $(*\bmod_{\mathbb{Q}} m) : \mathbb{Q} \to \{0, 1, \ldots, m-1\}$，保持：

- 对于整数 n，$n \bmod_{\mathbb{Q}} m = n \bmod m$
- $((a \bmod_{\mathbb{Q}} m) \pm (b \bmod_{\mathbb{Q}} m)) \bmod m = (a \pm b) \bmod_{\mathbb{Q}} m$
- $((a \bmod_{\mathbb{Q}} m) \cdot (b \bmod_{\mathbb{Q}} m)) \bmod m = (a \cdot b) \bmod_{\mathbb{Q}} m$

其中，a, b 为任意有理数。

然而，注意到，当我们取 $a = m$ 而 $b = \frac{1}{m}$ 时，如果 $b \mapsto x$，则必须有 $0 \cdot x \equiv 1 \pmod{m}$，而这当 $m \neq 1$ 时是不可能的。

有理数取模 – 困难与联系

正如前文所述，在定义 $m \neq 1$ 时的有理数取模时遇到了困难。

有理数取模 – 困难与联系

正如前文所述，在定义 $m \neq 1$ 时的有理数取模时遇到了困难。

让我们考虑一有理数的最简分数表示为 $x = \frac{b}{a}$，则根据上述定义，需要满足 $((a \operatorname{mod}_{\mathbb{Q}} m) \cdot (x \operatorname{mod}_{\mathbb{Q}} m)) \bmod m = b \operatorname{mod}_{\mathbb{Q}} m$。

有理数取模 – 困难与联系

正如前文所述，在定义 $m \neq 1$ 时的有理数取模时遇到了困难。

让我们考虑一有理数的最简分数表示为 $x = \frac{b}{a}$，则根据上述定义，需要满足 $((a \operatorname{mod}_{\mathbb{Q}} m) \cdot (x \operatorname{mod}_{\mathbb{Q}} m)) \bmod m = b \operatorname{mod}_{\mathbb{Q}} m$。
令 $x' = x \operatorname{mod}_{\mathbb{Q}} m$。上式即

$$ax' \equiv b \pmod{m}。$$

即一线性同余方程。如果此方程有解，则有希望定义有理数 x 对 m 的模。

有理数取模 – 困难与联系

正如前文所述，在定义 $m \neq 1$ 时的有理数取模时遇到了困难。

让我们考虑一有理数的最简分数表示为 $x = \frac{b}{a}$，则根据上述定义，需要满足 $((a \bmod_{\mathbb{Q}} m) \cdot (x \bmod_{\mathbb{Q}} m)) \bmod m = b \bmod_{\mathbb{Q}} m$。
令 $x' = x \bmod_{\mathbb{Q}} m$。上式即

$$ax' \equiv b \pmod{m}。$$

即一线性同余方程。如果此方程有解，则有希望定义有理数 x 对 m 的模。

而线性同余方程可以被转化为线性 Diophantus 方程。接下来我们将展示线性 Diophantus 方程的求解方法。

当前进度

Bézout 定理

为了解决方程 $ax + by = c$，首先考察当 x, y 任取时，$ax + by$ 的取值。接下来我们给出 **Bézout 定理（Bézout's lemma）**。

Bézout 定理

为了解决方程 $ax+by=c$，首先考察当 x,y 任取时，$ax+by$ 的取值。接下来我们给出 **Bézout 定理（Bézout's lemma）**。

定理 (Bézout 定理)

回顾前文提到过的 $a_1,a_2,\ldots,a_n$ 的整系数线性组合。
令 $g=\gcd(a_1,a_2,\ldots,a_n)$，则 $a_{1\sim n}$ 的所有整系数线性组合恰好就是 g 的所有倍数。

Bézout 定理

为了解决方程 $ax+by=c$，首先考察当 x,y 任取时，$ax+by$ 的取值。接下来我们给出 **Bézout 定理（Bézout's lemma）**。

定理 (Bézout 定理)

回顾前文提到过的 $a_1,a_2,\ldots,a_n$ 的整系数线性组合。
令 $g=\gcd(a_1,a_2,\ldots,a_n)$，则 $a_{1\sim n}$ 的所有整系数线性组合恰好就是 g 的所有倍数。

例 (Bézout 定理)

- 当 $a=[6,12,16]$ 时，a 的整系数线性组合有 $\{\ldots,-4,-2,0,2,4,\ldots\}$，其中 $2=\gcd(6,12,16)$。

Bézout 定理 – 证明

Bézout 定理.

Bézout 定理 – 证明

Bézout 定理.

当 $g=0$，即 $a_{1\sim n}$ 均为 0 时，定理显然成立。接下来假定 $g\geq 1$。

Bézout 定理 – 证明

Bézout 定理.

当 $g=0$，即 $a_{1\sim n}$ 均为 0 时，定理显然成立。接下来假定 $g\ge 1$。

若 $g\ge 1$ 则说明至少有一个 $a_k\ne 0$，则 a 的整系数线性组合中至少有一正元素 $|a_k|$。取 s 为 a 的整系数线性组合中的最小正元素。

Bézout 定理 – 证明

Bézout 定理.

当 $g=0$，即 $a_{1\sim n}$ 均为 0 时，定理显然成立。接下来假定 $g\geq 1$。

若 $g\geq 1$ 则说明至少有一个 $a_k\neq 0$，则 a 的整系数线性组合中至少有一正元素 $|a_k|$。取 s 为 a 的整系数线性组合中的最小正元素。考虑进行有余数的除法，令 a_i 除以 s 的商为 q，余数为 r。那么有 $r=a_i-q\cdot s$，同样是一个整系数线性组合。

Bézout 定理 – 证明

Bézout 定理.

当 $g = 0$，即 $a_{1\sim n}$ 均为 0 时，定理显然成立。接下来假定 $g \geq 1$。

若 $g \geq 1$ 则说明至少有一个 $a_k \neq 0$，则 a 的整系数线性组合中至少有一正元素 $|a_k|$。取 s 为 a 的整系数线性组合中的最小正元素。
考虑进行有余数的除法，令 a_i 除以 s 的商为 q，余数为 r。那么有 $r = a_i - q \cdot s$，同样是一个整系数线性组合。
然而 $0 \leq r < s$，结合 s 的最小性，只能有 $r = 0$，这说明 $s \mid a_i$。

Bézout 定理 – 证明

Bézout 定理.

当 $g = 0$，即 $a_{1\sim n}$ 均为 0 时，定理显然成立。接下来假定 $g \geq 1$。

若 $g \geq 1$ 则说明至少有一个 $a_k \neq 0$，则 a 的整系数线性组合中至少有一正元素 $|a_k|$。取 s 为 a 的整系数线性组合中的最小正元素。
考虑进行有余数的除法，令 a_i 除以 s 的商为 q，余数为 r。那么有 $r = a_i - q \cdot s$，同样是一个整系数线性组合。
然而 $0 \leq r < s$，结合 s 的最小性，只能有 $r = 0$，这说明 $s \mid a_i$。
由 i 的任意性，这说明 s 是 $a_{1\sim n}$ 的一个公因数。于是 $s \mid g$。

Bézout 定理 – 证明

Bézout 定理.

当 $g=0$，即 $a_{1\sim n}$ 均为 0 时，定理显然成立。接下来假定 $g\geq 1$。

若 $g\geq 1$ 则说明至少有一个 $a_k\neq 0$，则 a 的整系数线性组合中至少有一正元素 $|a_k|$。取 s 为 a 的整系数线性组合中的最小正元素。
考虑进行有余数的除法，令 a_i 除以 s 的商为 q，余数为 r。那么有 $r=a_i-q\cdot s$，同样是一个整系数线性组合。
然而 $0\leq r<s$，结合 s 的最小性，只能有 $r=0$，这说明 $s\mid a_i$。
由 i 的任意性，这说明 s 是 $a_{1\sim n}$ 的一个公因数。于是 $s\mid g$。
然而 g 应整除任意整系数线性组合，即 $g\mid s$。于是只能有 $g=s$。 (续……)

Bézout 定理 – 证明

Bézout 定理（续）.

已证 g 即为所有整系数线性组合中的最小正元素。

Bézout 定理 – 证明

Bézout 定理（续）.

已证 g 即为所有整系数线性组合中的最小正元素。
若某一整系数线性组合 x 不是 g 的倍数，则根据已证结论，$\gcd(g,x)$ 也为一整系数线性组合。然而由于 $g \nmid x$，有 $0 < \gcd(g,x) < g$，与 g 的最小性矛盾。

Bézout 定理 – 证明

Bézout 定理（续）.

已证 g 即为所有整系数线性组合中的最小正元素。
若某一整系数线性组合 x 不是 g 的倍数，则根据已证结论，$\gcd(g, x)$ 也为一整系数线性组合。然而由于 $g \nmid x$，有 $0 < \gcd(g, x) < g$，与 g 的最小性矛盾。
故所有整系数线性组合均是 $g = \gcd(a_1, a_2, \ldots, a_n)$ 的倍数。又有整系数线性组合的倍数自然也为整系数线性组合，故所有整系数线性组合恰好就是 $g = \gcd(a_1, a_2, \ldots, a_n)$ 的所有倍数。 □

解的存在性判定

Bézout 定理解决了线性 Diophantus 方程的解的存在性判定问题。对于 $ax + by = c$，令 $g = \gcd(a, b)$，则方程有解等价于 $g \mid c$。

解的存在性判定

Bézout 定理解决了线性 Diophantus 方程的解的存在性判定问题。
对于 $ax + by = c$，令 $g = \gcd(a, b)$，则方程有解等价于 $g \mid c$。

方程有解时，对于解的结构，Bézout 定理并未给出更多信息。

当前进度

齐次情况

考察当 $c = 0$ 时的齐次情况：$ax + by = 0$。

齐次情况

考察当 $c = 0$ 时的齐次情况：$ax + by = 0$。
若在所有整数上枚举 y 的取值，对 x 解得 $x = -\frac{by}{a}$。
要让 x 为整数，需要 $a \mid by$。

齐次情况

考察当 $c=0$ 时的齐次情况：$ax+by=0$。
若在所有整数上枚举 y 的取值，对 x 解得 $x=-\frac{by}{a}$。
要让 x 为整数，需要 $a \mid by$。

从 p 进赋值序列的视角考虑，即

$$\begin{aligned} \nu_p(a) &\le \nu_p(b)+\nu_p(y)\text{，} \\ \text{这等价于 } \nu_p(y) &\ge \max(\nu_p(a)-\nu_p(b),0) \\ &= \nu_p(a)-\min(\nu_p(b),\nu_p(a)) \\ &= \nu_p\left(\frac{a}{\gcd(a,b)}\right)\text{。} \end{aligned}$$

齐次情况

考察当 $c = 0$ 时的齐次情况：$ax + by = 0$。
若在所有整数上枚举 y 的取值，对 x 解得 $x = -\frac{by}{a}$。
要让 x 为整数，需要 $a \mid by$。

从 p 进赋值序列的视角考虑，即

$$\begin{aligned} \nu_p(a) &\le \nu_p(b) + \nu_p(y), \\ \text{这等价于 } \nu_p(y) &\ge \max(\nu_p(a) - \nu_p(b), 0) \\ &= \nu_p(a) - \min(\nu_p(b), \nu_p(a)) \\ &= \nu_p\left(\frac{a}{\gcd(a,b)}\right)。 \end{aligned}$$

即 y 应为 $\frac{a}{\gcd(a,b)}$ 的倍数。

齐次情况

考察当 $c = 0$ 时的齐次情况：$ax + by = 0$。
设 $y = k \cdot \frac{a}{\gcd(a,b)}$，解得 $x = -k \cdot \frac{b}{\gcd(a,b)}$，代回原式验证：

$$a \cdot \left(-k \cdot \frac{b}{\gcd(a,b)}\right) + b \cdot \left(k \cdot \frac{a}{\gcd(a,b)}\right) = 0$$

确实成立。

齐次情况

考察当 $c=0$ 时的齐次情况：$ax+by=0$。
设 $y=k\cdot\frac{a}{\gcd(a,b)}$，解得 $x=-k\cdot\frac{b}{\gcd(a,b)}$，代回原式验证：

$$a\cdot\left(-k\cdot\frac{b}{\gcd(a,b)}\right)+b\cdot\left(k\cdot\frac{a}{\gcd(a,b)}\right)=0$$

确实成立。

也即，令 $a'=\frac{a}{\gcd(a,b)}$ 和 $b'=\frac{b}{\gcd(a,b)}$，
有通解形式 $\langle x,y\rangle=\langle -kb',ka'\rangle$。

齐次情况 – 特解与通解

如上给出了齐次情况的通解形式 $\langle -kb', ka' \rangle$，这大大简化了问题。

齐次情况 – 特解与通解

如上给出了齐次情况的通解形式 $\langle -kb', ka' \rangle$，这大大简化了问题。

这是因为，考虑 $ax + by = c$ 的一特解 $\langle x_0, y_0 \rangle$，则此特解加上任意齐次情况下的解 $\langle x_1, y_1 \rangle$ 得到的 $\langle x_0 + x_1, y_0 + y_1 \rangle$ 必然也是原方程的解。
反之，任意原方程的解 $\langle x_2, y_2 \rangle$ 都能通过与特解 $\langle x_0, y_0 \rangle$ 相减得到齐次情况下的解 $\langle x_0 - x_2, y_0 - y_2 \rangle$。

齐次情况 – 特解与通解

如上给出了齐次情况的通解形式 $\langle -kb', ka' \rangle$，这大大简化了问题。

这是因为，考虑 $ax + by = c$ 的一特解 $\langle x_0, y_0 \rangle$，则此特解加上任意齐次情况下的解 $\langle x_1, y_1 \rangle$ 得到的 $\langle x_0 + x_1, y_0 + y_1 \rangle$ 必然也是原方程的解。
反之，任意原方程的解 $\langle x_2, y_2 \rangle$ 都能通过与特解 $\langle x_0, y_0 \rangle$ 相减得到齐次情况下的解 $\langle x_0 - x_2, y_0 - y_2 \rangle$。

也即，$ax + by = c$ 的解集必然呈现一特解 $\langle x_0, y_0 \rangle$ 加上任意齐次情况下的通解 $\left\langle -k \cdot \frac{b}{\gcd(a,b)}, k \cdot \frac{a}{\gcd(a,b)} \right\rangle$ 的形式。

齐次情况 – 特解与通解

如上给出了齐次情况的通解形式 $\langle -kb', ka' \rangle$，这大大简化了问题。

这是因为，考虑 $ax + by = c$ 的一特解 $\langle x_0, y_0 \rangle$，则此特解加上任意齐次情况下的解 $\langle x_1, y_1 \rangle$ 得到的 $\langle x_0 + x_1, y_0 + y_1 \rangle$ 必然也是原方程的解。
反之，任意原方程的解 $\langle x_2, y_2 \rangle$ 都能通过与特解 $\langle x_0, y_0 \rangle$ 相减得到齐次情况下的解 $\langle x_0 - x_2, y_0 - y_2 \rangle$。

也即，$ax + by = c$ 的解集必然呈现一特解 $\langle x_0, y_0 \rangle$ 加上任意齐次情况下的通解 $\left\langle -k \cdot \frac{b}{\gcd(a,b)}, k \cdot \frac{a}{\gcd(a,b)} \right\rangle$ 的形式。

接下来只需解决求出一组特解的问题即可。

当前进度

扩展 Euclid 算法 – 引入

Bézout 定理显示了 $ax + by = \gcd(a, b)$ 必定有解。
进一步地，注意到 $ax + by = c$ 有解当且仅当 $\gcd(a, b) \mid c$，则求出 $ax + by = \gcd(a, b)$ 的一组特解 $\langle x_0, y_0 \rangle$ 后，将其乘 $\frac{c}{\gcd(a,b)}$ 后即可得到 $ax + by = c$ 的一组特解。

扩展 Euclid 算法 – 引入

Bézout 定理显示了 $ax + by = \gcd(a, b)$ 必定有解。
进一步地，注意到 $ax + by = c$ 有解当且仅当 $\gcd(a, b) \mid c$，则求出 $ax + by = \gcd(a, b)$ 的一组特解 $\langle x_0, y_0\rangle$ 后，将其乘 $\frac{c}{\gcd(a,b)}$ 后即可得到 $ax + by = c$ 的一组特解。

接下来介绍**扩展 Euclid 算法**以解决 $ax + by = \gcd(a, b)$ 的特解问题。注意到 Bézout 定理给出的形式中带有 $\gcd(a, b)$，这启发我们改造 Euclid 算法，这也就是算法的名字由来。

扩展 Euclid 算法

Euclid 算法依赖有余数的除法进行，其递归形式形如：

$$
\begin{aligned}
a &= q_1 \cdot b + r_1 \\
b &= q_2 \cdot r_1 + r_2 \\
r_1 &= q_3 \cdot r_2 + r_3 \\
&\vdots \\
r_{n-2} &= q_n \cdot r_{n-1} + r_n \\
r_{n-1} &= q_{n+1} \cdot r_n + r_{n+1}
\end{aligned}
$$

其中 $r_{n+1} = 0$，于是递归过程在计算 $\gcd(r_n, r_{n+1})$ 时停止。
最终有 $g = \gcd(a, b) = r_n$。

扩展 Euclid 算法

Euclid 算法依赖有余数的除法进行，其递归形式形如：

$$
\begin{aligned}
a &= q_1 &\cdot b &+ r_1 \\
b &= q_2 &\cdot r_1 &+ r_2 \\
r_1 &= q_3 &\cdot r_2 &+ r_3 \\
&\vdots \\
r_{n-2} &= q_n &\cdot r_{n-1} &+ r_n \\
r_{n-1} &= q_{n+1} &\cdot r_n &+ r_{n+1}
\end{aligned}
$$

其中 $r_{n+1} = 0$，于是递归过程在计算 $\gcd(r_n, r_{n+1})$ 时停止。
最终有 $g = \gcd(a, b) = r_n$。

当计算 $\gcd(r_n, r_{n+1})$ 时有 $1 \cdot r_n + 0 \cdot r_{n+1} = g$，此即为边界。

扩展 Euclid 算法

考察连续两层递归 $\gcd(r_{i-2}, r_{i-1}) \rightleftharpoons \gcd(r_{i-1}, r_i)$，它基于：

$$r_{i-2} = q_i \cdot r_{i-1} + r_i。$$

假设已求出 $x_{i+1} \cdot r_{i-1} + y_{i+1} \cdot r_i = g$。
现在要求 $x_i \cdot r_{i-2} + y_i \cdot r_{i-1} = g$ 的一组特解。

扩展 Euclid 算法

考察连续两层递归 $\gcd(r_{i-2}, r_{i-1}) \rightleftharpoons \gcd(r_{i-1}, r_i)$，它基于：

$$r_{i-2} = q_i \cdot r_{i-1} + r_i。$$

假设已求出 $x_{i+1} \cdot r_{i-1} + y_{i+1} \cdot r_i = g$。
现在要求 $x_i \cdot r_{i-2} + y_i \cdot r_{i-1} = g$ 的一组特解。
考虑 $r_i = r_{i-2} - q_i \cdot r_{i-1}$，将其代入 $x_{i+1} \cdot r_{i-1} + y_{i+1} \cdot r_i = g$ 得

$$\begin{aligned} & x_{i+1} \cdot r_{i-1} + y_{i+1} \cdot (r_{i-2} - q_i \cdot r_{i-1}) = g \\ \Longrightarrow\ & y_{i+1} \cdot r_{i-2} + (x_{i+1} - q_i \cdot y_{i+1}) \cdot r_{i-1} = g。\end{aligned}$$

于是有 $\langle x_i, y_i \rangle = \langle y_{i+1}, x_{i+1} - q_i \cdot y_{i+1} \rangle$。根据此式从边界 $\langle x_{n+2}, y_{n+2} \rangle = \langle 1, 0 \rangle$ 一步步回溯至 $\langle x_1, y_1 \rangle$ 即得原方程的一组特解。

扩展 Euclid 算法 – C++ 实现

以下是扩展 Euclid 算法的 C++ 代码（C++17）：

```
std::pair<int, int> extended_Euclidean(int a, int b) {
  if (!b) return {1, 0};
  const auto &[x, y] = extended_Euclidean(b, a % b);
  return {y, x - a / b * y};
}
```

扩展 Euclid 算法 – C++ 实现

以下是扩展 Euclid 算法的 C++ 代码（C++17）：

```
std::pair<int, int> extended_Euclidean(int a, int b) {
  if (!b) return {1, 0};
  const auto &[x, y] = extended_Euclidean(b, a % b);
  return {y, x - a / b * y};
}
```

在 C++98 标准下可过编译的代码：

```
int extended_Euclidean(int a, int b, int &x, int &y) {
  if (!b) return x = 1, y = 0, a;
  int d = extended_Euclidean(b, a % b, y, x);
  return y -= a / b * x, d;
}
```

扩展 Euclid 算法 – C++ 实现

以下是扩展 Euclid 算法的 C++ 代码（C++17）：

```
std::pair<int, int> extended_Euclidean(int a, int b) {
  if (!b) return {1, 0};
  const auto &[x, y] = extended_Euclidean(b, a % b);
  return {y, x - a / b * y};
}
```

在 C++98 标准下可过编译的代码：

```
int extended_Euclidean(int a, int b, int &x, int &y) {
  if (!b) return x = 1, y = 0, a;
  int d = extended_Euclidean(b, a % b, y, x);
  return y -= a / b * x, d;
}
```

扩展 Euclid 算法的递归过程与 Euclid 算法相同，于是其时间复杂度也为 $\mathcal{O}\left(\log \frac{\min(a,b)}{\gcd(a,b)}\right)$。

扩展 Euclid 算法 – 例子

$$
\begin{array}{c|r|r}
\gcd(240, 46) & & -9 \cdot 240 + 47 \cdot 46 = 2 \\
\downarrow & 240 = 5 \cdot 46 + 10 & \uparrow \\
\gcd(46, 10) & & 2 \cdot 46 - 9 \cdot 10 = 2 \\
\downarrow & 46 = 4 \cdot 10 + 6 & \uparrow \\
\gcd(10, 6) & & -1 \cdot 10 + 2 \cdot 6 = 2 \\
\downarrow & 10 = 1 \cdot 6 + 4 & \uparrow \\
\gcd(6, 4) & & 1 \cdot 6 - 1 \cdot 4 = 2 \\
\downarrow & 6 = 1 \cdot 4 + 2 & \uparrow \\
\gcd(4, 2) & & 0 \cdot 4 + 1 \cdot 2 = 2 \\
\downarrow & 4 = 2 \cdot 2 + 0 & \uparrow \\
\gcd(2, 0) & & 1 \cdot 2 + 0 \cdot 0 = 2
\end{array}
$$

扩展 Euclid 算法 – 特解的最小性

当对不能同时为 0 的非负整数 a, b 运行如上编写的扩展 Euclid 算法时，有性质：

扩展 Euclid 算法 – 特解的最小性

当对不能同时为 0 的非负整数 a, b 运行如上编写的扩展 Euclid 算法时，有性质：

- 如果 $b \mid a$，那么将解得 $0 \cdot a + 1 \cdot b = \gcd(a, b) = b$。
- 如果 $b \nmid a$ 且 $a \mid b$，那么将解得 $1 \cdot a + 0 \cdot b = \gcd(a, b) = a$。

扩展 Euclid 算法 – 特解的最小性

当对不能同时为 0 的非负整数 a, b 运行如上编写的扩展 Euclid 算法时，有性质：

- 如果 $b \mid a$，那么将解得 $0 \cdot a + 1 \cdot b = \gcd(a, b) = b$。
- 如果 $b \nmid a$ 且 $a \mid b$，那么将解得 $1 \cdot a + 0 \cdot b = \gcd(a, b) = a$。
- 如果 $b \nmid a$ 且 $a \nmid b$，那么将解得的 $\langle x, y \rangle$ 满足

$$|x| \le \frac{b}{2\gcd(a,b)} \text{ 以及 } |y| \le \frac{a}{2\gcd(a,b)}。$$

扩展 Euclid 算法 – 特解的最小性

当对不能同时为 0 的非负整数 a, b 运行如上编写的扩展 Euclid 算法时，有性质：

- 如果 $b \mid a$，那么将解得 $0 \cdot a + 1 \cdot b = \gcd(a, b) = b$。
- 如果 $b \nmid a$ 且 $a \mid b$，那么将解得 $1 \cdot a + 0 \cdot b = \gcd(a, b) = a$。
- 如果 $b \nmid a$ 且 $a \nmid b$，那么将解得的 $\langle x, y \rangle$ 满足

$$|x| \le \frac{b}{2\gcd(a,b)} \text{ 以及 } |y| \le \frac{a}{2\gcd(a,b)}。$$

这意味着解得的 $\langle x, y \rangle$ 是所有解中绝对值最小的一组，并且同时说明算法运行过程中无需担心整型溢出的问题，将 x, y 使用与 a, b 相同的类型存储就足够了。

解线性 Diophantus 方程 – 总结

解线性 Diophantus 方程 $ax + by = c$ 时只需进行下列步骤。

解线性 Diophantus 方程 – 总结

解线性 Diophantus 方程 $ax + by = c$ 时只需进行下列步骤。

1. 用扩展 Euclid 算法求 $ax + by = \gcd(a, b)$ 的特解 $\langle x_0, y_0 \rangle$。

解线性 Diophantus 方程 – 总结

解线性 Diophantus 方程 $ax + by = c$ 时只需进行下列步骤。

1. 用扩展 Euclid 算法求 $ax + by = \gcd(a, b)$ 的特解 $\langle x_0, y_0 \rangle$。
2. 判断是否有 $\gcd(a, b) \mid c$，如果 $\gcd(a, b) \nmid c$ 则无解。

解线性 Diophantus 方程 – 总结

解线性 Diophantus 方程 $ax + by = c$ 时只需进行下列步骤。

1. 用扩展 Euclid 算法求 $ax + by = \gcd(a, b)$ 的特解 $\langle x_0, y_0 \rangle$。
2. 判断是否有 $\gcd(a, b) \mid c$，如果 $\gcd(a, b) \nmid c$ 则无解。
3. 如果 $\gcd(a, b) \mid c$，则原方程通解形式为（$k \in \mathbb{Z}$）：

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

其中 $a' = \frac{a}{\gcd(a,b)}$ 且 $b' = \frac{b}{\gcd(a,b)}$ 且 $c' = \frac{c}{\gcd(a,b)}$。

线性同余方程 – 伏笔回收

如前文所述，线性同余方程 $ax \equiv b \pmod{m}$ 可转化为线性 Diophantus 方程 $ax + my = b$ 的求解。

若 $ax + my = b$ 通解为 $\langle x_0 + km', y_0 + ka' \rangle$ $(k \in \mathbb{Z})$，
则 $ax \equiv b \pmod{m}$ 的解有形式 $x \equiv x_0 \pmod{m'}$。
其中 $m' = \frac{m}{\gcd(a,m)}$。

若 $ax + my = b$ 无解，即 $\gcd(a, m) \nmid b$，则 $ax \equiv b \pmod{m}$ 也无解。
只有当 $\gcd(a, m) \neq 1$ 时才有可能发生无解的情况（回顾线性 Diophantus 方程的解的存在性判定条件）。

有理数取模 – 伏笔回收

如前文所述，为最简分数 $\frac{b}{a}$ 定义对 m 的模时，可以考虑线性同余方程 $ax \equiv b \pmod{m}$ 的解。

假设 $ax \equiv b \pmod{m}$ 有解 $x \equiv x_0 \pmod{m'}$，其中 $m' = \frac{m}{\gcd(a,m)}$。

- 如果 $m' = m$，即 $\gcd(a, m) = 1$，也即 $a \perp m$，
  可以定义 $\frac{b}{a} \bmod_{\mathbb{Q}} m = x_0 \bmod m$。
- 如果 $m' \neq m$，即 $\gcd(a, m) \neq 1$，这是不可能的，因为有解当且仅当 $\gcd(a, m) \mid b$，于是这说明 $\gcd(a, b) \neq 1$，故 $\frac{b}{a}$ 并非最简分数，与前提矛盾。

得出结论：最简分数 $\frac{b}{a}$ 的模有定义当且仅当 $a \perp m$。

有理数取模 – 乘法逆元

如前文所述，最简分数 $\frac{b}{a}$ 的模有定义当且仅当 $a \perp m$。
考虑模的性质：$(a \bmod_{\mathbb{Q}} m) \cdot (b \bmod_{\mathbb{Q}} m) \equiv (a \cdot b) \bmod_{\mathbb{Q}} m$。
于是有 $\frac{b}{a}$ 的模等于 b 的模乘 $\frac{1}{a}$ 的模。
故只需要定义全体与 m 互素的正整数 a 的倒数 $\frac{1}{a}$ 的模即可。
假设 $\frac{1}{a}$ 的模为 x，则 x 将会满足 $ax \equiv 1 \pmod m$，在同余式的运算中扮演消去一侧 a 因子的角色：

$$
\begin{aligned}
(\mathcal{A}) \cdot a &\equiv \mathcal{B} \\
(\mathcal{A}) \cdot a \cdot x &\equiv (\mathcal{B}) \cdot x \\
\mathcal{A} &\equiv (\mathcal{B}) \cdot x
\end{aligned}
$$

我们称满足 $ax \equiv 1 \pmod m$ 的 x 为 a 模 m 的**乘法逆元**。
可以证明模 m 同余的所有 a 的乘法逆元相同，并且扩展 Euclid 算法给出了乘法逆元的一种计算方法。

感谢倾听

5 附录

中英文对照表

- 整除：divide
- 被⋯⋯整除：be divided by
- 倍数：multiple
- 因数（约数）：divisor，factor
- 平凡因数（平凡约数）：trivial divisor，trivial factor
- 整系数线性组合：linear combination with integer coefficients
- 商：quotient
- 余数：remainder
- 素数（质数）：prime number
- 合数：composite number
- 同余：congruent
- 模：modulo
- 模数：modulus
- 公因数（公约数）：common divisor，common factor
- 最大公因数（最大公约数）：greatest common divisor，highest common factor
- 互素（互质）：coprime，relatively prime，mutually prime
- 公倍数：common multiple
- 最小公倍数：least common multiple，smallest common multiple
- 算术基本定理（唯一分解定理）：fundamental theorem of arithmetic，unique factorization theorem
- 标准分解式：canonical representation，standard form
- p 进阶数：p-adic order
- p 进赋值：p-adic valuation
- 素性测试：primality test
- 试除法：trial division
- 素因数分解（分解素因数、质因数分解、素因子分解）：prime factorization
- Euclid（欧几里得）算法：Euclidean algorithm
- Eratosthenes（埃拉托斯特尼）筛法：sieve of Eratosthenes
- Euler（欧拉）筛法：Euler's sieve，sieve of Euler
- Diophantus（丢番图）方程：Diophantine equation
- 线性同余方程：linear congruence equation
- Bézout（裴蜀）定理：Bézout's lemma
- 扩展 Euclid（欧几里得）算法：extended Euclidean algorithm
- 乘法逆元：modular multiplicative inverse

参考文献与致谢

- OI Wiki，https://oi-wiki.org/
- Wikipedia，https://en.wikipedia.org/wiki/Main_Page
- Pr∞fWiki，https://proofwiki.org/wiki/Main_Page
- 具体数学（*Concrete Mathematics: A Foundation for Computer Science*），R. L. Graham, D. E. Knuth, and O. Patashnik
- 数论导引，华罗庚

- 本课件编写时的哔哩哔哩直播间观众