素数的检验与证明

本文档以 Apache 2.0 开源协议公开发布
地址：https://gitee.com/elflyao/primality_test

一、前言

假设 $n \in \mathbb{N}+$，如何确定 n 是一个素数，是一个重要且复杂的问题，至今依然是一个没有解决的问题。本文档介绍实用的几个算法，包括素数素性的概率性测试算法，和素数素性的确定性证明，并给出相应的 Julia 源代码。本文档并不涉及相关的算法的理论依据，相关理论请查看对应参考文献。

二、试除与筛法

（一）、小素数因子测试（trial division）

当我们尝试确定某个数 $n \in \mathbb{N}+$ 是否是素数的时候，首先要做的是排除小素数 $p \mid n$，这样做能在一开始用很小的代价过滤掉大部分合数。

1、令 P 表示小于等于某个给定正整数上界 B 的素数集合。

2、对所有的 $p \in P$，如果 $p \mid n$，则比较 p, n 是否相等，相等则返回 n 是素数，否则返回 n 是合数。

3、此时，称 n 通过测试，若 $n \geq B^2$，则根据素数定义 n 是素数， 否则需要进行更进一步的测试。

由 n 以内素数个数的近似公式 $\pi(n) = \frac{n}{\ln n}$，n 以内正整数是素数的概率是 $\frac{1}{\ln n}$， 用上界 B 的素数集合试除，n 通过测试的概率等价于 B^2 以内正整数是素数的概率，即概率是 $\frac{1}{2\ln B}$。例如，用 100 以内素数试除， $B = 100$，概率是 $\frac{1}{2\ln 100} \approx 0.11$。

Julia 示例代码

```
Prm = [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]
    for p in Prm
        (n % p == 0) && return (n == p)
    end
```

当 n 比较大的时候，下一步测试一般是概率性素性测试，比如 Miller-Robin 概率性素性测试，有文献建议，选择 B 使得小素因子测试的时间和一次 Miller-Robin 概率性素性测试的时间接近比较好。

（二）、筛法 （ sieve ）

如果待测试正整数是连续多个，即寻找在区间 $[start, stop]$ 中的素数，筛法是一个很好的办法。筛法名字来自埃拉托斯特尼筛（sieve of Eratosthenes），原理很多文献提到了，这里不再赘述。

令 P 表示小于某个给定正整数上界 B 的素数集合，

A、假设 $B^2 >= stop$，此时可以用筛法求出所有区间 $[start, stop]$ 中的素数

令 $B_l = \lfloor \sqrt{start} \rfloor, B_u = \lfloor \sqrt{stop} \rfloor$

1、令 P_l 为小于等于 B_l 的所有素数，P_u 为大于 B_l 小于等于 B_u 所有素数，$Bit[i]$， $i \in [start, stop]$ 为表示对应的 i 是否素数的标志数组，预置所有值为 1。

(1) 对所有 $p \in P_l$，执行

若 $p = 2$，则对所有
$\lceil start/2 \rceil <= k <= \lfloor stop/2 \rfloor$ 置 $Bit[2 * k] = 0$

若 $p > 2$，则执行

$start_p = p * \lceil start/p \rceil$

$if \quad start_p \mod 2 = 0$

$\quad start_p = start_p + p$

$endif$

$k = start_p$

$while \quad k <= stop$

$\qquad Bit[k] = 0$

$\qquad k = k + 2p$

$endwhile$

(2) 对所有 $p \in P_u$，执行

$k = p * p$

$while \quad k <= stop$

$\qquad Bit[k] = 0$

$\qquad k = k + 2p$

$endwhile$

(3) 输出所有 $Bit[i] = 1$ 的 i，既是区间 $[start, stop]$ 中的素数。

B、 假设 B 远小于 $\lfloor \sqrt{start} \rfloor$，则用 1 中的步骤 2 的方法筛选就行了，筛选出的素数叫候选素数，需要经过下面的概率算法筛掉合数

三、Miller-Robin算法

对素数 p，正整数 b，有 Fermat 小定理：

$$b^p \equiv p \pmod{p} \tag{3-1}$$

如果 $GCD(b,p) = 1$，则 $b^{p-1} \equiv 1 \pmod{p}$

费马小定理的逆命题是不成立的，

如果 $GCD(b,n) = 1$，且$b^{n-1} \equiv 1 \pmod{n}$，则 n 不一定是素数，这样的 n 称为基 b 费马伪素数(fermat pseudoprime)，简称 psp(b)。但是，对每个 b 值来说，基于实际计算，可

以发现，基 b 费马伪素数相对素数的比例是很低的，所以有下面的Fermat素性测试，由于测试结果并不能保证一定是素数，所以又称为概率性素性测试方法。

（一）、Fermat 素性测试

假设待测试正整数 n，通过了小于 B 的小因子试除，怀疑为素数，挑选 b 满足 $GCD(b，n)=1$，测试 n 是否满足 $b^{n-1}\equiv 1 \pmod n$，如果满足，则称 n 通过 b 为基的费马素性测试。如果 n 通过了多个 b 的测试，则 n 是素数的可能就很大。

Julia 示例代码

```
function fermatProbablePrimeTest(n, b)
    if gcd(n, b) > 1
        return false
    end
    a = powermod(b, (n-1) >> 1, n)
    return (a == 1) || (a == (n - 1))
end

println("341 test: ", fermatProbablePrimeTest(341, 2))
println("561 test: ", fermatProbablePrimeTest(561, 2))
println("2047 test: ", fermatProbablePrimeTest(2047, 2))
println("I17 test: ", fermatProbablePrimeTest(11111111111111111, 2))
println("I19 test: ", fermatProbablePrimeTest(1111111111111111111, 2))
println("2^31-1 test: ", fermatProbablePrimeTest(2^31-1, 2))
println("2^67-1 test: ", fermatProbablePrimeTest(2^67-1, 2))
```

输出：

```
341 test: true
561 test: true
2047 test: true
I17 test: false
I19 test: true
2^31-1 test: true
2^67-1 test: false
```

其中，I19 是素数，2^31-1 是素数，341是最小的基 2 Fermat伪素数，561 是最小的卡米切尔数。

（二）、Miller-Rabin 素性测试

Fermat 素性测试存在很多通过测试的合数，甚至存在可以通过所有满足 $GCD(b,n)$ 测试的合数，即卡米切尔数（carmichael number）。

考虑更严格的条件，奇数 $p>9$ 如果是素数，令 $2^k*m=p-1$，m 是奇数，必然存在 $0\le i<k$使得

$$b^{2^i*m}\equiv\pm1(\bmod p) \tag{3-2}$$

如果奇合数通过该测试，则称为以b为基的强伪素数（strong pseudoprime），简称 spsp(b)。

以此为基础的素性测试方法称为 Miller-Rabin 素性测试。

Miller-Rabin 强伪素数要比 Fermat 伪素数在整数中的比例更稀少，所以出现测试失败的概率更低，另外，卡米切尔数不会通过所有的 Miller-Rabin 测试。

Julia 示例代码

```
function millerRabinProbablePrimeTest( n, b )
    #n - 1 = 2^k * m, m % 2 != 0
    k = trailing_zeros(n-1)
    m = (n-1) >> k
    a = powermod(b, m, n)
    if (a == 1) || (a == (n-1))
        return true
    else
        for i in 1:k-1
            a = powermod(a, 2, n)
            if a == n - 1
                return true
            end
        end
    end
    return false
end

println("Miller-Rabin 基2素性测试：")
println("341: ", millerRabinProbablePrimeTest(341, 2))
println("561: ", millerRabinProbablePrimeTest(561, 2))
println("2047: ", millerRabinProbablePrimeTest(2047, 2))
println("1093^2: ", millerRabinProbablePrimeTest(1093^2, 2))
println("3511^2: ", millerRabinProbablePrimeTest(3511^2, 2))
println("I17: ", millerRabinProbablePrimeTest(11111111111111111, 2))
println("I19: ", millerRabinProbablePrimeTest(1111111111111111111, 2))
println("2^31-1: ", millerRabinProbablePrimeTest(2^31-1, 2))
println("2^67-1: ", millerRabinProbablePrimeTest(2^67-1, 2))
println("Miller-Rabin 基3素性测试：")
println("341: ", millerRabinProbablePrimeTest(341, 3))
println("561: ", millerRabinProbablePrimeTest(561, 3))
println("2047: ", millerRabinProbablePrimeTest(2047, 3))
println("1093^2: ", millerRabinProbablePrimeTest(1093^2, 3))
println("3511^2: ", millerRabinProbablePrimeTest(3511^2, 3))
println("I17: ", millerRabinProbablePrimeTest(11111111111111111, 3))
println("I19: ", millerRabinProbablePrimeTest(1111111111111111111, 3))
println("2^31-1: ", millerRabinProbablePrimeTest(2^31-1, 3))
println("2^67-1: ", millerRabinProbablePrimeTest(2^67-1, 3))
```

输出

```
Miller-Rabin 基2素性测试：
341: false
561: false
2047: true
1093^2: true
3511^2: true
I17: false
I19: true
2^31-1: true
2^67-1: false
Miller-Rabin 基3素性测试：
341: false
561: false
2047: false
1093^2: false
3511^2: false
I17: false
I19: true
2^31-1: true
2^67-1: false
```

可以看到，卡米切尔数 561 没有通过测试。合数 2047，1093^2，3511^2 通过了基2的测试，特别是仅有的两个 Wieferich primes：1093， 2511的平方，这个如果使用下面几个需要计算 Jaoobi 符号的测试不特别处理，会造成死循环。但是他们没有通过基 3 的测试，因此在 Miller Rabin 测试阶段，可以组合不同的测试基，减少出错可能。

在广义黎曼猜想（GRH）成立情况下，Miller-Robin 测试可以变成多项式运行时间的确定性素性证明算法，即如果 n 通过基于区间 $[1, 2log^2n]$ 的所有整数的Miller-Robin 测试，则 n 是素数。

（三）、伪素数

关于伪素数，有下列事实：

1、若 n 是 fpsp(b)，则 GCD(n, b)=1

2、spsp(b) 必然是 fpsp(b)

3、如果 n 同时是是 fpsp(b1)、 fpsp(b2) 那么 n 必然是 fpsp(b1b2) ,反之则不成立，比如最小的 fpsp(6) 是35，而 35 既不是 fpsp(2)，也不是 fpsp(3)

4、存在一类整数 C，是所有满足 GCD(C, x)=1 的 fpsp(x)，即卡米切尔数（Carmichael number）

5、n 以内卡米切尔数大于 $n^{2/7}$

6、spsp(b) 要比 fpsp(b) 少的多，很多卡米切尔数是 fpsp(b)，但不是 spsp(b)

7、简单的计算可以发现一个事实，n以内的卡米切尔数，spsp(b), fpsp(b)，数量依次递增

有人证明 n 以内卡米切尔数个数 $C(n) > n^{0.332}$, 计算表明，10^{15} 以内有 105212 个，猜测 $C(n) > \sqrt[3]{n}$

猜想，n 以内 spsp(b) 个数大于$\sqrt[3]{n}$ 小于 $\sqrt{n}$。

（四）、强伪素数测试的一些策略

GMP里的素性测试用了高度复合数 210，原理未知，可能是一次性排除待测试数字是2，3，5，7的倍数的可能吧

理论计算表明如果 n是奇数，通过一次米勒罗宾测试是合数的概率是 1/4，费马素性测试合数的概率是 1/2。2^64 内的实际计算表明，真实概率远小于这个，总体上，通过费马素性测试的合数多于米勒罗宾测试，所以实践中采用多次米勒罗宾测试来减少出错可能。

实际计算表明，spsp(2) 数量要少于 spsp(b)， b > 2，一般一次测试采用 b=2
同样，因为合数b并不能减少米勒罗宾测试出错概率，可以采用素数的基b的米勒罗宾测试，即用 b=2,3,5,7....

用ψn表示通过前n个素数的强伪素数测试的最小合数，则：

$\psi1 = 2047 = 23 * 89$

$\psi2 = 1373653 = 829 * 1657$

$\psi3 = 25326001 = 2251 * 11251$

$\psi4 = 3215031751 = 151 * 751 * 28351$

$\psi5 = 2152302898747 = 6763 * 10627 * 29947$

$\psi6 = 3474749660383 = 1303 * 16927 * 157543$

$\psi7 = \psi8 = 341550071728321 = 10670053 * 32010157$

$\psi 9 = \psi 10 = \psi 11 = 3825123056546413051 = 149491 * 747451 * 34233211$

实践中，也可以采取随机整数基测试（即对 n 先基 2 测试，再随机选择几个 $GCD(n,b) = 1$ 的随机整数 b 测试，当然如果发现了 $GCD(n，b) \neq 1$ 则表明 n 是合数）。

实践中，如果 n 是大整数，二进制有 k 位，一次米勒罗宾测试出错的实际概率 < $k^2 * 4^{2-\sqrt{k}}$ （Damg˚ard et al. 1993）。按照这个公式，$k = 500$, 概率小于 4^{-28}，本人猜想，应该小于 4^{-125}，即 2-4 次就足够坚信 n 是素数了（张振祥建议次数用 6 次），出现例外的可能性很低（实际上也存在一些很大的伪素数能通过多次测试）。

四、n-1和n+1方法

（一）n-1 方法

令 P_{n-1} 是 $n-1$ 的所有素因子的集合，如果存在整数 a 使得 $a^{n-1} \equiv 1(\bmod n)$ 且 $a^{(n-1)/p} \not\equiv 1(\bmod n)$ 对所有的 $p \in P_{n-1}$ 都成立，则 n 是素数。

这个测试，对 $n-1$ 的因子已知的情况是确定性的证明方法，当然 n 如果比较大，很难求得 $n-1$ 的分解式。不过，对形如 $k * t^n + 1$ 的数字，如果 k，t 都是小整数，这个要比其他确定性素性证明方法更有效。

Julia 示例代码：

```
using Hecke

function factorList(m)
    lst = []
    f = factor(m)
    for d in f
        append!(lst, BigInt(d.first))
    end
    return lst
end

function ns1prov(n)
    a = 2
    lst = factorList(n-1)
    while true
        if powermod(a, n-1, n) != 1
            return false
        end
        all = true
        for r in lst
            if powermod(a, div(n-1, r), n) == 1
                all = false
                break
            end
        end
        if all
            #println("$n: $a")
            return true
        end
        a = a + 1
    end
end

for t in range(60001, stop=70000, step=1)
    isprime(t) && !ns1prov(t) && println("n-1 prov error on $t")
    !isprime(t) && ns1prov(t) && println("n-1 prov error on $t")
end

println("341:", ns1prov(341))
println("561:", ns1prov(561))
println("2^47-1:", ns1prov(2^47-1))
println("2^61-1:", ns1prov(2^61-1))
println("2^127+45:", ns1prov(big(2)^127+45))
```

输出

```
341:false
561:false
2^47-1:false
2^61-1:true
2^127+45:true
```

对十进制表示在95位以内的整数，如果用二次筛和数域筛等强力分解工具分解 $n-1$，其证明时间还是比较快的。

如果仅仅知道 $n-1$ 的部分分解，记 $n-1=ab$，$0<a\le b+1$，其中 b 完全分解，且 $b>\sqrt{n-1}$，则有下列定理：

若对 b 的所有素因子 p 都存在整数 x，使得 $x^{n-1}\equiv 1(\bmod n)$ 且 $GCD(x^{(n-1)/p}-1,n)=1$，则 n 是素数。

（二）n+1 方法

若 P，Q为整数，定义 Lucas 序列：

$u_0=0$，$u_1=1$，...，$u_{k+2}=Pu_{k+1}-Qu_k$（$k\ge 0$）。

则 特征多项式 为 x^2-Px+Q，令 $D=P^2-4Q$。

设 n 为奇数，$D\equiv 1(\bmod 4)$，$D=P^2-4Q$，$GCD(n，DQ)=1$，若 $u_{n+1}\equiv 1(\bmod n)$，且对 $n+1$ 所有素因子 q 有 $GCD(u_{(n+1)/q}，n)=1$，则 n 是素数。

这个的具体实现方法和基于 Fibonacci、Locus 序列的素性测试是类似的，就不写实现代码了。同样的，因为依赖于 $n+1$ 的分解，所以，n 比较大的时候，可能会很困难。

五、基于 Fibonacci，Lucas 序列的素性测试

（一）、Fibonacci 序列素性测试

定义 Fibonacci 序列，$F_0=0$，$F_1=F_2=1$，$F_{k+2}=F_{k+1}+F_k$，素数 $n>5$，jacobi 符号 $J(\frac{n}{5})=j$，则 $F_{n-j}\equiv 0(\bmod n)$。

利用上面的结果，假如 n 是待测试整数，

1、若 $5|n$ 则返回 n 是合数。

2、否则若 $n \equiv \pm 1(\bmod 5)$ 则 $j=1$，若 $n \equiv \pm 2(\bmod 5)$ 则 $j=-1$。

3、若 $F_{n-j} \equiv 0(\bmod n)$，则返回 n 可能是素数，否则返回 n 是合数。

若合数 n 通过上述测试，称为 Fibonacci 伪素数，简称 fpp，fpp 是无限多的。

Julia示例代码

```
using Hecke

function fibonacciTest(n)
    r = n % 5
    r == 0 && return false
    if r == 1 || r == 4
        j = 1
    end
    if r == 2 || r == 3
        j = -1
    end
    return fibonacci(n-j) % n == 0
end

for i in range(ZZ(7), stop=ZZ(9999), step=2)
    if fibonacciTest(i)
        if !isprime(i)
            println("Fibonacci Pseudoprime: $i")
        end
    end
    if isprime(i)
        if !fibonacciTest(i)
            println("Fibonacci Primality test error at: $i")
        end
    end
end
```

输出

```
Fibonacci Pseudoprime: 323
Fibonacci Pseudoprime: 377
Fibonacci Pseudoprime: 1891
Fibonacci Pseudoprime: 3827
Fibonacci Pseudoprime: 4181
Fibonacci Pseudoprime: 5777
Fibonacci Pseudoprime: 6601
Fibonacci Pseudoprime: 6721
Fibonacci Pseudoprime: 8149
```

对 10000 内大于 5 奇数测试表明，存在 9 个奇伪素数。如果不限定 n 是奇数，则相应存在偶伪素数，最小的一个是 8539786。

另外，该代码是示例性质代码，并没考虑优化，序列的计算没有在模 n 环上进行，当 n 很大时候，可能会内存溢出、计算时间超长等。

（二）、Lucas 序列素性测试

若 $P \in \mathbb{Z}+$ ，$Q \in \mathbb{Z}$，$D = P^2 - 4Q$为整数，定义 Lucas 序列 U，V：

$U_0 = 0$，$U_1 = 1$，$\ldots$，$U_{k+2} = PU_{k+1} - QU_k$（$k \geq 0$），

$V_0 = 2$，$V_1 = P$，$\ldots$，$V_{k+2} = PV_{k+1} - QV_k$（$k \geq 0$），

若 $n \in N+$ 是大于 1 的奇数，选择 D，P，Q 满足 Jacobi 符号 $J(\frac{D}{n}) = -1$,

当 n 是素数且 $GCD(n，Q) = 1$，我们有：

$$U_{n+1} = 0(\bmod n) \tag{5-1}$$

$$V_{n+1} = 2Q(\bmod n) \tag{5-2}$$

另外，如果 n 是奇素数，$\delta(n) = n - J(\frac{D}{n})$，且 $GCD(D,n) = 1$，则：

$$U_{\delta(n)} \equiv 0(\bmod n) \tag{5-3}$$

$$V_{\delta(n)} \equiv 2Q^{(1-J(\frac{D}{n}))/2}(\bmod n) \tag{5-4}$$

$$U_n \equiv J(\frac{D}{n})(\bmod n) \tag{5-5}$$

$$V_n \equiv V_1 = P(\bmod n) \tag{5-6}$$

利用上面的结果的素性测试，称为 Lucas 序列素性测试。如果合数通过测试 $(5-1)$，称为以 P，Q 为参数的 lucas 伪素数，简称 lpsp(P，Q)。如果合数通过测试 $(5-2)$，称为以 P，Q 为参数的 lucas-v 伪素数，简称 vpsp(P，Q)。

对 Lucas 序列有如下快速迭代公式。

$$U_{2k}=U_kV_k \tag{5-7}$$

$$V_{2k}=V_k^2-2Q^k \tag{5-8}$$

$$Q^{2k}=(Q^k)^2 \tag{5-9}$$

$$U_{k+1}=(PU_k+V_k)/2 \tag{5-10}$$

$$V_{k+1}=(DU_k+PV_k)/2 \tag{5-11}$$

$$Q^{k+1}=Q^kQ \tag{5-12}$$

A、参数的选择

（注意，以下方法当 n 是完全平方数时候，将出现$J(\frac{D}{n})\neq -1$，这里建议用前置算法筛选掉完全平方数和含有小素因子的数）

方法一、D 在序列 5，-7，9，-11，13，-15，$...$ 中选择第一个满足 $J(\frac{D}{n})=-1$ 的值，令 $P=1,Q=\frac{1-D}{4}$

这个方法，不会设置 $Q=1$，但是常会设置 $Q=-1$，当 $n\equiv\pm3(\mathrm{mod}10)$ 时，此时 $D=5$。

计算表明， 当 $Q\equiv\pm1(\mathrm{mod}n)$ 时要比 $Q\not\equiv\pm1(\mathrm{mod}n)$ 有更多合数满足公式 $(5-3)$、$(5-4)$、$(5-5)$、$(5-6)$，因此有下面的改进方法。

方法二、按照方法一的选择参数，当出现 $Q=-1$时候，置 D，P，Q 都为5。

用方法二的参数，前 10 个 lpsp 是 323，377，1159，1829，3827，5459，5777，9071，9179，10877，10^{15} 以内有 2402549 个 lpsp，vpsp 更稀少，10^{15} 以内只有 5 个 vpsp。

B、强 lucas 素性测试

如果 n 是奇素数，$J(\frac{D}{n})=-1$，$n+1=m*2^k$，m 是奇数，则必有 r 满足 $0\leq r<k$ 满足下面两个条件之一：

$$U_m \equiv 0(\bmod n) \tag{5-13}$$

$$V_{m*2^r} \equiv 0(\bmod n) \tag{5-14}$$

如果 n 是合数且不是完全平方数，用方法二选择参数D、P、Q，$J(\frac{D}{n})=-1$，满足 $(5-11)$ 或者 $(5-12)$，n 称为强 lucas 伪素数，记做 slpsp(P，Q)。前 10 个 slpsp 是 5459，5777，10877，16109，18971，22499，24569，25199，40309，58519。强 lucas 伪素数远少于 lucas 伪素数，10^{15} 以内只有 474971 个。

C、BPSW 算法

该算法是对 n 首先进行基 2 的 Miller-Rabin 测试，然后用方法二选择参数进行强 Lucas 素性测试。即：

1、如果 n 没有通过基 2 Miller-Rabin 测试，输出 n 是合数，结束。

(这里需要注意的是，因为下面进行Jacobi符号计算的时候，要求n不能是平方数，而基 2 Miller-Rabin 测试，已知是有2个平方数可以通过测试，所以，这里可以附加一次基 3 Miller-Rabin 测试，以避免出现平方数)

2、用方法二选择参数 D，P，Q。

3、进行强 lucas 素性测试，通过则输出 n 可能是素数，否则 n 是合数。

D、增强的 BPSW 算法

基于 vpsp 很稀少的事实，针对 BPSW 增加了两个检验。

1、如果 n 没有通过基 2 Miller-Rabin 测试，输出 n 是合数，结束。

2、用方法二选择参数 D，P，Q。

3、进行强 lucas 素性测试，通过则转步骤 4 进一步检验，否则输出 n 是合数，结束。

4、基于步骤 3 的计算，进一步计算并检验 $5-2$，不满足 $5-2$，则输出 n 是合数，结束。

5、验证 n 是不是满足 $Q^{(n+1)/2} \equiv Q*J(\frac{Q}{n})(\bmod n)$，如果满足，输出 n 可能素数，否则输出 n 是合数。

E、实现算法

(针对BPSW需求的基 2 强伪素数测试，不再赘述，看上面的相应介绍)

输入待测试正奇数 n

1、d = 5, delta = 2

2、测试 $J(\frac{d}{n})$，如果不等于-1，则 delta = -delta, d = delta - d，重复 2。

3、置 D = d, P = 1, Q = (1 - D) / 4，如果 Q = -1，则 D = 5，P = 5，Q = 5。

4、令 $n+1=m*2^k$，m 是奇数，令 ms 表示 m 的二进制表示的长度，即 $2^{ms-1}\leq m<2^{ms}$。$U_c=U_0=0,V_c=V_0=2,Q_c=1$。

5、令 i 从 $ms-1$ 到 0 循环执行 5.1-5.2（ks 至少为1）

5.1 $U_c=U_c*V_c(\bmod n)$，$V_c=V_c^2-2Q_c(\bmod n)$，$Q_c=Q_c^2(\bmod n)$

5.2 如果 m 二进制表示第 i 位是 1，则 $U_c=(PU_c+V_c)/2(\bmod n)$，$V_c=(DU_c+PV_c)/2(\bmod n)$，$Q_c=Q_cQ(\bmod n)$。

6、此时 U_c 即为 U_q，若 $U_c\neq 0$，执行7，否则跳到9。

7、继续测试 V_c。令 $r=0$，循环执行：

7.1、如果 $V_c=0$，跳转到9。

7.2、$V_c=V_c^2-2Q_c(\bmod n)$，$Q_c=Q_c^2(\bmod n)$，$r=r+1$。如果 $r=k$，跳转到8，否则跳转到 7.1 继续循环。

8、输出 n 是合数，结束。

9、此时，强 lucas 素性测试输出 n 可能是素数，如果进一步测试，进行步骤 10。

10、强化的 BPSW 算法，利用了 5-7 的步骤的计算，附加计算并不多。分成两个检验：

10.1、循环执行 $V_c=V_c^2-2Q_c(\bmod n)$，$Q_c=Q_c^2(\bmod n)$，$r=r+1$，直到 $r=k$。

10.2、此时，$V_c=V_{n+1}$，测试 $V_c\equiv 2Q(\bmod n)$ 是否成立，如果不成立，输出 n 是合数，结束。

10.3、测试 $Q^{(n+1)/2} \equiv Q * J(\frac{Q}{n}) \pmod n$ 是否成立，如果不成立，输出 n 是合数，结束。否则，输出 n 通过强化的 BPSW 素性测试，可能是素数。

Julia示例代码

```
using Hecke

function fibonacciTest(n)
    r = n % 5
    r == 0 && return false
    if r == 1 || r == 4
        j = 1
    end
    if r == 2 || r == 3
        j = -1
    end
    return fibonacci(n-j) % n == 0
end

function millerRabbinTest(n, b)
    k = trailing_zeros(n-1)
    m = (n-1) >> k
    a = powermod(b, m, n)
    if (a == 1) || (a == (n-1))
        return true
    else
        for i in 1:k-1
            a = powermod(a, 2, n)
            if a == n - 1
                return true
            end
        end
    end
    return false
end

function lucasSeq(p, q, t)
    ua, ub = 0, 1
    va, vb =  2, p
    u = []
    v = []
    for  i in 1:t
        push!(u, ub)
        t = p*ub - q*ua
        ua, ub = ub, t
        push!(v, vb)
        t = p*vb - q*va
        va, vb = vb, t
    end
    println(u)
    println(v)
```

```
end

function selectParam(n)
    d = 5
    delta = 2
    t = 0
    while jacobi_symbol(d, n) != -1
        delta = -delta
        d = delta - d
    end
    p, q = 1, div((1 - d), 4)
    return q == -1 ? (5, 5, 5) : (d, p, q)
end

function lucasDouble(u, v, q, n)
    u = mod(u * v,  n)
    v = mod(v * v - 2 * q ,  n)
    q = mod(q * q, n)
    return (u, v, q)
end

function lucasNext(u, v, d, p, q, q1, n)
    t = u
    u = mod(p*u + v, n)
    isodd(u) && (u=u+n)
    u=u>>1
    v = mod(d*t + p*v, n)
    isodd(v) && (v=v+n)
    v=v>>1
    q = mod(q * q1, n)
    return (u, v, q)
end

function strongLucasTest(n)
    (d, p, q) = selectParam(n)
    q1 = q
    (u, v, q) = (0, 2, 1)
    k = trailing_zeros(n+1)
    m = (n+1) >> k
    for b in bits(ZZ(m))
        (u, v, q) = lucasDouble(u, v, q, n)
        if  b
            (u, v, q) = lucasNext(u, v, d, p, q, q1, n)
        end
    end
    if u != 0
```

```
        for r in 0:k-1
            if v!= 0
                v = mod(v * v - 2 * q ,  n)
                q = mod(q * q, n)
            else
                return (true, v, q, r, k, q1)
            end
        end
        return (false, 0, 0, 0, 0, 0)
    else
        return (true, v, q, 0, k, q1)
    end
end

function BPSW(n)
    !millerRabbinTest(n, 2) && return false
    (f, _, _, _, _, _) = strongLucasTest(n)
    !f && return false
    return true
end

function enhancedStrongLucasTest(n)
    (f, v, q, r, k, q1) = strongLucasTest(n)
    !f && return false
     for i in r+1:k-1
        v = mod(v * v - 2 * q ,  n)
        q = mod(q * q, n)
    end
    v = mod(v * v - 2 * q ,  n)
    v !=  mod(2*q1, n) && return false
    q != mod(q1*jacobi_symbol(q1, n), n) && return false
    return true
end

function EBPSW(n)
    !millerRabbinTest(n, 2) && return false
    !millerRabbinTest(n, 3) && return false
    !enhancedStrongLucasTest(n) && return false
    return true
end

for i in range(ZZ(7), stop=ZZ(999999), step=2)

    if fibonacciTest(i)
       if !isprime(i)
            println("Fibonacci Pseudoprime: $i")
```

```
        end
    end

    if isprime(i)
        if !fibonacciTest(i)
            println("Fibonacci Primality test error at: $i")
        end
    end
end

for i in range(3, stop=999999, step=2)

    if BPSW(i)
        if !isprime(i)
            println("BPSWPseudoprime: $i")
        end
    end

    if isprime(i)
        if !BPSW(i)
            println("BPSW Primality test error at: $i")
        end
    end
end

for i in range(5, stop=999999, step=2)

    if EBPSW(i)
        if !isprime(i)
            println("EBPSW Pseudoprime: $i")
        end
    end

    if isprime(i)
        if !EBPSW(i)
           println("EBPSW Primality test error at: $i")
        end
    end
end

println("end")
```

六、基于二次域的Frobenius算法

六和七基于下面的定理：如果 n 是素数，对任何交换环 R，我们有

$$a,b \in R,(a+b)^n \equiv a^n+b^n(\mathrm{mod} n)$$

现在考虑二次域，令$a,b,n \in \mathbb{Z}$, P_+ 是-1和所有素数的集合，$c \in P_+$，n 是素数，Jacobi 符号 $J(\frac{c}{n})=-1$ （保证 c 在模 n 环里平方根不是整数）。

二次域整数 $z=a+b\sqrt{c}$，共轭数 $\overline{z}=a-b\sqrt{c}$，范数 $N(z)=z\overline{z}=a^2-b^2c$。

假设 n 是素数，考虑 $z^n=(a+b\sqrt{c})^n \equiv a^n+b^n(\sqrt{c})^n \equiv a+b*c^{(n-1)/2}*\sqrt{c}(\mathrm{mod} n)$

又 $c^{(n-1)/2} \equiv J(\frac{c}{n})=-1(\mathrm{mod} n)$

即 $(a+b\sqrt{c})^n \equiv a-b\sqrt{c}(\mathrm{mod} n)$，或者 $z^n \equiv \overline{z}(\mathrm{mod} n)$，这也被称为二次域上的费马小定理。

以此定理为基础的一系列素性测试称为二次域 Frobenius 素性测试。

以下约定 $R(n,c)$ 表示 $[z=a+b\sqrt{c}|a,b \in \mathbb{Z}_n]$

（一）简单的形式

若 n 不是平方数，令，$a,b \in \mathbb{Z}_n, c \in P_+$，Jacobi符号 $J(\frac{c}{n})=-1$，$(a+b\sqrt{c})^n \equiv a-b\sqrt{c}(\mathrm{mod} n)$，则 n 通过以 (a, b, c) 为参数的二次域 Frobenius 素性测试。

A、单参数的形式

在 P_+ 中找到最小的 c 满足 $J(\frac{c}{n})=-1$，
若 $c=-1$，2，则 $z=2+\sqrt{c}$ 否则 $z=1+\sqrt{c}$
然后测试 $z^n=\overline{z}(\mathrm{mod} n)$ 是否成立。
{此处，若 $c=-1$，取 $z=1+i$，则 $z^2=2i$，存在合数 n = 2047，满足 $J(\frac{-1}{2047})=-1$，$(1+i)^{2047} \equiv (2i)^{1023}(1+i) \equiv 2^{1023}i^{1023}(1+i) \equiv i(1+i) \equiv (1-i)(\mathrm{mod} n)$}

Julia 示例代码

```
using Hecke

function qfconjmod(x, n)
    re = coeff(x, 0)
    ir = mod(ZZ(-coeff(x, 1)), n)
    return parent(x)([re, ir])
end

function qfpowermod(x, p , n)
    @assert p >= 0
    p == 0 && return one(x)
    b = x
    t = ZZ(prevpow(BigInt(2), BigInt(p)))
    r = one(x)
    while true
        if p >= t
            r = mod(r * b, n)
            p -= t
        end
        t >>= 1
        t <= 0 && break
        r = mod(r * r ,n)
    end
    return r
end

function frobenius(n)
    small = [
               -1, 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
            ]
    find = false
    global p = 0
    for c in small
        if jacobi_symbol(ZZ(c), ZZ(n)) == -1
            find = true
            global p = c
            break
        end
    end
    if !find
        #n is prefect square number?
        sq = isqrt(n)
        if sq * sq == n
```

```
                return false
        end
        c = next_prime(last(small)+1)
        while jacobi_symbol(ZZ(c), ZZ(n)) != -1
            c = next_prime(c+1)
        end
        global p = c
    end
    #println("$n: $p")
    F, d = quadratic_field(p)
    if p >= 3
        b = F([1, 1])
    else
        b = F([2, 1])
    end
    r = qfpowermod(b, n, n)
    nb = qfconjmod(b, n)
    return r == nb
end

global b = big(10)^67
@time for i in range(3, step = 2, stop = 19999)
    test = b + i
    if isprime(test)
        if !frobenius(test)
            println("frobenius error at $test")
        end
    end
    if frobenius(test)
        if !isprime(test)
            println("frobenius error at $test")
        end
    end
end
println("end")
```

B、三参数的形式

对待测整数 n，随机选择$z = a + b\sqrt{c} \in R(n,c), c \in P_+$，满足 $J(\frac{c}{n}) = -1$，$J(\frac{N(z)}{n}) = -1$，当

$$z^n \equiv \overline{z} (\mathrm{mod} n)$$

称为 n 通过二次域 Frobenius(a, b, c) 素性测试。实践中，也可以忽略条件 $J(\frac{N(z)}{n}) = -1$。

（二）增强的形式

定义 $\phi_3(x)=x^2+x+1$，$\phi_4(x)=x^2+1$，$\phi_8(x)=x^4+1$。

增强形式利用了如果 n 是素数，则一定存在八次单位根 $\epsilon\in R(n,c)$，满足 $\phi_8(\epsilon)=0$，三次单位根 $\epsilon_3\in R(n,c)$，满足 $\phi_3(\epsilon_3)=0$

A、前置 MR2 测试

输入： 奇数 n

输出： n 为合数，或者

c，$J(\frac{c}{n})=-1$，$\epsilon\in R(n,c)$，$\epsilon^4=-1$，$\phi_8(\epsilon)=0$

1、$n \mod 4=3$

计算 $\alpha=2^{(n-3)/4}(\bmod n)$

如果 $2\alpha^2\neq\pm1(\bmod n)$ 输出 n 是合数

否则，输出$c=-1,\epsilon=\alpha+\alpha\sqrt{c}$

2、$n \mod 8=5$

计算 $\alpha=2^{(n-1)/4}(\bmod n)$

如果 $\alpha^2\neq-1(\bmod n)$ 输出 n 是合数

否则，输出 $c=2,\epsilon=\frac{1+\alpha}{2}\sqrt{c}$,

（这里的分数，在 $\mathbb{Z}_n$ 中的计算可以转换成整数，即如果 α 是偶数， $\frac{1+\alpha}{2}\equiv\frac{1+\alpha+n}{2}(\bmod n)$，考虑到 α 无论奇偶必然小于n-1，所以这里可以省略模 n 运算）

3、$n \mod 8=1$

如果 n 是完全平方数，输出 n 是合数（这里 n 如果是很大的数，其实是完全平方数的概率很低，可以不测试，也可以在基 2 的MR测试前做一次基 3 的测试，同时通过基 2、基 3 测试的平方数，应该概率很小，如果发现了，就是大新闻）

否则找到小的 c 满足 $(\frac{c}{n})=-1$，c 从 3 开始

计算 $\alpha = c^{(n-1)/8} (\bmod n)$

如果 $\alpha^4 \neq -1 (\bmod n)$ 输出 n 是合数

否则，输出 $c = -1, \epsilon = \alpha$

B、SQFT循环 $SQFT_{round}$

输入： 奇数 n，c，$J(\frac{c}{n}) = -1$，$\epsilon \in R(n, c)$，$\epsilon^4 = -1$，$\phi_8(\epsilon) = 0$

输出：n 是合数或者可能是素数

1、随机选择 $z \in R(n, c)$ 满足 $J(\frac{N(z)}{n}) = -1$

（这个对 $N(z)$ 的条件可以去掉，考虑1加上小素数的集合，$z \in R(n, c)$ 表达成$a + b\sqrt{c}$，那么 a, b 可以在集合 P_+ 中取值，假设 a，b 小于100，可以有 26*25 = 650 次测试）

2、如果 $z^n \neq \overline{z}$ 输出 n 是合数

3、如果 $z^{(n^2-1)/8} \notin \{\pm 1, \pm\epsilon, \pm\epsilon^2, \pm\epsilon^3\}$ 输出 n 是合数

4、输出 n 可能是素数

C、SQFT测试 $SQFT(Simplified\, Quadratic\, Frobenius\, test)$

输入： $n, n > 10000$, 测试次数 t

输出：n 是合数或者可能是素数

1、如果 n 被小于 B 素数整除，输出 n 是合数，B 取值可以是 100

2、调用算法 $MR2$

3、如果 n 没被判定为合数，则 $MR2$ 输出 c, ϵ

4、重复 t 次：

用 n, c, ϵ 调用算法 $SQFT_{round}$，如果算法判定 n 是合数，则终止

5、输出 n 可能是素数

Julia 示例代码

```
using Hecke

function qfconjmod(x, n)
    re = coeff(x, 0)
    ir = mod(ZZ(-coeff(x, 1)), n)
    return parent(x)([re, ir])
end

function qfpowermod(x, p , n)
    @assert p >= 0
    p == 0 && return one(x)
    b = x
    t = ZZ(prevpow(BigInt(2), BigInt(p)))
    r = one(x)
    while true
        if p >= t
            r = mod(r * b, n)
            p -= t
        end
        t >>= 1
        t <= 0 && break
        r = mod(r * r, n)
    end
    return r
end

function makeEpsilonList(e, n)
    list = []
    append!(list, [one(e)])
    append!(list, [mod(- one(e), n)])
    append!(list, [e])
    append!(list, [mod(- e, n)])
    e2 = mod(e * e, n)
    append!(list, [e2])
    append!(list, [mod(- e2, n)])
    e3 = mod(e2 * e, n)
    append!(list,[e3])
    append!(list, [mod(- e3, n)])
    return list
end

function MR2(n)
    if n % 4 == 3
        alpha = powermod(ZZ(2), (n-3)>>2, n)
        tmp = (2*alpha*alpha) % n
        (tmp != 1) && (tmp != (n-1)) && return (false, 0, [])
```

```
        F, t = quadratic_field(-1)
        return (true, F, makeEpsilonList(F([alpha, alpha]), n))
    end
    if n % 8 == 5
        alpha = powermod(ZZ(2), (n-1)>>2, n)
        tmp = (alpha*alpha) % n
        (tmp != (n-1)) && return (false, 0, [])
        iseven(alpha) && (alpha += n)
        alpha=(alpha+1)>>1
        F, t = quadratic_field(2)
        return (true, F, makeEpsilonList(F([0, alpha]), n))
    end
    if n % 8 == 1
        c = ZZ(3)
        try_t=0
        while jacobi_symbol(c, n) != -1
            c = next_prime(c)
            try_t+=1
            if try_t > 16
              issquare(n) && return (false, 0, [])
              try_t = -1024
            end
        end
        alpha = powermod(c, (n-1)>>3, n)
        tmp = powermod(alpha, 4, n)
        (tmp != (n-1)) && return (false, 0, [])
        F, t = quadratic_field(c)
        return (true, F, makeEpsilonList(F([alpha, 0]), n))
    end
end

function SQFTRound(x, n, list)
    r = qfpowermod(x, n, n)
    tmp = qfconjmod(x, n)
    #println("1:$n: x:$x, r:$r, conj(x):$tmp")
    r != tmp && return false
    tmp = (n^2-1) >> 3
    (td, tr) = divrem(tmp, n)
    tmp1 = qfpowermod(r, td, n)
    tmp2 = qfpowermod(x, tr, n)
    r = mod(tmp1 * tmp2, n)
    #println("2: $n: $x, $r")
    return in(r, list)
end

function SQFT(n, t)
```

```
    Prm = [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]
    for p in Prm
        (n % p == 0) && return (n == p)
    end
    Q = append!([1], Prm)
    (r, F, list) = MR2(n)
    if r
        #println("MR2 True at $n: $F")
        #println("E: $list")
        i = 0
       for x in Q
            for y in Q
                if x != y
                    if SQFTRound(F([x, y]), n, list)
                        i = i + 1
                        (i > t) && return true
                    else
                        return false
                    end
                end
            end
       end
    else
        return false
    end
end

########################################################

global b = ZZ(10)^67
@time for i in range(3, step = 2, stop = 19999)
    test = b + i
    isprime(test) && !SQFT(test, 1) && println("frobenius error at $test")
    SQFT(test, 1) && !isprime(test) && println("frobenius error at $test")
end
println("end")
```

附加三次根测试的$SQFT3$算法

D、SQFT3循环 $SQFT3_{round}$

输入：整数 n，$GCD(n,6)=1$， 小整数 c，$J(\frac{c}{n})=-1$， 值 $\epsilon,\epsilon^4=-1$ 和 ϵ_3 满足 $\epsilon_3=1$ 或者 $\phi_3(\epsilon_3)=0$,

输出：n 是合数，或者是可能的素数，同时输出下面的值

ϵ'_3, 满足 $\epsilon'_3 = 1$ 或者 $\phi_3(\epsilon'_3) = 0$,

如果 $\epsilon_3 \neq 1$，那么 $\epsilon'_3 = \epsilon_3^{\pm 1}$

1、随机选择 $z \in R(n, c)$ 满足 $(\frac{N(z)}{n}) = -1$

（这里可以忽略 $(\frac{N(z)}{n}) = -1$）

2、如果 $z^n \neq \overline{z}$ 输出 n 是合数

3、如果 $z^{(n^2-1)/8} \notin \{\pm 1, \pm\epsilon, \pm\epsilon^2, \pm\epsilon^3\}$ 输出 n 是合数

4、置 $u = v_3(n^2 - 1), n^2 - 1 = 3^u r$

5、令 $i = \min\{j : 0 \leq j \leq u, z^{3^j r} = 1\}$

6、如果 $i = 0$，输出 n 可能是素数，$\epsilon'_3 = \epsilon_3$

7、置 $\epsilon'_3 = z^{3^{i-1} r}$

8、如果 $\epsilon_3 = 1$ 且 $\phi_3(\epsilon'_3) \neq 0$，输出n是合数

9、如果 $\epsilon_3 \neq 1$ 且 $\epsilon'_3 \neq \epsilon_3^{\pm 1}$，输出 n 是合数

10、输出 n 可能是素数和 ϵ'_3

E、SQFT3测试 $SQFT3$

输入： $n, n > 200$, 测试次数 t

输出：n 是合数或者可能是素数

1、如果 n 被小于 200 素数整除，输出 n 是合数

2、调用 $MR2$，如果判定 n 是合数，结束

3、从 $MR2$ 获得输出的 c, ϵ，并且置 $\epsilon_3 = 1$

4、循环 t次

用 $n, c, \epsilon, \epsilon_3$ 调用 $SQFT3_{round}$

如果 $SQFT3_{round}$ 判定 n 是合数，结束

置 $\epsilon_3 = \epsilon'_3$，这里 ϵ'_3 是算法 $SQFT3_{round}$ 的输出

5、输出 n 可能是素数

七、基于Jacobi和的分圆域素性检验算法： APR-CL算法

算法的一些符号约定

ζ_{p^k} 代表 p^k 次分圆域的单位元，p, q 约定为素数，

$v_q(t)$ 表示 q 在 t 中出现的次数，即若 $q^k \mid t$ 且 $p^{k+1} \nmid t$，则 $v_q(t) = k$

假设需要证明 n 的素性

（一）确定前置参数和预先存储表

A、确定高度复合数 t，保证 t 比较小，且有小因子

比如，$t = 5040 = 2^4 * 3^2 * 5 * 7$

B、确定

$$e(t) = 2 \prod_{q-1|t} q^{v_q(t)+1}$$

其中 q 是素数，$v_q(t)$ 是 q 在 t 中出现的次数，保证 $e(t) > \sqrt{n}$

比如$e(5040) = 2^6 * 3^3 * 5^2 * 7^2 * 11 * 13 * 17 * 19 * 29 * 31 * 37 * 41 * 43 * 61 * 71 * 73 * 113 * 127 * 181 * 211 * 241 * 281 * 337 * 421 * 631 * 1009 * 2521$

一般可以预先存储一些 t 和 $e(t)$ 来简化计算。

C、对所有 $q|e(t)$，执行下列操作

1、找到q 的最小原根 g, 然后，定义函数 $f : [1..p-2] => [1..p-2]$ 满足 $1 - g^x \equiv g^{f(x)} (\mathrm{mod} q)$。

这可以通过建立表 $log(g^x \bmod q) = x$，然后 $f(x) = log((1 - g^x) \bmod q)$ 得

到。

2、对每一个 $p|q-1$ (同时满足 $p|t$) 执行

1)、令 $k=v_p(q-1)$

2)、若 $p^k\neq 2$，计算并存储

$$j_{p,q}=\sum_{x=1}^{q-2}\zeta_{p^k}^{x+f(x)}\in\mathbb{Z}[\zeta_{p^k}]$$

3)、若 $p=2,k\geq 3$，计算并存储

$$j_{2,q}^{*}=\sum_{x=1}^{q-2}\zeta_{2^k}^{2x+f(x)}\in\mathbb{Z}[\zeta_{2^k}]$$

和

$$j_{2,q}^{\#}=\sum_{x=1}^{q-2}\zeta_{2^k}^{2^{k-3}(3x+f(x))}\in\mathbb{Z}[\zeta_{2^k}]$$

（二）素性测试

1、按照前面的步骤预先计算参数和表。

2、判断 $GCD(t*e(t),n)=1$，是否成立，如果不成立，则 n 是合数，算法终止。

3、选择试除上界 B (比如65536)，判断 n 是否有小于等于 B 的因子，如果有，判定 n 是合数，算法终止。否则，如果 $B\geq\sqrt{n}$，判定 n 是素数，算法终止。

令 l^- 表示 $n-1$ 的 $\leq B$ 的奇素因子的集合，令 r^- 表示 $n-1$ 的最大的没有 $\leq B$ 的奇素因子的奇数因子，令 $f^-=(n-1)/r^-$。

同样，令 l^+ 表示 $n+1$ 的 $\leq B$ 的奇素因子的集合，令 r^+ 表示 $n+1$ 的最大的没有 $\leq B$ 的奇素因子的奇数因子，令 $f^+=(n+1)/r^+$。

4、选择小整数 m，进行 m 次概率性素性测试（Miller-Rabin 素性测试），如果失败，n 是合数，算法终止。通常 m 选择1、2已经足够了。（前面的概率性测试方法已经足够了，如果做了前面的算法测试，这里的步骤2、4其实是可以省略的）

5、对所有的 $p^k|t$ 测试是否整除 $n-1$，如果 $n \equiv 1(\bmod p^k)$，则置 $flag_{p^k}=true$，否则 $flag_{p^k}=false$。

6、执行 Lucas-Lehmer 测试：

6.1、$n-1$ 测试：

6.1.1、在 $x \in \{p_1, p_2, ..., p_{50}\}$ 中找一个素数，对所有 $p \in l^-$ 满足：
$x^{(n-1)/p} \not\equiv 1(\bmod n)$ 且 $x^{n-1} \equiv 1(\bmod n)$

6.1.2、若找不到 x 则 n 是合数，算法终止。

6.1.3、对所有 $p \in l^-$ 且 $flag_{p^k}=true$ 的置 $\beta^i_{p^k}=x^{i(n-1)/p^k}, i=0..p^k-1$

6.1.4、对 r^- 执行同样测试，即在 $x \in \{p_1, p_2, ..., p_{50}\}$ 中找一个素数满足 $x^{(n-1)/r^-} \not\equiv 1(\bmod n)$ 且 $x^{n-1} \equiv 1(\bmod n)$

6.2、$n+1$ 测试：计算在环 $A=[\mathbb{Z}/n\mathbb{Z}](T)/(T^2-uT-a)$ 上进行，$\alpha = T \bmod (T^2-uT-a)$

6.2.1、如果$n \equiv 1(\bmod 4)$，对 $a \in \{p_1, p_2, ..., p_{50}\}$，找到 a 满足 $a^{(n-1)/2} \equiv -1(\bmod n)$，如果没找到，$n$ 是合数，算法终止。然后二次式是 T^2-a，其中 $u=0$。
对所有 $flag_{p^k}=true$ 的计算 $\beta^i_{p^k}=a^{i(n-1)/p^k}(\bmod n), i=0..p^k-1$

6.2.2、如果$n \equiv 3(\bmod 4)$，对所有的 $u \in \{1, 2, ..., 50\}$ 找到满足 Jacobi 符号 $J(\frac{u^2+4}{n})=-1$ 的，如果没找到，n 是合数，算法终止，否则，二次式是 T^2-uT-1，此时 $a=1$，如果 $\alpha^{n+1}=-1$ 不成立，n 是合数，算法终止。

6.2.3、找到$x \in A$ 且范数 $N(x)=1$ 的满足所有 $p \in l^+$， $x^{(n+1)/p} \neq 1$ 且 $x^{n+1}=1$，如果没找到，则n 是合数，算法终止。

6.2.4、对 r^+ 执行同样测试。

6.3、对 $n-1$ 测试，如果 $f^- \geq \sqrt{n}$，则通过 6.1.3 即证明 n 是素数，如果 $f^- < \sqrt{n}$，但是 $f^- * B \geq \sqrt{n}$，证明 n 是素数，还需要通过 6.1.4。同样的情况适用于 $n+1$ 测试。

6.4、如果 n 通过 Lucas-Lehmer 测试，对任意 $r|n$，我们有，存在 $i \geq 0$ 使得 $r = n^i (\bmod f^- * f^+)$。

7、选择合适的 t, s

7.1、令 $t^{'}$ 是 t 的一个偶因子，定义

$$s_1 = \frac{1}{2} * \prod_{p\ prime, p|f^- * f^+} p^{v_p(t^{'}) + v_p(f^- * f^+)}$$

$$\tilde{s_2} = \prod_{q\ prime, q-1|t^{'}, q\nmid s_1} q$$

$$s_2 = \tilde{s_2} * \prod_{p\ prime, p|t^{'}, p|\tilde{s_2}} p^{v_p(n^{p-1}-1) + v_p(t^{'}) - 1}$$

取 $t^{'}$ 尽可能小，使得 $s = s_1 * s_2 > \sqrt{n}$

令 $t = t^{'}$，此时我们有 $n^t \equiv 1(\bmod s)$。

8、对每一个奇素数 $p|t$，如果 $n^{p-1} \not\equiv 1(\bmod p^2)$ 或者 $p|(f^- * f^+)$，则 $\lambda_p = true$，否则 $\lambda_p = false$。

9、对每一个整数 k 满足 $p^k|t$, 确定整数 u_k, v_k 满足 $n = u_k p^k + v_k, 0 \leq v_k < p^k$。

10、对每一个素数 p, q 满足 $q|s_2, p|q-1$ 执行：

10.1、令 $k = v_p(q-1), u = u_k, v = v_k$

10.1.1、如果 $p \neq 2$，令 $M = \{x \in \mathbb{Z} : 1 \leq x \leq p^k, x \not\equiv 0(\bmod p)\}$

令 $\sigma_x(x \in M)$ 是 $Q(\zeta_{p^k})$ 的自同构，即 $\sigma_x(\zeta_{p^k}) = \zeta_{p^k}^x$，

计算

$$j_{0,p,q}=\prod_{x\in M}\sigma_x^{-1}((j_{p,q})^x)\in\mathbb{Z}[\zeta_{p^k}]/n\mathbb{Z}[\zeta_{p^k}]$$

和

$$j_{v,p,q}=\prod_{x\in M}\sigma_x^{-1}((j_{p,q})^{[vx/p^k]})\in\mathbb{Z}[\zeta_{p^k}]/n\mathbb{Z}[\zeta_{p^k}]$$

10.1.2、如果 $p^k=2$，置

$$j_{0,2,q}=q,j_{1,2,q}=1$$

10.1.3、如果 $p^k=4$，计算

$$j_{0,2,q}=j_{2,q}^2*q\in\mathbb{Z}[\zeta_4]/n\mathbb{Z}[\zeta_4]$$

和

$$j_{v,2,q}=\begin{cases}1 & v=1\\ j_{2,q}^2 & v=3\end{cases}$$

10.1.4、如果 $p=2,k\geq 3$，定义

$$L=\{x\in\mathbb{Z},1\leq x\leq 2^k,x\text{是奇数}\},M=\{x\in L,x\equiv 1,3(\mathrm{mod}8)\}$$

令 $\sigma_x(x\in M)$ 是 $Q(\zeta_{2^k})$ 的自同构，即 $\sigma_x(\zeta_{2^k})=\zeta_{2^k}^x$，

计算

$$j_{0,2,q}=\prod_{x\in M}\sigma_x^{-1}((j_{2,q}^**j_{2,q})^x)\in\mathbb{Z}[\zeta_{2^k}]/n\mathbb{Z}[\zeta_{2^k}]$$

和

$$j_{v,2,q}=\begin{cases}\prod_{x\in M}\sigma_x^{-1}((j_{2,q}^**j_{2,q})^{[vx/2^k]})\in\mathbb{Z}[\zeta_{2^k}]/n\mathbb{Z}[\zeta_{2^k}] & v\in M\\ (j_{2,q}^{\#})^2\prod_{x\in M}\sigma_x^{-1}((j_{2,q}^**j_{2,q})^{[vx/2^k]})\in\mathbb{Z}[\zeta_{2^k}]/n\mathbb{Z}[\zeta_{2^k}] & v\in L-M\end{cases}$$

10.2、如果 $flag_{p^k}=true$ 执行 10.2.1 否则执行 10.2.2

10.2.1、利用环同态 $\lambda : \mathbb{Z}[\zeta_{p^k}]/n\mathbb{Z}[\zeta_{p^k}] \to \mathbb{Z}/n\mathbb{Z}, \lambda(\zeta_{p^k}) = \beta_{p^k}$ (参见6.2.1)验证

$$\lambda(j_{0,p,q})^u * \lambda(j_{v,p,q}) = \beta_{p^k}^h, 0 \le h \le p^k - 1$$

如果不存在这样的 h 则 n 是合数，算法终止。

10.2.2、验证

$$j_{0,p,q}^u * j_{v,p,q} = \zeta_{p^k}^h \bmod n\mathbb{Z}[\zeta_{p^k}], 0 \le h \le p^k - 1$$

如果不存在这样的 h 则 n 是合数，算法终止。

10.3、如果 $h \not\equiv 0 (\bmod p)$ 且 p 是奇数，则置 $\lambda_p = true$。

如果有 $p|t, \lambda_p = false$ 则执行如下附加测试11,12，否则跳到13

11、选择一个小素数 $q = 2mp + 1$ 满足 $q \nmid s$ 且 $n^{(q-1)/p} \not\equiv 1 (\bmod q)$

12、令 $n = up + v, 0 \le v < p$，执行预计算的 (一) C.1，(一) C.2的2) 两个步骤，然后计算 10.1.1 步骤，最后测试 10.2.2，如果满足 $h \not\equiv 0 (\bmod p)$ 则置 $\lambda_p = true$，否则 n 是合数，算法终止。

13、最后的试除过程，令 $r \equiv n^i (\bmod s), 0 \le i < t$，测试是否存在 $r|n$，如果存在，则 n 是合数，算法终止，否则 n 是素数，证明完成。

八、椭圆曲线素性测试方法

椭圆曲线素性判定算法（Elliptic Curve Primality Proving, ECPP）是一种基于椭圆曲线的素性测试算法，由Atkin和Morain在1993年提出。ECPP算法运用了椭圆曲线上的数学理论，可以在多项式时间内判断一个整数是否为素数。
ECPP算法的基本思路是通过构造合适的椭圆曲线和随机曲线点，利用椭圆曲线点的阶的特性进行素性判断。具体来说，ECPP算法的步骤如下：

首先，随机选择一个整数a，然后构造椭圆曲线E: y^2 = x^3 + ax + b，其中b是随机选择的整数，满足E上至少有一个阶为素数的点P。

利用P的阶的特性，可以得到一个整数r，使得$n = r * |E(F_p)|$，其中$|E(Fp)|$表示在有限域

F_p上的椭圆曲线E上的点数。如果$n \leq \sqrt{2} * |E(F_p)|$，那么可以通过试除法快速判断n是否为素数。

如果n > sqrt(2) * |E(Fp)|，则进行下一步处理。随机选择一个在Fp上的点Q，并计算kP和kQ，其中k是随机选择的整数。

对于所有的素数q <= r，检查是否存在一个整数kq满足kP ≡ kqQ mod n。如果存在这样的kq，那么n不是素数；否则，进行下一步处理。

构造一个新的椭圆曲线E': y^2 = x^3 + ax + b'，其中b'是随机选择的整数，并且满足E'上至少有一个阶为素数的点P'。然后，重复步骤2-4，如果n是素数，则ECPP算法结束。
ECPP算法相对于传统的素性测试算法，具有更高的效率和更强的安全性。但是，ECPP算法的实现较为复杂，需要大量的计算和存储空间，因此在实际应用中不常使用。

九、AKS算法

2002年夏天，三位印度计算机科学家M. Agrawal, N. Kayal和N. Saxena提出了一个确定性多项式时间素数证明算法，称为 AKS 算法。

$n > 1$ 是一个奇数，
1、测试 n 是不是一个整数的幂，如果是，则返回 n 是一个合数。
2、从 2 开始找到最小的数 r，满足：
$GCD(r,n) = 1$, 且 r 不是满足 $1 \leq i \leq (log_2 n)^2$的 $n^i - 1$ 的所有整数的因子。
3、对所有 $1 \leq j < \sqrt{\varphi(r)} log_2 n$，检查是不是满足 $(x+j)^n \equiv x^n + j (\bmod x^r - 1, n)$，如果都满足，则 n 是素数，否则 n 是合数。

该算法，也可以在分圆域中进行，令 ζ_r 是 r 次分圆域的一个单位元，则
对所有 $1 \leq j < \sqrt{\varphi(r)} log_2 n$，检查是不是满足 $(\zeta_r + j)^n \equiv \zeta_r^n + j (\bmod n)$，如果都满足，则 n 是素数，否则 n 是合数。

在理论上，AKS算法是一种确定性的素性测试算法，可以保证在多项式级别的时间内判断一个整数是否为素数。但是，实际上AKS算法的时间复杂度仍然比较高，在实际应用中并不常用。
针对AKS算法的时间复杂度较高这一问题，学术界提出了很多改进方案，比如：

Berrizbeitia-Vildósola算法：这是一种基于AKS算法的改进算法，使用了循环向量和分治技术，在一定程度上降低了AKS算法的时间复杂度。

Almeida-Carvalho算法：这是一种基于AKS算法和多项式求逆的算法，可以在AKS算法的基础上进一步缩短判断素数的时间。

Fouvry-Klüners-Morain算法：这是一种使用了数学定理和算法技巧的素数测试算法，可以在多项式时间内判断一个整数是否为素数。
需要注意的是，这些改进算法仍然是相对较为复杂的算法，一般只在学术研究中使用，而在实际应用中，仍然会优先考虑使用更加高效的素性测试算法，比如Miller-Rabin算法。

参考文献见 https://gitee.com/elflyao/primality_test/tree/master/references

---

附录

一、基2伪素数分布情况

范围	伪素数	强伪素数	卡米切尔数	伪素数比例	强伪素数比例	卡米切尔数比例
2^9	1	0	0	1.95E03	0.00E+00	3.81E06
2^10	3	0	1	2.93E03	0.00E+00	9.77E04
2^11	8	1	3	3.91E03	4.88E04	1.46E03
2^12	13	3	5	3.17E03	7.32E04	1.22E03
2^13	19	4	6	2.32E03	4.88E04	7.32E04
2^14	32	6	9	1.95E03	3.66E04	5.49E04
2^15	45	7	10	1.37E03	2.14E04	3.05E04
2^16	64	11	15	9.77E04	1.68E04	2.29E04
2^17	89	18	19	6.79E04	1.37E04	1.45E04
2^18	124	24	23	4.73E04	9.16E05	8.77E05
2^19	175	34	33	3.34E04	6.49E05	6.29E05

范围	伪素数	强伪素数	卡米切尔数	伪素数比例	强伪素数比例	卡米切尔数比例
2^20	251	49	45	2.39E04	4.67E05	4.29E05
2^21	361	75	55	1.72E04	3.58E05	2.62E05
2^22	502	104	69	1.20E04	2.48E05	1.65E05
2^23	693	147	95	8.26E05	1.75E05	1.13E05
2^24	944	210	130	5.63E05	1.25E05	7.75E06
2^25	1264	296	162	3.77E05	8.82E06	4.83E06
2^26	1713	409	214	2.55E05	6.09E06	3.19E06
2^27	2361	552	290	1.76E05	4.11E06	2.16E06
2^28	3169	734	375	1.18E05	2.73E06	1.40E06
2^29	4232	981	483	7.88E06	1.83E06	9.00E07
2^30	5749	1311	656	5.35E06	1.22E06	6.11E07
2^31	7750	1736	864	3.61E06	8.08E07	4.02E07
2^32	10403	2314	1118	2.42E06	5.39E07	2.60E07
2^33	14011	3093	1446	1.63E06	3.60E07	1.68E07
2^34	18667	4139	1874	1.09E06	2.41E07	1.09E07
2^35	24958	5511	2437	7.26E07	1.60E07	7.09E08
2^36	33389	7396	3130	4.86E07	1.08E07	4.55E08
2^37	44540	9835	4058	3.24E07	7.16E08	2.95E08
2^38	59565	13106	5188	2.17E07	4.77E08	1.89E08
2^39	79343	17493	6642	1.44E07	3.18E08	1.21E08
2^40	105659	23270	8521	9.61E08	2.12E08	7.75E09
2^41	141147	31115	11002	6.42E08	1.41E08	5.00E09

范围	伪素数	强伪素数	卡米切尔数	伪素数比例	强伪素数比例	卡米切尔数比例
2^42	188231	41664	14236	4.28E08	9.47E09	3.24E09
2^43	250568	55763	18400	2.85E08	6.34E09	2.09E09
2^44	333737	74739	23631	1.90E08	4.25E09	1.34E09
2^45	445316	100342	30521	1.27E08	2.85E09	8.67E10
2^46	593366	134559	39376	8.43E09	1.91E09	5.60E10
2^47	792172	180725	50685	5.63E09	1.28E09	3.60E10
2^48	1059097	243566	65590	3.76E09	8.65E10	2.33E10
2^49	1416055	327731	84817	2.52E09	5.82E10	1.51E10
2^50	1893726	441270	109857	1.68E09	3.92E10	9.76E11
2^51	2532703	594585	141892	1.12E09	2.64E10	6.30E11
2^52	3390284	803252	183507	7.53E10	1.78E10	4.07E11
2^53	4540673	1085426	237217	5.04E10	1.21E10	2.63E11
2^54	6086093	1468777	307278	3.38E10	8.15E11	1.71E11
2^55	8167163	1988905	398506	2.27E10	5.52E11	1.11E11
2^56	10964612	2697846	517446	1.52E10	3.74E11	7.18E12
2^57	14731767	3662239	672105	1.02E10	2.54E11	4.66E12
2^58	19806649	4976375	873109	6.87E11	1.73E11	3.03E12
2^59	26651383	6767707	1136472	4.62E11	1.17E11	1.97E12
2^60	35893886	9212942	1479525	3.11E11	7.99E12	1.28E12
2^61	48374139	12552513	1927138	2.10E11	5.44E12	8.36E13
2^62	65247459	17114780	2513234	1.41E11	3.71E12	5.45E13
2^63	88069251	23355139	3278553	9.55E12	2.53E12	3.55E13

范围	伪素数	强伪素数	卡米切尔数	伪素数比例	强伪素数比例	卡米切尔数比例
2^64	118968379	31894014	4279356	6.45E12	1.73E12	2.32E13

二、二次域上的计算

对 $z=a+b\sqrt{c}\in R(n,c)$ 可以存储成两个整数对 (a,b) 的形式。

若 $z_1=a_1+b_1\sqrt{c}\in R(n,c),z_2=a_2+b_2\sqrt{c}\in R(n,c)$,

1、 $z=a+b\sqrt{c}=z_1+z_2,a=(a_1+a_2)(\mathrm{mod} n),b=(b_1+b_2)(\mathrm{mod} n)$

2、 $z=a-b\sqrt{c}=z_1-z_2,a=(a_1-a_2)(\mathrm{mod} n),b=(b_1-b_2)(\mathrm{mod} n)$

3、 $z=a+b\sqrt{c}=z_1*z_2,a=(a_1*a_2+b_1*b_2*c)(\mathrm{mod} n),b=(a_1*b_2+a_2*b_1)(\mathrm{mod} n)$。

这里令 $m_1=a_1*b_2,m_2=a_2*b_1$，则

$$a=((a_1+c*b_1)*(a_2+b_2)-m_1-c*m_2)(\mathrm{mod} n)$$

$$b=m_1+m_2(\mathrm{mod} n)$$

这样可以从 4 个大数乘法变成 3 个。

三、一些特殊选定的 t 与对应的 $e(t)$

t	$log_{10}(e(t))$
$2=2$	1.38
$4=2^2$	2.38
$6=2\cdot 3$	2.702
$12=2^2\cdot 3$	4.816
$24=2^3\cdot 3$	5.117
$30=2\cdot 3\cdot 5$	5.235

t	$log_{10}(e(t))$
$36 = 2^2 \cdot 3^2$	8.14
$60 = 2^2 \cdot 3 \cdot 5$	9.833
$72 = 2^3 \cdot 3^2$	10.304
$108 = 2^2 \cdot 3^3$	10.654
$120 = 2^3 \cdot 3 \cdot 5$	11.747
$144 = 2^4 \cdot 3^2$	11.836
$180 = 2^2 \cdot 3^2 \cdot 5$	15.415
$240 = 2^4 \cdot 3 \cdot 5$	15.66
$360 = 2^3 \cdot 3^2 \cdot 5$	19.192
$420 = 2^2 \cdot 3 \cdot 5 \cdot 7$	20.574
$540 = 2^2 \cdot 3^3 \cdot 5$	23.095
$720 = 2^4 \cdot 3^2 \cdot 5$	23.105
$840 = 2^3 \cdot 3 \cdot 5 \cdot 7$	24.936
$1008 = 2^4 \cdot 3^2 \cdot 7$	25.465
$1080 = 2^3 \cdot 3^3 \cdot 5$	26.872
$1200 = 2^4 \cdot 3 \cdot 5^2$	29.004
$1260 = 2^2 \cdot 3^2 \cdot 5 \cdot 7$	31.059
$1680 = 2^4 \cdot 3 \cdot 5 \cdot 7$	33.43
$2016 = 2^5 \cdot 3^2 \cdot 7$	33.886
$2160 = 2^4 \cdot 3^3 \cdot 5$	36.757
$2520 = 2^3 \cdot 3^2 \cdot 5 \cdot 7$	40.687
$3360 = 2^5 \cdot 3 \cdot 5 \cdot 7$	42.073

t	$log_{10}(e(t))$
$3780 = 2^2 \cdot 3^3 \cdot 5 \cdot 7$	44.198
$5040 = 2^4 \cdot 3^2 \cdot 5 \cdot 7$	52.185
$7560 = 2^3 \cdot 3^3 \cdot 5 \cdot 7$	57.704
$8400 = 2^4 \cdot 3 \cdot 5^2 \cdot 7$	59.712
$10080 = 2^5 \cdot 3^2 \cdot 5 \cdot 7$	64.132
$12600 = 2^3 \cdot 3^2 \cdot 5^2 \cdot 7$	68.994
$15120 = 2^4 \cdot 3^3 \cdot 5 \cdot 7$	79.352
$25200 = 2^4 \cdot 3^2 \cdot 5^2 \cdot 7$	89.622
$30240 = 2^5 \cdot 3^3 \cdot 5 \cdot 7$	95.78
$42840 = 2^3 \cdot 3^2 \cdot 5 \cdot 7 \cdot 17$	101.235
$50400 = 2^5 \cdot 3^2 \cdot 5^2 \cdot 7$	101.569
$55440 = 2^4 \cdot 3^2 \cdot 5 \cdot 7 \cdot 11$	106.691
$65520 = 2^4 \cdot 3^2 \cdot 5 \cdot 7 \cdot 13$	115.895
$75600 = 2^4 \cdot 3^3 \cdot 5^2 \cdot 7$	116.79
$85680 = 2^4 \cdot 3^2 \cdot 5 \cdot 7 \cdot 17$	129.398
$110880 = 2^5 \cdot 3^2 \cdot 5 \cdot 7 \cdot 11$	137.324
$128520 = 2^3 \cdot 3^3 \cdot 5 \cdot 7 \cdot 17$	145.431
$131040 = 2^5 \cdot 3^2 \cdot 5 \cdot 7 \cdot 13$	151.897
$166320 = 2^4 \cdot 3^3 \cdot 5 \cdot 7 \cdot 11$	156.844
$196560 = 2^4 \cdot 3^3 \cdot 5 \cdot 7 \cdot 13$	169.327
$257040 = 2^4 \cdot 3^3 \cdot 5 \cdot 7 \cdot 17$	188.309
$332640 = 2^5 \cdot 3^3 \cdot 5 \cdot 7 \cdot 11$	206.979

t	$log_{10}(e(t))$
$393120 = 2^5 \cdot 3^3 \cdot 5 \cdot 7 \cdot 13$	215.405
$514080 = 2^5 \cdot 3^3 \cdot 5 \cdot 7 \cdot 17$	223.283
$655200 = 2^5 \cdot 3^2 \cdot 5^2 \cdot 7 \cdot 13$	232.767
$720720 = 2^4 \cdot 3^2 \cdot 5 \cdot 7 \cdot 11 \cdot 13$	237.414
$831600 = 2^4 \cdot 3^3 \cdot 5^2 \cdot 7 \cdot 11$	251.01
$942480 = 2^4 \cdot 3^2 \cdot 5 \cdot 7 \cdot 11 \cdot 17$	251.021
$982800 = 2^4 \cdot 3^3 \cdot 5^2 \cdot 7 \cdot 13$	260.117
$1081080 = 2^3 \cdot 3^3 \cdot 5 \cdot 7 \cdot 11 \cdot 13$	263.037
$1285200 = 2^4 \cdot 3^3 \cdot 5^2 \cdot 7 \cdot 17$	272.555
$1413720 = 2^3 \cdot 3^3 \cdot 5 \cdot 7 \cdot 11 \cdot 17$	283.806
$1441440 = 2^5 \cdot 3^2 \cdot 5 \cdot 7 \cdot 11 \cdot 13$	301.222
$1663200 = 2^5 \cdot 3^3 \cdot 5^2 \cdot 7 \cdot 11$	315.558
$1965600 = 2^5 \cdot 3^3 \cdot 5^2 \cdot 7 \cdot 13$	326.018
$2162160 = 2^4 \cdot 3^3 \cdot 5 \cdot 7 \cdot 11 \cdot 13$	349.475
$2827440 = 2^4 \cdot 3^3 \cdot 5 \cdot 7 \cdot 11 \cdot 17$	357.833
$3341520 = 2^4 \cdot 3^3 \cdot 5 \cdot 7 \cdot 13 \cdot 17$	389.642
$3603600 = 2^4 \cdot 3^2 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13$	396.884
$4324320 = 2^5 \cdot 3^3 \cdot 5 \cdot 7 \cdot 11 \cdot 13$	455.899
$5654880 = 2^5 \cdot 3^3 \cdot 5 \cdot 7 \cdot 11 \cdot 17$	458.434
$6683040 = 2^5 \cdot 3^3 \cdot 5 \cdot 7 \cdot 13 \cdot 17$	469.891
$7207200 = 2^5 \cdot 3^2 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13$	494.198
$10810800 = 2^4 \cdot 3^3 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13$	560.776

t	$log_{10}(e(t))$
$16707600 = 2^4 \cdot 3^3 \cdot 5^2 \cdot 7 \cdot 13 \cdot 17$	575.923
$18378360 = 2^3 \cdot 3^3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17$	599.16
$21621600 = 2^5 \cdot 3^3 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13$	716.709
$36756720 = 2^4 \cdot 3^3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17$	762.754
$61261200 = 2^4 \cdot 3^2 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13 \cdot 17$	819.989
$73513440 = 2^5 \cdot 3^3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17$	966.85
$122522400 = 2^5 \cdot 3^2 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13 \cdot 17$	1038.433
$183783600 = 2^4 \cdot 3^3 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13 \cdot 17$	1171.776
$367567200 = 2^5 \cdot 3^3 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13 \cdot 17$	1501.792
$698377680 = 2^4 \cdot 3^3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19$	1532.79
$1163962800 = 2^4 \cdot 3^2 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19$	1650.98
$1396755360 = 2^5 \cdot 3^3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19$	1913.604
$2327925600 = 2^5 \cdot 3^2 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19$	2082.848
$3491888400 = 2^4 \cdot 3^3 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19$	2388.47
$6983776800 = 2^5 \cdot 3^3 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19$	3010.872

四、$N+1$一些计算的说明

乘法 $(x_0+x_1\alpha)(y_0+y_1\alpha)=z_0+z_1\alpha$

(1)、$n=1(\mathrm{mod} 4)$，$p_0=x_0y_0, p_1=x_1y_1, s_0=x_0+x_1, s_1=y_0+y_1$，则 $z_0=(p_0+ap_1)(\mathrm{mod} n), z_1=(s_0s_1-p_0-p_1)(\mathrm{mod} n)$

(2)、$n=3(\mathrm{mod} 4)$，$p_0=x_0y_0, p_1=x_1y_1, s_0=x_0+x_1, s_1=y_0+y_1$，则 $z_0=(p_0+p_1)(\mathrm{mod} n), z_1=(s_0s_1+(u-1)p_1-p_0)(\mathrm{mod} n)$

平方 $(x_0+x_1\alpha)^2=z_0+z_1\alpha$，由于这个测试只针对 $(x_0+x_1\alpha)$ 范数是 1 的，所以只需要计算 $s=ux_1+2x_0$，然后 $z_0=x_0s-1(\mathrm{mod} n), z_1=x_1s(\mathrm{mod} n)$。

在环 $A=[\mathbb{Z}/n\mathbb{Z}](T)/(T^2-uT-1)$ 上计算 α^{n+1} 时，注意到 $\alpha=u\alpha+1$，然后 $\alpha^{n+1}=(\alpha^2)^{(n+1)/2}$，然后按照上面的规则计算即可。

对范数是 1 的环 A 中元素 x，可以这样获得：对 $m\in\{1,2,...,50\}$ 考虑 $(\alpha+m)/(\overline{\alpha}+m)\in A$，若 $n\equiv 1(\mathrm{mod} 4), \overline{\alpha}=-a$, 若 $n\equiv 3(\mathrm{mod} 4), \overline{\alpha}=u-a$, 于是 $x=\frac{m^2+a}{m(m+u)-a}+\frac{(2m+u)}{m(m+u)-a}\alpha$。此处，当 n 是素数时候，$(m(m+u)-a)^{-1}$ 总可以计算出来。

五、分圆整数环上的计算

p^k 次分圆多项式可以表示成$f(x)=\frac{x^p-1}{x^{p-1}-1}$

或者可以写成

$$f(x)=\sum_{i=0}^{p-1}x^{i*p^{k-1}}$$

由此定义的本原单位根 ζ_{p^k} 满足

$$\sum_{i=0}^{p-1}x^{i*p^{k-1}}=0$$

或者说

$$x^{(p-1)p^{k-1}}=-\sum_{i=0}^{p-2}x^{i*p^{k-1}}$$

令 $m=(p-1)p^{p-1}$，于是分圆整数环 $\mathbb{Z}[\zeta_{p^k}]$ 上的整数可以表示为

$$\sum_{i=0}^{m-1}a_i\zeta_{p^k}^i$$

假设有分圆整数环 $\mathbb{Z}[\zeta_{p^k}]$ 上的整数

$$A=\sum_{i=0}^{m-1} a_i \zeta_{p^k}^i, B=\sum_{i=0}^{m-1} b_i \zeta_{p^k}^i$$

1、对于加减法，若

$$C=A+B=\sum_{i=0}^{m-1} c_i \zeta_{p^k}^i$$

则 $c_i=a_i+b_i, 0 \leq i \leq m-1$

同样若 $C=A-B$，则 $c_i=a_i-b_i, 0 \leq i \leq m-1$

2、对于乘法，则相对复杂，若求

$$C=A * B=\sum_{i=0}^{m-1} c_i \zeta_{p^k}^i$$

则先置 $c_i=0,0 \leq i \leq m-1$

然后，对 $0 \leq i \leq m-1,0 \leq j \leq m-1$

令 $l=i+j(\bmod p^k)$，若 $0 \leq l<m$则 $c_l=c_l+a_i * b_j$，若 $m \leq l<p^k$，则对每一个 $0 \leq n \leq p-2$，$c_{l-n p^{k-1}}=c_{l-n p^{k-1}}-a_i b_j$

3、σ_x^{-1} 的计算

对 $a=(a_i)_{i=0}^{m-1} \in \mathbb{Z}[\zeta_{p^k}]/n\mathbb{Z}[\zeta_{p^k}]$，计算得到 $b=(b_i)_{i=0}^{m-1} \in \mathbb{Z}[\zeta_{p^k}]/n\mathbb{Z}[\zeta_{p^k}]$，满足 $\sigma_x^{-1}(a)=b$。

3.1、令 $a_i=0, i \geq m$

3.2、对 $i=0,1,\ldots,m-1$，令 $b_i=a_{xi \bmod p^k}$

3.3、对 $i=m, m+1,\ldots,p^k-1, j=1,2,\ldots,p-1$，令

$$b_{i-j p^{k-1}}=(b_{i-j p^{k-1}}-a_{xi \bmod p^k}) \bmod n$$