# 第 1 章　数论基础

## 1.1　算法原理

数论主要研究的是整数的性质，许多加密算法都用到了数论知识。本章介绍一些在密码学中应用较为广泛的基础数论算法，包括厄拉多塞筛算法、欧几里得算法、快速幂取模算法、中国剩余定理、Miller-Rabin 素性检测算法。

### 1.1.1　厄拉多塞筛算法

厄拉多塞筛算法（Eratosthenes Sieve）是一种求素数的方法，由古希腊数学家厄拉多塞提出。它的原理是给定一个数字 $N$，从 2 开始依次将 $\sqrt{N}$ 以内的素数的倍数标记为合数，标记完成后剩余未被标记的数为素数（从 2 开始）。如此可省去检查每个数的步骤，使筛选素数的过程更加简单。厄拉多塞筛算法流程图如图 1-1 所示，具体步骤如下：

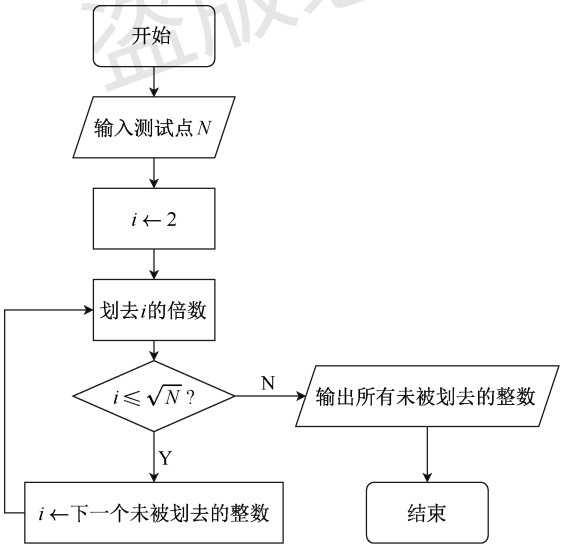


图 1-1　厄拉多塞筛算法流程图

（1）读取输入的数字 $N$，将 2～$N$ 的所有整数记录在表中；

（2）从 2 开始，划去表中所有 2 的倍数；

（3）由小到大寻找表中下一个未被划去的整数，再划去表中所有该整数的倍数；

（4）重复步骤（3），直到找到的整数大于 $\sqrt{N}$ 为止；

（5）表中所有未被划去的整数均为素数，输出所有未被划去的整数。

### 1.1.2 欧几里得算法

欧几里得算法（Euclid's Algorithm）又称辗转相除法。古希腊数学家欧几里得在其著作 *The Elements* 中最早描述了这种算法，所以被命名为欧几里得算法。欧几里得算法利用计算公式 $\gcd(a,b)=\gcd(b,a \bmod b)$ 求两个整数 $a$ 和 $b$ 的最大公约数，其中 $\gcd(x,y)$ 代表 $x$ 、 $y$ 的最大公约数。本书中关注非负整数 $a$ 和 $b$ 的最大公约数，欧几里得算法流程图如图 1-2 所示。欧几里得算法具体步骤如下：

（1）使用带余除法， $b$ 除 $a$ 得到余数 $r$ ；

（2）若 $r>0$ ，则用 $b$ 代替 $a$ ，用 $r$ 代替 $b$ ，重复步骤（1）；

（3） $b$ 的值就是最大公约数 $d$ 。

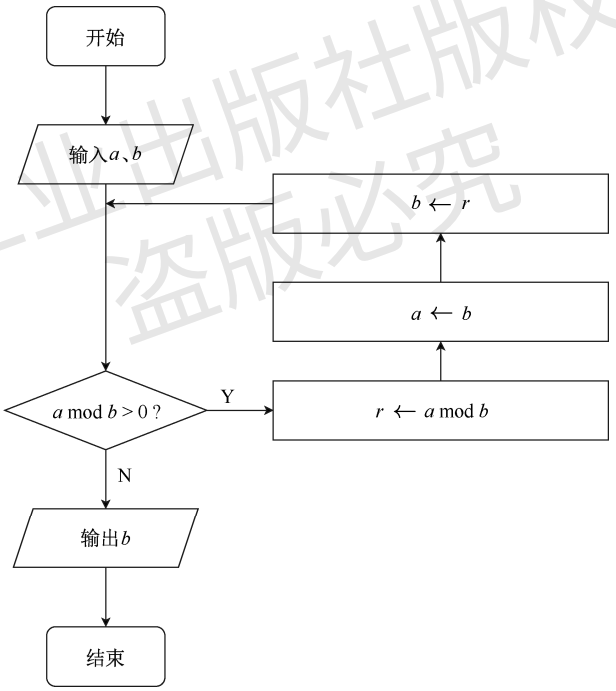


图 1-2 欧几里得算法流程图

扩展欧几里得算法（Extended Euclid's Algorithm）是对欧几里得算法的扩展，基于以下定理：对任意两个整数 $a$ 、 $b$ ，必然存在整数 $x$ 、 $y$ ，使得 $ax+by=\gcd(a,b)$ 成立，求出整数解 $x$ 、 $y$ ，则可以得到 $\gcd(a,b)$ 。同样，本书中关注非负整数 $a$ 、 $b$ 。扩展欧几里得算法可以分为递归版本和非递归版本。

在递归版本中，扩展欧几里得算法的主要思想与欧几里得算法类似，根据公式 $\gcd(a,b)=\gcd(b,a \bmod b)$ ，有以下等式成立：

$$ax_1+by_1=bx_2+(a \bmod b)y_2=bx_2+\left(a-\lfloor a/b\rfloor b\right)y_2=ay_2+b\left(x_2-\lfloor a/b\rfloor y_2\right)$$

可得 $x_1=y_2$，$y_1=x_2-\lfloor a/b\rfloor y_2$。因此，递归版本将求解整数 $x$、$y$ 使得 $ax+by=\gcd(a,b)$ 成立转换为求解 $x'$、$y'$ 使得 $bx'+(a \bmod b)y'=\gcd(b,a \bmod b)$ 成立，直至 $b$ 整除 $a$ 为止。

在非递归版本中，扩展欧几里得算法的主要思想是找到 $x$、$y$ 的递推关系，并利用该递推关系进行计算，具体步骤为：

（1）根据带余除法，假设每个步骤 $i$（$i=1,2,\cdots,n$）都可找到对应的 $x_i$ 和 $y_i$，则可列式：

$$a=q_1b+r_1,\quad r_1=ax_1+by_1$$
$$b=q_2r_1+r_2,\quad r_2=ax_2+by_2$$
$$r_1=q_3r_2+r_3,\quad r_3=ax_3+by_3$$
$$\vdots$$
$$r_{n-2}=q_nr_{n-1}+r_n,\quad r_n=ax_n+by_n$$
$$r_{n-1}=q_{n+1}r_n+0$$

（2）移项，得到：

$$r_i=r_{i-2}-r_{i-1}q_i$$

（3）以此类推，将 $r_{i-2}=ax_{i-2}+by_{i-2}$ 和 $r_{i-1}=ax_{i-1}+by_{i-1}$ 代入 $r_i=r_{i-2}-r_{i-1}q_i$，可得：

$$r_i=a\left(x_{i-2}-q_ix_{i-1}\right)+b\left(y_{i-2}-q_iy_{i-1}\right)$$

（4）由 $r_i=ax_i+by_i$，得：

$$x_i=x_{i-2}-q_ix_{i-1},\quad y_i=y_{i-2}-q_iy_{i-1}$$

（5）依次递推，直到 $x_0=0$，$y_0=1$；$x_{-1}=1$，$y_{-1}=0$。

（6）将中间式子迭代即可得到 $x_n$ 和 $y_n$，即所求的 $x$ 和 $y$；同时可得 $a$ 和 $b$ 的最大公约数 $d=r_n=ax_n+by_n$。

### 1.1.3　快速幂取模算法

观察到，对于 $b^e \bmod p$，其中 $e$ 为 $n$ 比特长的非负整数，有以下公式成立：

$$b^e \bmod p=b^{\sum_{i=0}^{n-1}2^ie_i} \bmod p=\left(\prod_{i=0}^{n-1}b^{2^ie_i} \bmod p\right) \bmod p$$

因此，若要计算 $b^e \bmod p$，可以先计算 $b^2 \bmod p$，$b^4 \bmod p$，$b^8 \bmod p$，…，再将 $e$ 的二进制表示中等于 1 的比特所对应的 $b$ 的幂累乘，便可得到结果。快速幂取模算法流程图如图 1-3 所示，具体步骤如下：

（1）令结果 $r$ 的初始值为 1；

（2）考察 $e$ 的最低比特，若最低比特为 1，则当前结果 $r$ 乘以 $b$ 模 $p$；若最低比特为 0，则当前结果 $r$ 不变；

（3）将 $e$ 右移 1 比特；

（4）将 $b$ 变为 $b^2 \bmod p$；

（5）重复步骤（2）、步骤（3）和步骤（4），直至 $e=0$，当前结果 $r$ 即最终结果。

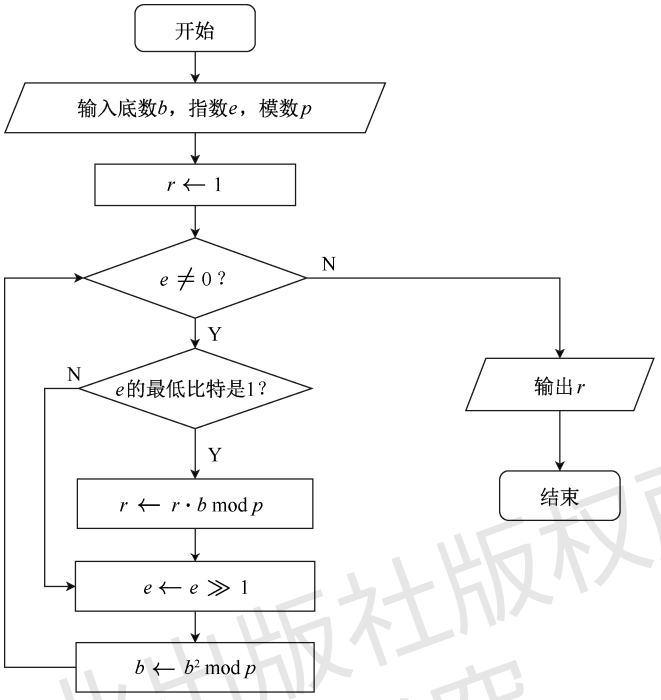


图 1-3　快速幂取模算法流程图



## 1.1.4　中国剩余定理算法

中国剩余定理（Chinese Remainder Theorem，CRT）即孙子定理，是中国古代求解一次同余式组的方法，又称中国余数定理。在《孙子算经》中“物不知数”的问题原文如下：有物不知其数，三三数之剩二，五五数之剩三，七七数之剩二。问物几何？如果用现代语言描述，则中国剩余定理是一种求解以下一元线性同余方程组的算法，其流程图如图 1-4 所示。

$$\begin{cases} x \equiv a_1 \pmod{m_1} \\ x \equiv a_2 \pmod{m_2} \\ \quad\vdots \\ x \equiv a_n \pmod{m_n} \end{cases} (S)$$

假设整数 $m_1, m_2, \cdots, m_n$ 两两互素，则对任意的整数 $a_1, a_2, \cdots, a_n$，方程组有解，并且通解可以用如下方式构造得到：

（1）设 $M = m_1 m_2 \cdots m_n = \prod_{i=1}^{n} m_i$ 是整数 $m_1, m_2, \cdots, m_n$ 的乘积，并设 $M_i = M / m_i$（$i = 1, 2, \cdots, n$）是除 $m_i$ 外的 $n-1$ 个数的乘积；

（2）设 $M_i' = M_i^{-1} \bmod m_i$ 是 $M_i$ 模 $m_i$ 意义下的逆元，即 $M_i' M_i \equiv 1 \pmod{m_i}$，$i = 1, 2, \cdots, n$；

（3）方程组$(S)$的通解形式为：

$$x = a_1M_1'M_1 + a_2M_2'M_2 + \cdots + a_nM_n'M_n + kM = kM + \sum_{i=1}^{n} a_iM_i'M_i,\quad k \in \mathbb{Z}$$

在模 $M$ 的意义下，方程组$(S)$只有一个解 $x = \left(\sum_{i=1}^{n} a_iM_i'M_i\right) \bmod M$ 。

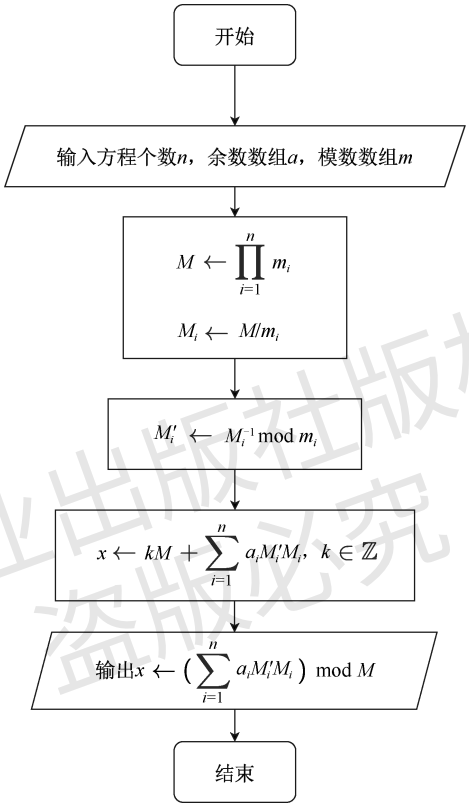


图 1-4　中国剩余定理流程图

## 1.1.5　Miller-Rabin 素性检测算法

首先给出两个定理：费马小定理和二次探测定理。

费马小定理可描述为，设 $p$ 是素数，$a$ 为整数，且 $\gcd(a,p)=1$，则 $a^{p-1} \equiv 1 \pmod{p}$。

二次探测定理可描述为，如果 $p$ 是一个素数，且 $0<x<p$，且方程 $x^2 \equiv 1 \pmod{p}$ 成立，那么 $x=1$ 或 $x=p-1$。

Miller-Rabin 素性检测是基于以上两个定理的随机化算法，用于判断一个数是合数还是素数。判断 $n$ 是否为素数的具体算法过程如下：

（1）令 $n-1=2^k q$，其中 $k>0$，$q$ 为奇数，随机选取整数 $a$，$1<a<n-1$；

（2）若 $a^q \equiv 1 \pmod{n}$，那么 $n$ 可能是素数；

（3）取整数 $i$，$0 \leqslant i < k$，若存在 $i$，使得 $a^{2^i q} \equiv n-1 \pmod{n}$，那么 $n$ 可能是素数；否则，$n$ 为合数。

其中，$a$ 的取值范围为 $[2, n-2]$，而不是 $[1, n-1]$。原因在于，当 $a$ 取 1 或 $n-1$ 时，无论 $n$ 的取值如何，算法总是输出 $n$ 可能为素数，产生了无意义的检测轮次，具体分析如下：

（1）当 $a=1$ 时，有 $a^q \equiv 1^q \equiv 1 \pmod{n}$，因此 $\forall n$，有 $a^q \bmod n = 1$ 成立；

（2）当 $a=n-1$ 时，$\forall i>0$，有 $a^{2^i q} \equiv (n-1)^{2^i q} \equiv (-1)^q \equiv n-1 \pmod{n}$。

因此，当 $a$ 取 1 或 $n-1$ 时，会产生无意义的检测轮次，增加误报率。

由以上分析可知，素数一定通过检测，不通过检测的必为合数，通过检测的可能是素数，这就是 Miller-Rabin 素性检测。Miller-Rabin 素性检测算法流程图如图 1-5 所示。

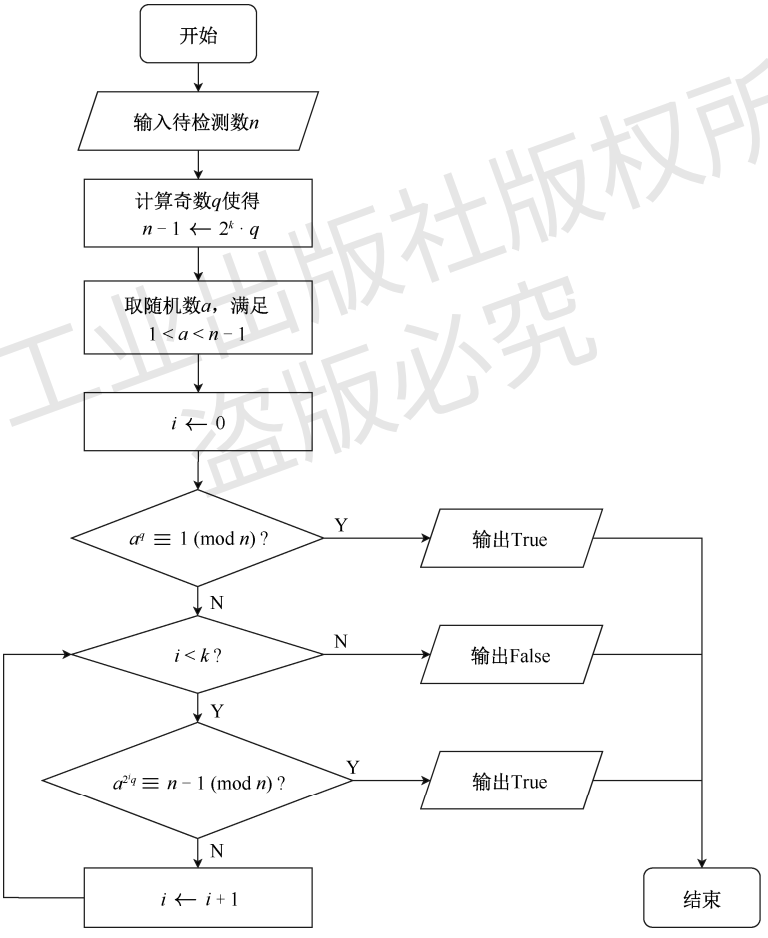


图 1-5　Miller-Rabin 素性检测算法流程图

## 1.2　算法伪代码

本节介绍上述算法的伪代码描述。伪代码清单如表 1-1 所示。

表 1-1　伪代码清单

| 算 法 序 号 | 算　　法 | 算　法　名 |
|---|---|---|
| 1.2.1.1 | 厄拉多塞筛算法 | eratosthenes |
| 1.2.2.1 | 欧几里得算法 | gcd |
| 1.2.2.2 | 扩展欧几里得算法 | exgcd |
| 1.2.3.1 | 快速幂取模算法 | quick_pow_mod |
| 1.2.4.1 | 中国剩余定理算法 | crt |
| 1.2.5.1 | Miller-Rabin 素性检测算法 | mr_test |

## 1.2.1　厄拉多塞筛算法伪代码

算法的输入为一个正整数 $n$，输出为数组 primes，代表小于或等于 $n$ 的所有素数。算法伪代码如下：

**算法 1.2.1.1　eratosthenes$(n)$**

```
// 输入：正整数 n
// 输出：数组 primes
flag[1,2,⋯,n] ← {0,1,⋯,1}
for i ← 2 to ⌊√n − 1⌋ do
    for j ← 2 to ⌊n / i⌋ do
        flag[i · j] ← 0
for i ← 2 to n do
    if flag[i] ≠ 0 then
        将 i 添加至 primes 中
return primes
```

## 1.2.2　欧几里得算法伪代码

算法的输入为两个非负整数 $a$、$b$，输出为其最大公约数 $d$。算法伪代码如下：

**算法 1.2.2.1　gcd$(a,b)$**

```
// 输入：非负整数 a、b
// 输出：最大公约数 d
if b = 0 then
    return a
else
    return gcd(b, a mod b)
```

扩展欧几里得算法的输入为两个非负整数 $a$、$b$，输出为其最大公约数 $d$ 和 $x$、$y$，满

足 $ax+by=d$ 。算法伪代码如下：

**算法 1.2.2.2** $\mathbf{exgcd}(\boldsymbol{a},\boldsymbol{b})$

```
// 输入：非负整数 a 和 b
// 输出：三元组 (d,x,y)
if b = 0 then
    return (a,1,0)
else
    (d,x',y') ← exgcd(b, a mod b)
    x ← y'
    y ← x' − y' · ⌊a / b⌋
    return (d,x,y)
```

## 1.2.3 快速幂取模算法伪代码

算法的输入为整数 $b$ 、 $e$ 、 $p$ ，其中 $b$ 代表底数， $e$ 代表指数， $p$ 代表模数；输出为整数 $r$ ，满足 $r=b^e \bmod p$ 。算法伪代码如下：

**算法 1.2.3.1** $\mathbf{quick\_pow\_mod}(\boldsymbol{b},\boldsymbol{e},\boldsymbol{p})$

```
// 输入：整数 b 、 e 、 p
// 输出：整数 r
r ← 1
while e ≠ 0 do
    if e ≡ 1 (mod 2) then
        r ← r · b mod p
    e ← e ≫ 1
    b ← b^2 mod p
return r
```

## 1.2.4 中国剩余定理算法伪代码

算法的输入为长度为 $n$ 的余数数组 $a$ 和模数数组 $m$ ，代表一共有 $n$ 个同余方程；输出为同余方程组的解 result，算法在实现过程中调用了扩展欧几里得算法 exgcd 。算法伪代码如下：

**算法 1.2.4.1** $\mathbf{crt}(\boldsymbol{a},\boldsymbol{m})$

```
// 输入：余数数组 a[0,1,⋯,n−1] 、模数数组 m[0,1,⋯,n−1]
// 输出：整数 result
result ← 0
```

$M \leftarrow \prod_{i=0}^{n-1} m[i]$

**for** $i \leftarrow 0$ **to** $n-1$ **do**

$M_i \leftarrow M / m[i]$

$M_i' \leftarrow M_i^{-1} \bmod m[i]$

result $\leftarrow$ result $+ a[i] \cdot M_i \cdot M_i'$

**return** $(\text{result} \bmod M)$

### 1.2.5　Miller-Rabin 素性检测算法伪代码

算法的输入为正整数 $n$ 和检测次数 $k$，如果 $n$ 不为素数，则输出 False，否则输出 True。算法在实现过程中调用了快速幂取模算法 quick _ pow _ mod 。算法伪代码如下：

**算法 1.2.5.1**　**mr _ test**$(n,k)$

// 输入：正整数 $n$ 、检测次数 $k$

// 输出：是否为素数的结果 ret

**if** $n=2$ **or** $n=3$ **then**

**return** True

ret $\leftarrow$ False

**for** $i \leftarrow 0$ **to** $k$ **do**

$a \leftarrow_R [2, n-2]$

计算整数 $k$ 和 $q$，使得 $2^k \cdot q = n-1$

**if** $a^q \equiv 1 \pmod n$ **then**

ret $\leftarrow$ ret $\vee$ True

**for** $j \leftarrow 0$ **to** $k-1$ **do**

**if** $a^{2^j \cdot q} \equiv n-1 \pmod n$ **then**

ret $\leftarrow$ ret $\vee$ True

**else**

ret $\leftarrow$ ret $\vee$ False

**return** ret

## 1.3　算法实现与测试

针对厄拉多塞筛算法、欧几里得算法、扩展欧几里得算法、快速幂取模算法、中国剩余定理算法、Miller-Rabin 素性检测算法，本节给出使用 Python（版本大于 3.9）实现的源代码及相应的测试数据。源代码清单如表 1-2 所示。

表 1-2　源代码清单

| 文　件　名 | 包 含 算 法 |
|---|---|
| eratosthenes.py | 厄拉多塞筛算法 |
| gcd.py | 欧几里得算法 |
| exgcd.py | 扩展欧几里得算法 |
| quick_pow_mod.py | 快速幂取模算法 |
| crt.py | 中国剩余定理算法 |
| mr_test.py | Miller-Rabin 素性检测算法 |

### 1.3.1　厄拉多塞筛算法实现与测试

程序输入为 $N$，即求 2～$N$ 之间的素数。以 $N=10$ 为例，给出一组中间数据，如表 1-3 所示。

表 1-3　厄拉多塞筛算法中间数据

| 轮　　数 | 划 去 数 字 |
|---|---|
| 第 1 轮 | 4、6、8、10 |
| 第 2 轮 | 6、9 |
| 得到最终素数 | 2、3、5、7 |

取 $N$ 为 100，则输出素数结果为：2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97。

取 $N$ 为 200，则输出素数结果为：2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101 103 107 109 113 127 131 137 139 149 151 157 163 167 173 179 181 191 193 197 199。

### 1.3.2　欧几里得算法实现与测试

欧几里得算法的结果用 $\gcd(a,b)$ 表示，程序输入为 $a$ 和 $b$，即求 $a$ 和 $b$ 的最大公约数。以 $\gcd(710,310)$ 为例，给出一组中间数据，如表 1-4 所示。

表 1-4　欧几里得算法中间数据

| 轮　　数 | $a$ | $b$ |
|---|---|---|
| 1 | 710 | 310 |
| 2 | 310 | 90 |
| 3 | 90 | 40 |
| 4 | 40 | 10 |
| 5 | 10 | 0 |

此时，$\gcd(710,310)=\gcd(10,0)=10$。

扩展欧几里得算法的结果用 $\text{exgcd}(a,b)$ 表示，源代码实现了递归版本的扩展欧几里得算法，程序输入为 $a$ 和 $b$，即求 $a$ 和 $b$ 的最大公约数。以 $\text{exgcd}(7,5)$ 为例，给出一组中间数据，如表 1-5 所示。

表 1-5　扩展欧几里得算法中间数据 1

| 递 归 深 度 | $d$ | $x'$ | $y'$ | $x=y'$ | $y=x'-y'\cdot\lfloor a/b\rfloor$ |
|---|---|---|---|---|---|
| 3 | 1 | 1 | 0 | 0 | 1 |
| 2 | 1 | 0 | 1 | 1 | −2 |
| 1 | 1 | 1 | −2 | −2 | 3 |

则结果为 $1=(-2)\times 7+3\times 5$，即 $\text{exgcd}(7,5)=(1,-2,3)$。

再以 $\text{exgcd}(1769,551)$ 为例，给出一组中间数据，如表 1-6 所示。

表 1-6　扩展欧几里得算法中间数据 2

| 递 归 深 度 | $d$ | $x'$ | $y'$ | $x=y'$ | $y=x'-y'\cdot\lfloor a/b\rfloor$ |
|---|---|---|---|---|---|
| 4 | 29 | 1 | 0 | 0 | 1 |
| 3 | 29 | 0 | 1 | 1 | −1 |
| 2 | 29 | 1 | −1 | −1 | 5 |
| 1 | 29 | −1 | 5 | 5 | −16 |

则结果为 $29=5\times 1769+(-16)\times 551$，即 $\text{exgcd}(1769,551)=(29,5,-16)$。

下面给出几组测试数据供读者参考，如表 1-7 所示。

表 1-7　扩展欧几里得算法测试数据

| $a$ | $b$ | $x$ | $y$ | $\gcd(a,b)$ |
|---|---|---|---|---|
| 31 | 13 | −5 | 12 | 1 |
| 246150272351567308<br>665870425694491242<br>6065172925575 | 172087657754277021<br>481119930882347652<br>8929542231719 | −279503883787274514253902210297369853780943628056 | 3997959994074502645700<br>9085475842906511784400<br>5679 | 1 |
| 137096164691449488<br>835122291235023051<br>763859318102840889<br>067550902384318989<br>727089044391788984<br>680217107984018759<br>866571252110844726<br>214995953712543463<br>90738382042 | 192350399949876251<br>675909634808997772<br>559337752383120440<br>971227732556475302<br>768063176360267276<br>798008253704593216<br>177248715154421474<br>324209512570378231<br>41069640181 | −1076045803680437575165069317517720056816012968550552297497847522201869587140865136718292159497811949579872126471694232065112371318866682108000433819750738 | 7669427916726892264502<br>0091944585685325849075<br>4050379704483194087100<br>1157002257559190662110<br>2754048417886982308492<br>9299459228617007521725<br>575705884841982026337 | 1 |

## 1.3.3　快速幂取模算法实现与测试

快速幂取模算法 quick_pow_mod 以底数 $b$、指数 $e$ 和模数 $p$ 为输入，输出为 $b^e \bmod p$。

以 $\text{quick\_pow\_mod}(7,19,13)$ 为例，首先将指数 19 化为二进制数，为 $(10011)_2$，然后给出一组中间数据，如表 1-8 所示。

表 1-8　快速幂取模算法中间数据 1

| $i$ | $e_i$ | $b^{2^i} \bmod p$ | $\prod_{j=0}^{i} b^{2^j e_j} \bmod p$ |
|---|---|---|---|
| 0 | 1 | 7 | 7 |
| 1 | 1 | 10 | 5 |
| 2 | 0 | 9 | 5 |
| 3 | 0 | 3 | 5 |
| 4 | 1 | 9 | 6 |

因此，$7^{19} \bmod 13 = 6$。

下面再给出一组中间数据供读者参考，如表 1-9 所示。这里底数 $b = 713945793857\,93847593287459$，指数 $e = 65537$，模数 $p = 27548927592857298457294572950275 9$。指数 65537 化为二进制数为 $(1000000000000000 1)_2$。

表 1-9　快速幂取模算法中间数据 2

| $e_i$ | $\prod_{j=0}^{i} b^{2^j e_j} \bmod p$ |
|---|---|
| 1 | 713945793857938475932874 59 |
| 0 | 713945793857938475932874 59 |
| 0…0（14 个 0） | 713945793857938475932874 59 |
| 1 | 177913638500253600517830301630694 |

除上述两组中间数据外，再给出如表 1-10 所示的两组快速幂取模算法测试数据供读者参考。

表 1-10　快速幂取模算法测试数据

| $b$ | $e$ | $p$ | $b^e \bmod p$ |
|---|---|---|---|
| 5 | 10003 | 31 | 5 |
| 14944626594292900<br>47815067355171411<br>187560751791530 | 65537 | 22688387113047243<br>04304396119509416<br>774597723292474 | 20995381637208914<br>67842744895846522<br>520832379454230 |

## 1.3.4　中国剩余定理算法实现与测试

算法的输入为同余方程组，输出为同余方程组的解。下面以 3 个方程的同余方程组为例，$m$ 中数据分别表示 3 个方程的模数，$a$ 中数据分别表示 3 个方程的余数，$M_i$ 是除 $m_i$ 外的 $n-1$ 个数的乘积，$M_i^{'}$ 是 $M_i$ 在模 $m_i$ 意义下的逆元，测试数据如表 1-11 所示。

表 1-11　中国剩余定理算法测试数据

| $a$ | $m$ | $M_i$ | $M_i'$ | result |
|---|---|---|---|---|
| 0, 0, 0 | 23, 28, 33 | 924, 759, 644 | 6, 19, 2 | result ≡ 0 (mod 21252) |
| 5, 20, 1 | 23, 28, 33 | 924, 759, 644 | 6, 19, 2 | result ≡ 19900 (mod 21252) |
| 283, 102, 23 | 23, 28, 33 | 924, 759, 644 | 6, 19, 2 | result ≡ 9230 (mod 21252) |

### 1.3.5　Miller-Rabin 素性检测算法实现与测试

算法的输入为待检测的大数，检测次数为 20 次，输出为 True 或 False。若输出为 True，则表示该数通过了 Miller-Rabin 素性检测，可能为素数；若输出为 False，则表示该数不为素数。Miller-Rabin 素性检测算法测试数据如表 1-12 所示。

表 1-12　Miller-Rabin 素性检测算法测试数据

| 输　入 | 输　出 |
|---|---|
| 1000023 | False |
| 1000033 | True |
| 100160063 | False |
| 1500450271 | True |

## 1.4　思考题

（1）Miller-Rabin 素性检测算法并非确定性的素性判定方法，如何通过重复检测提高该算法的可信度？试简要说明。

（2）在厄拉多塞筛算法中需要临时存放大量的数据，如何减少运行时的内存开销？