[toc]

最大公约数

最大公约数即为 Greatest Common Divisor，常缩写为 gcd。

在 素数 一节中，我们已经介绍了约数的概念。

一组数的公约数，是指同时是这组数中每一个数的约数的数。而最大公约数，则是指所有公约数里面最大的一个。

那么如何求最大公约数呢？我们先考虑两个数的情况。

欧几里得算法

如果我们已知两个数 a 和 b，如何求出二者的最大公约数呢？

不妨设 $a > b$

我们发现如果 b 是 a 的约数，那么 b 就是二者的最大公约数。 下面讨论不能整除的情况，即 $a = b \times q + r$，其中 $r < b$。

我们通过证明可以得到 $\gcd(a,b)=\gcd(b,a \bmod b)$，过程如下：

设 $a=bk+c$，显然有 $c=a \bmod b$。设 $d \mid a,~d \mid b$，则 $c=a-bk, \frac{c}{d}=\frac{a}{d}-\frac{b}{d}k$。

由右边的式子可知 $\frac{c}{d}$ 为整数，即 $d \mid c$ 所以对于 a,b 的公约数，它也会是 $a \bmod b$ 的公约数。

反过来也需要证明：

设 $d \mid b,~\mid (a \bmod b)$，我们还是可以像之前一样得到以下式子 $\frac{a\bmod b}{d}=\frac{a}{d}-\frac{b}{d}k,~\frac{a\bmod b}{d}+\frac{b}{d}k=\frac{a}{d}$。

因为左边式子显然为整数，所以 $\frac{a}{d}$ 也为整数，即 $d \mid a$，所以 $b,a\bmod b$ 的公约数也是 a,b 的公约数。

既然两式公约数都是相同的，那么最大公约数也会相同。

所以得到式子 $\gcd(a,b)=\gcd(b,a\bmod b)$

既然得到了 $\gcd(a, b) = \gcd(b, r)$，这里两个数的大小是不会增大的，那么我们也就得到了关于两个数的最大公约数的一个递归求法。

```
int gcd(int a, int b) {
  if (b == 0) return a;
  return gcd(b, a % b);
}
```

递归至 `b==0`（即上一步的 `a%b==0`) 的情况再返回值即可。

上述算法被称作欧几里得算法（Euclidean algorithm）。

如果两个数 a 和 b 满足 $\gcd(a, b) = 1$，我们称 a 和 b 互质。

欧几里得算法的时间效率如何呢？下面我们证明，欧几里得算法的时间复杂度为 $O(\log n)$。

当我们求 $\gcd(a,b)$ 的时候，会遇到两种情况：

- $a < b$，这时候 $\gcd(a,b)=\gcd(b,a)$；
- $a \geq b$，这时候 $\gcd(a,b)=\gcd(b,a \bmod b)$，而对 a 取模会让 a 至少折半。这意味着这一过程最多发生 $O(\log n)$ 次。

第一种情况发生后一定会发生第二种情况，因此第一种情况的发生次数一定 **不多于** 第二种情况的发生次数。

从而我们最多递归 $O(\log n)$ 次就可以得出结果。

事实上，假如我们试着用欧几里得算法去求 斐波那契数列 相邻两项的最大公约数，会让该算法达到最坏复杂度。

多个数的最大公约数

那怎么求多个数的最大公约数呢？显然答案一定是每个数的约数，那么也一定是每相邻两个数的约数。我们采用归纳法，可以证明，每次取出两个数求出答案后再放回去，不会对所需要的答案造成影响。

最小公倍数

接下来我们介绍如何求解最小公倍数（Least Common Multiple, LCM）。

两个数的

首先我们介绍这样一个定理——算术基本定理：

> 每一个正整数都可以表示成若干整数的乘积，这种分解方式在忽略排列次序的条件下是唯一的。

用数学公式来表示就是 $x = p_1^{k_1}p_2^{k_2} \cdots p_s^{k_s}$

设 $a = p_1^{k_{a_1}}p_2^{k_{a_2}} \cdots p_s^{k_{a_s}}$，$b = p_1^{k_{b_1}}p_2^{k_{b_2}} \cdots p_s^{k_{b_s}}$

我们发现，对于 a 和 b 的情况，二者的最大公约数等于

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

最小公倍数等于

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

由于 $k_a + k_b = \max(k_a, k_b) + \min(k_a, k_b)$

所以得到结论是 $\gcd(a, b) \times \operatorname{lcm}(a, b) = a \times b$

要求两个数的最小公倍数，先求出最大公约数即可。

多个数的

可以发现，当我们求出两个数的 $\gcd$ 时，求最小公倍数是 $O(1)$ 的复杂度。那么对于多个数，我们其实没有必要求一个共同的最大公约数再去处理，最直接的方法就是，当我们算出两个数的 $\gcd$，或许在求多个数的 $\gcd$ 时候，我们将它放入序列对后面的数继续求解，那么，我们转换一下，直接将最小公倍数放入序列即可。

扩展欧几里得算法

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

证明

设

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

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

由欧几里得定理可知：$\gcd(a,b)=\gcd(b,a\bmod b)$

所以 $ax_1+by_1=bx_2+(a\bmod b)y_2$

又因为 $a\bmod b=a-(\lfloor\frac{a}{b}\rfloor\times b)$

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

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

因为 $a=a,b=b$，所以 $x_1=y_2,y_1=x_2-\lfloor\frac{a}{b}\rfloor y_2$

将 x_2,y_2 不断代入递归求解直至 $\gcd$（最大公约数，下同）为 `0` 递归 `x=1,y=0` 回去求解。

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

函数返回的值为 $\gcd$，在这个过程中计算 x,y 即可。

迭代法编写拓展欧几里得算法

因为迭代的方法避免了递归，所以代码运行速度将比递归代码快一点。

```
int gcd(int a, int b, int& x, int& y) {
  x = 1, y = 0;
  int x1 = 0, y1 = 1, a1 = a, b1 = b;
  while (b1) {
    int q = a1 / b1;
    tie(x, x1) = make_tuple(x1, x - q * x1);
    tie(y, y1) = make_tuple(y1, y - q * y1);
    tie(a1, b1) = make_tuple(b1, a1 - q * b1);
  }
  return a1;
}
```

如果你仔细观察 a_1 和 b_1，你会发现，他们在迭代版本的欧几里德算法中取值完全相同，并且以下公式无论何时（在 while 循环之前和每次迭代结束时）都是成立的：$x \cdot a +y \cdot b =a_1$ 和 $x_1 \cdot a +y_1 \cdot b= b_1$。因此，该算法肯定能正确计算出 $\gcd$。

最后我们知道 a_1 就是要求的 $\gcd$，有 $x \cdot a +y \cdot b =g$。

例题

- 10104 - Euclid Problem
- GYM - (J) once upon a time
- UVA - 12775 - Gift Dilemma