

(ex)gcd，逆元与其他

junyu33

2024/09/13

约数与最大公约数

我们在小学三年级的时候就学过带余除法，例如：

$$14 \div 4 = 3 \dots 2$$

或者，如果有人觉得这个除号很难看，我们可以写成：

$$14 = 4 \times 3 + 2$$

将其推广到整数上，我们可以得到：

$$a = b \times q + r$$

其中 a, b, q, r 都是整数， a 是被除数， b 是除数， q 为商 (quotient)， r 是余数 (remainder)， $0 \leq r < b$ 。

另外，我们记商 q 为 $\lfloor \frac{a}{b} \rfloor$ (下取整)，余数 r 为 $a \bmod b$ 。

请记住这几个字母和符号代表的意义，我们在后面的内容中会经常用到。

约数与最大公约数

当 $r = 0$ 时，我们称 b 是 a 的**约数**，或者说 a 可以被 b 整除，记作 $b \mid a$ 。

如果存在两个数 m, n ，有 $x \mid m, x \mid n$ ，那么我们称 x 是 m 和 n 的**公约数**。

如果 x 是 m 和 n 的公约数，且对于任意的公约数 y ，有 $y \leq x$ （或者 $y \mid x$ ，为什么？），那么我们称 x 是 m 和 n 的**最大公约数**，记作 $\gcd(m, n)$ 。

那现在的问题是，如何求两个数的最大公约数呢？

枚举法

最简单的方法就是枚举法，我们可以枚举 m 和 n 的所有约数，然后找到它们的最大公约数。

伪代码如下：

```
int gcd(int m, int n) {  
    int ans = 0;  
    for (int i = 1; i ≤ min(m, n); i++) {  
        if (m % i == 0 && n % i == 0) {  
            ans = i;  
        }  
    }  
    return ans;  
}
```

显然，这个方法的时间复杂度是 $O(\min(m, n))$ ，当 m, n 较大时，这个方法是不可取的。

辗转相除法

假设 $m > n$ ，有性质 $\gcd(m, n) = \gcd(m \bmod n, n)$ ，证明如下：

我们先证 $\gcd(m, n) = \gcd(m - n, n)$ ：

设 $d = \gcd(m, n)$ ，那么 $d \mid m$ ，且 $d \mid n$ ，设 $m = dm'$ ， $n = dn'$ 。

显然，我们有 $\gcd(m', n') = 1$ ，否则我们可以找到 $d' = d \times \gcd(m', n')$ ，使得 d' 是 m, n 的最大公约数。

那么 $m - n = d(m' - n')$ ，显然 $d \mid m - n$ ，所以 d 也是 $m - n$ 和 n 的公约数。

因为 m 与 n 互质，所以 $m - n$ 与 n 互质。按照第四行的结论，可得 d 是 $m - n$ 和 n 的最大公约数。

利用上述结论，只需要进行 $\lfloor \frac{m}{n} \rfloor$ 次操作，就可以得到 $\gcd(m, n) = \gcd(m \bmod n, n)$ 。

但是，这个证明是错误的，所以这个证明有什么问题？

错误的点在于，"因为 m 与 n 互质，所以 $m - n$ 与 n 互质"这个推理用到了我们要证明的结论，所以陷入了循环论证。

我们需要重新开始。

辗转相除法

假设 $m > n$ ，有性质 $\gcd(m, n) = \gcd(m \bmod n, n)$ ，证明如下：

我们先证 $\gcd(m, n) = \gcd(m - n, n)$ ：

设 $d = \gcd(m, n)$ ，那么 $d \mid m$ ，且 $d \mid n$ ，故 $d \mid m - n$ 。

因此， d 是 $m - n$ 和 n 的公约数。

另一方面，设 $d' = \gcd(m - n, n)$ ，那么 $d' \mid m - n$ ，且 $d' \mid n$ 。

因此， $d' \mid ((m - n) + n)$ ，即 $d' \mid m$ 。因此， d' 是 m 和 n 的公约数。

这样，我们可得 $d' \mid d$ 。同理，我们可以得到 $d \mid d'$ ，因此 $d = d'$ ，即 $\gcd(m, n) = \gcd(m - n, n)$ 。

利用上述结论，只需要进行 $\lfloor \frac{m}{n} \rfloor$ 次操作，就可以得到 $\gcd(m, n) = \gcd(m \bmod n, n)$ 。

Quod erat demonstrandum.

辗转相除法

因为 $\gcd(m, n) = \gcd(m \bmod n, n)$ ，而 $m \bmod n < n$ ，又化归为一个更小的问题。

因此，我们可以递归进行辗转相除法，直到某一项为 0，那么另一项就是两个数的最大公约数。

```
int gcd(int m, int n) {  
    if (n == 0) {  
        return m;  
    }  
    return gcd(n, m % n);  
}
```

这个算法的时间复杂度是 $O(\log \min(m, n))$ ，是一个非常高效的算法。

但是，问题来了，我们知道这个算法是对数级别的，那么这个对数的底数究竟是多少呢？

问题：我们假设这个算法的时间复杂度是 $O(\log_x \min(m, n))$ ，那么 x 的值为？

答案是 $\frac{1+\sqrt{5}}{2}$ ，也就是黄金分割比例。

辗转相除法的时间复杂度分析（有点难）

我们需要知道，在什么情况下，辗转相除法会有最坏的时间复杂度。（我们假设 $m > n$ ）

首先，我们显然可以知道，要达到最坏的时间复杂度，必有 $n > \frac{m}{2}$ ，不然可以找到一个更小的 $m' = m - n$ 也能达到同样的效果。

当 n 很大，和 m 很接近的时候？不对，那样 m 和 n 一除，余数就变小了，肯定不行。

同理， n 相对于 m 来说，不能太小（靠近 $\frac{m}{2}$ ）。否则 n 与 $m \bmod n$ 的值会很接近，这样也会很快结束。

当 n 和 m 基本上离倍数关系“很远”的时候？有点可能，但是这个“很远”是什么意思呢？

是不是当 m 与 n 之间的关系跟 n 和 $m \bmod n$ 之间的关系差不多的时候？

有点意思。

辗转相除法的时间复杂度分析（有点难）

是不是当 m 与 n 之间的关系跟 n 和 $m \bmod n$ 之间的关系差不多的时候？

这句话翻译成数学语言就是：

$$x : 1 = 1 : (x - 1) \wedge x > 0$$

解一下这个方程，我们可以得到 $x = \frac{1+\sqrt{5}}{2}$ 。

正是黄金分割比例。

当然，这只是一种非常直观的解释，具体的证明又需要分成两步：

1. 证明辗转相除法在 fibonacci 数列上达到最坏时间复杂度。
2. 证明 fibonacci 数列的通项公式近似于底数为黄金分割比例的指数数列。

辗转相除法与 fibonacci 数列（有点难）

我们可以使用数学归纳法证明，辗转相除法在 fibonacci 数列上达到最坏时间复杂度。

我们的命题是：假设用辗转相除法求自然数 a 和 b （ $a > b > 0$ ）的最大公约数需要 N 步，那么满足这一条件的 a 和 b 的最小值分别是斐波那契数 F_{N+2} 和 F_{N+1} 。

当 $N = 1$ 时， $a = F_3 = 2, b = F_2 = 1$ ，满足条件，此时 a, b 是满足条件的最小值。

假设当 $N = K - 1$ 时， $a = F_{K+1}, b = F_K$ 满足条件。

一个需要 K 步的算法第一步是 $a = q_0b + r_0$ ，第二步是 $b = q_1r_0 + r_1$ 。

根据 $N = K - 1$ 的假设，因为求 $\gcd(b, r_0)$ 需要 $K - 1$ 步，所以 $b \geq F_{K+1}, r_0 \geq F_K$ 。

而 $a = q_0b + r_0 \geq b + r_0 \geq F_{K+1} + F_K = F_{K+2}$

因此这个需要 K 步的算法需要满足 $a \geq F_{K+2}, b \geq F_{K+1}$ 。这样第一部分就证明完了。

1844年，加百利·拉梅发现这个证明，标志着计算复杂性理论的开端。同时，这也是 fibonacci 数列的第一个实际应用。

辗转相除法与 fibonacci 数列（有点难）

接下来我们证明 fibonacci 数列的通项公式近似于底数为黄金分割比例的指数数列。（这话读着可真费劲）

首先我们需要知道 fibonacci 数列的通项公式。我知道有两种方式可以求，第一种是使用特征方程，第二种是使用生成函数，这两种方法都不是很简单，这里我选择直接跳过。

我们换一种思路，如果我们求出相邻的斐波那契数的比值，如果这个比值是无限接近于黄金分割比例，那么我们就可以证明这个结论（无 $\epsilon - \delta$ 版本，不够严谨）。

假设这个极限存在，设其为 L ，即 $L = \lim_{n \rightarrow \infty} \frac{F_{n+1}}{F_n}$ 。根据 fibonacci 数列的递推关系 $F_{n+1} = F_n + F_{n-1}$ ，我们可以写出：

$$\frac{F_{n+1}}{F_n} = \frac{F_n + F_{n-1}}{F_n} = 1 + \frac{F_{n-1}}{F_n}$$

当 n 很大时， $\frac{F_{n-1}}{F_n}$ 也接近 $\frac{1}{L}$ ：

$$L = 1 + \frac{1}{L}$$

解这个方程，我们可以得到 $L = \frac{1+\sqrt{5}}{2}$ ，这样第二部分就证明完了。

乘法逆元

我们在小学五年级便系统学习了与分数运算相关的内容，我们知道除以一个数等于乘一个数的倒数，例如：

$$5 \div 3 = 5 \times \frac{1}{3}$$

或者也可以写成：

$$5 \div 3 = 5 \times 3^{-1}$$

我们就可以认为3和 $\frac{1}{3}$ 互为乘法逆元，因为他们的乘积为乘法的单位元1。

现在我们将目光从有理数域 \mathbb{Q} 转移到有限整数域 \mathbb{Z}_p （ p 为质数）：

我们假设取 $p = 17$ ，那么我们又该如何计算 $5 \div 3$ 呢？

乘法逆元

首先，我们不能直接套用“除以一个数等于乘一个数的倒数”这句话，因为 $\frac{1}{3}$ 这个数不在 \mathbb{Z}_{17} 中。

但是， 3^{-1} 这个数显然还是存在的，因为有域的基本性质，我们肯定能在 \mathbb{Z}_{17} 中找到一个数，使得 $3 \times 3^{-1} = 1$ 。

那么，我们该如何找到这个数呢？

当然，最简单的方法还是枚举 \mathbb{Z}_{17} 中的所有元素，来逐一验证他们的乘积 $\bmod 17$ ，结果是否为1，代码如下：

```
int inv(int x) {
    for (int i = 0; i < 17; i++) {
        if (x * i % 17 == 1) {
            return i;
        }
    }
    return -1;
}
```

当然，这样的计算效率还是太低了。

扩展欧几里得算法

我们来重述这个问题，为了求3在 \mathbb{Z}_{17} 中的乘法逆元，我们需要找到一个数 x ，使得 $3 \times x \equiv 1 \pmod{17}$ 。

我们对这个方程进行变形，得到：

$$3x - 17y = 1 \wedge x, y \in \{0, 1, 2, \dots, 16\}$$

我们通常会使用扩展欧几里得算法来解决这个问题，具体的算法如下：

1. 先使用辗转相除法求出 $\gcd(3, 17)$ 。
2. 再通过倒推的方法，求出 x 和 y 。

扩展欧几里得算法

我们先使用辗转相除法求出 $\gcd(3, 17)$ 。

$$17 = 5 \times 3 + 2$$

$$3 = 1 \times 2 + 1$$

$$2 = 2 \times 1 + 0$$

递归完成，我们得到 $\gcd(3, 17) = 1$ 。

接下来，我们使用倒推的方法，求出 x 和 y 。

$$\begin{aligned} 1 &= 3 - 1 \times 2 \\ &= 3 - 1 \times (17 - 5 \times 3) = -1 \times 17 + 6 \times 3 \end{aligned}$$

这样，我们就找到了满足条件的 $(x, y) = (6, 1)$ ，使得

$$3x - 17y = 1$$

我们也就找到了3在 \mathbb{Z}_{17} 中的乘法逆元，即 $3^{-1} = 6$ 。那么 $5 \times 3^{-1} = 5 \times 6 = 30 \equiv 13 \pmod{17}$ 。

扩展欧几里得算法

有了以上的理论基础，我们便可以编写一个在 \mathbb{Z}_p 上求乘法逆元的函数：

```
int exgcd(int a, int b, int &x, int &y) { // C++ 左值引用，可以修改 x 和 y 的值
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    int d = exgcd(b, a % b, x, y);
    // (x, y) → (y, x - a / b * y)
    int z = x;
    x = y;
    y = z - a / b * y;
    return d;
}

int inv(int a, int p) { // 求 a 在  $\mathbb{Z}_p$  上的乘法逆元
    int x, y;
    int d = exgcd(a, p, x, y);
    return (x % p + p) % p; // 保证返回值在  $[0, p)$  之间
}
```

其时间复杂度与辗转相除法相同，为 $O(\log_{\varphi} a)$ 。

扩展欧几里得算法代码分析

为了方面初次接触的同学理解，我对代码进行了简单的插桩，得到了如下递归结果：

```
exgcd(3, 17, 30170, 1651076199)
  exgcd(17, 3, 30170, 1651076199)
    exgcd(3, 2, 30170, 1651076199)
      exgcd(2, 1, 30170, 1651076199)
        exgcd(1, 0, 30170, 1651076199)
          exgcd(1, 0, 1, 0) = 1
        exgcd(2, 1, 0, 1) = 1
      exgcd(3, 2, 1, -1) = 1
    exgcd(17, 3, -1, 6) = 1
  exgcd(3, 17, 6, -1) = 1
6
```

首先算法的前半段跟普通的辗转相除法是一样的，没有什么特别需要说明的地方。

重点是后面的回溯过程，我们来看看这个过程如何跟之前我们的手算过程对应起来。

```
exgcd(1, 0, 1, 0) = 1
exgcd(2, 1, 0, 1) = 1
exgcd(3, 2, 1, -1) = 1
exgcd(17, 3, -1, 6) = 1
exgcd(3, 17, 6, -1) = 1
```

- 第一步，我们赋值 $(x, y) \rightarrow (1, 0)$ ，即 $1 = 1 \times 1 + 0 \times 0$ 。
- 第二步，我们将 $(x, y) \rightarrow (y, x - \lfloor \frac{a}{b} \rfloor \times y)$ ，即 $(1, 0) \rightarrow (0, 1)$ 。

$$1 = 1 \times 1 + (2 - \lfloor \frac{2}{1} \rfloor \times 1) \times 1 = 2 \times 0 + 1 \times 1$$

- 第三步，我们将 $(x, y) \rightarrow (y, x - \lfloor \frac{a}{b} \rfloor \times y)$ ，即 $(0, 1) \rightarrow (1, -1)$ 。

$$1 = 2 \times 0 + (3 - \lfloor \frac{3}{2} \rfloor \times 2) \times 1 = 3 \times 1 + 2 \times (-1)$$

- 第四步，我们将 $(x, y) \rightarrow (y, x - \lfloor \frac{a}{b} \rfloor \times y)$ ，即 $(1, -1) \rightarrow (-1, 6)$ 。

$$1 = 3 \times 1 + (17 - \lfloor \frac{17}{3} \rfloor \times 3) \times (-1) = 17 \times (-1) + 3 \times 6$$

- 第五步后，我们得到了满足条件的 $(x, y) = (6, -1)$ ，即 $(-1, 6) \rightarrow (6, -1)$ 。