数论基础

本文部分内容引用并改编自小粉兔 (@GitPinkRabbit) 的开源项目：算法竞赛中的数论–系列课件

2025/6/18

目录

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 $a \mid b$，否则称 a 不整除 b，记作 $a \nmid b$。
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 $a \mid b$，否则称 a 不整除 b，记作 $a \nmid b$。
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 – 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**。

带余除法

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

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

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

带余除法

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

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

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

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

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 $\leq$ 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 $= 0$。

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 – 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 $a / m = q$ 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。

在操作数可能为负数或 0 时，需要特别注意。

一些常用公式 – 1

对于 $(a \pm b) \mod c$，等价于 $((a \mod c) \pm (b \mod c)) \mod c$。一个例子：计算斐波那契数列的第 $n(1 \le n \le 10^5)$ 项，由于这个数字可能非常大，所以输出其对 998244353 取模的结果。

```
const int P = 998244353;
int f[N], s[N];
int calc(int n) {
    f[1] = f[2] = 1;
    for(int i = 3; i <= n; i++) {
        f[i] = (f[i - 1] + f[i - 2]) % P;
    }
    return f[n] % P;
}
```

一些常用公式 – 2

有时，将 $(a-b) \mod c$ 的结果使用 $(a \mod c - b \mod c) \mod c$ 的方式计算会出现负数，例如：
$a=14, b=9, c=6$ 时，$(a-b) \mod c = 5 \mod 6 = 5$ ，
但是 $(a \mod c - b \mod c) \mod c = (14 \mod 6 - 9 \mod 6) \mod 6 = (2-3) \mod 6 = -1$ 。
要想保证结果正确，我们可以在做类似计算的时候在先 $+c$ 在 $\mod c$ ，也就是：
$(a \mod c - b \mod c + c) \mod c$ ，可以保证最后的结果一定是数学意义上我们期望的余数。
由于大家目前的数论知识不足以证明这个操作，因此我们在这里不予证明。

一些常用公式 – 3

对于 $(a \times b) \mod c$，等价于 $((a \mod c) \times (b \mod c)) \mod c$。一个例子：计算 $n(1 \leq n \leq 1000)$ 的阶乘，由于这个数字可能非常大，所以输出其对 998244353 取模的结果。

```
const int P = 998244353;
int fac[N];
int calc(int n) {
    fac[1] = 1;
    for(int i = 2; i <= n; i++) {
        fac[i] = (i * fac[i - 1]) % P;
    }
    return fac[n] % P;
}
```

质数的定义

一个数的因数如果只有1和它本身，那么我们称这个数是质数

试除法判断质数 – 1

试除法是判断质数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。

```
bool check(int n) {
    if(n == 1) return false;
    for(int i = 2; i < n; i++) {
        if(n % i == 0) return false;
    }
    return true;
}
```

时间复杂度 $O(n)$

试除法判断质数 – 2

定理：如果 n 拥有非平凡因数，则其最小非平凡正因数 $\leq \sqrt{n}$。

试除法是判断质数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。

```
bool check(int n) {
    if(n == 1) return false;
    for(int i = 2; i * i <= n; i++) {
        if(n % i == 0) return false;
    }
    return true;
}
```

时间复杂度 $O(\sqrt{n})$