Int64 以内 Rabin-Miller 强伪素数测试

和 Pollard ρ因数分解的算法实现

在求解 POJ1811题 Prime Test中应用到的两个重要算法是 Rabin-Miller 强伪素数测试和 Pollard ρ因数分解算法。前者可以在 $O(\log n)$的时间内以很高的成功概率判断一个整数是否是素数。 后者可以在最优 $O(\sqrt[4]{n})$的时间内完成合数的因数分解。这两种算法相对于试除法都显得比较复杂。 本文试图对这两者进行简单的阐述，说明它们在 32 位计算机上限制在 64 位以内的条件下的实现中的细节。 下文提到的所有字母均表示整数。

一、 **Rabin-Miller** 强伪素数测试

Rabin-Miller 强伪素数测试的基本思想来源于如下的 Fermat小定理：

如果 p 是一个素数，则对任意 a 有 $a^p \equiv a \pmod p$。特别的，如果 p 不能整除 a，则还有 $a^{p-1} \equiv 1 \pmod p$。

利用 Fermat 小定理可以得到一个测试合数的有力算法： 对 $n>1$，选择 $a>1$，计算 $a^{n-1} \pmod n$，若结果不等于 1 则 n 是合数。若结果等于 1 则 n 可能是素数，并被称为一个以 a 为基的弱 可能素数 （weak **probable prime base** a， a**-PRP**）；若 n 是合数，则又被称为一个以 a 为基的 伪素数 （pseudoprime）。

这个算法的成功率是相当高的。 在小于 25,000,000,000的 1,091,987,405个素数中，一共只用 21,853 个以 2 为基的伪素数。但不幸的是， Alford 、Granville 和 Pomerance在 1994 年证明了存在无穷多个被称为 Carmichael 数的整数对于任意与其互素的整数 a 算法的计算结果都是 1。最小的五个 Carmichael 数是 561、1,105、1,729、2,465 和 2,801。

考虑素数的这样一个性质：若 n 是素数，则 1 对模 n 的平方根只可能是 1 和 -1。因此 a^{n-1}对模 n 的平方根 $a^{\frac{n-1}{2}}$也只可能是 1 和 -1。如果 $\frac{n-1}{2}$本身还是一个偶数，我们可以再取一次平方根……将这些写成一个算法：

（Rabin-Miller 强伪素数测试）记 $n-1=2^s d$，其中 d 是奇数而 s 非负。如果 $a^d \equiv 1 \pmod{n}$，或者对某个 $0 \le r < s$ 有 $a^{2^r d} \equiv -1 \pmod{n}$，则认为 n 通过测试，并称之为一个以 a 为基的强可能素数（strong probable prime base a，a-SPRP）。若 n 是合数，则又称之为一个以 a 为基的强伪素数（strong pseudoprime）。

这个测试同时被冠以 Miller 的名字是因为 Miller 提出并证明了如下测试：如果扩展黎曼猜想（extended Riemann hypothesis）成立，那么对于所有满足 $1 < a < 2(\log n)^2$ 的基 a，n 都是 a-SPRP，则 n 是素数。

尽管 Monier 和 Rabin 在 1980 年证明了这个测试的错误概率（即合数通过测试的概率）不超过 $\frac{1}{4}$，单个测试相对来说还是相当弱的（Pomerance，Selfridge 和 Wagstaff, Jr.证明了对任意 $a > 1$ 都存在无穷多个 a-SPRP）。但由于不存在“强 Carmichael 数”（任何合数 n 都存在一个基 a 试之不是 a-SPRP），我们可以组合多个测试来产生有力的测试，以至于对足够小的 n 可以用来证明其是否素数。

取前 k 个素数为基，并用 Ψ_k 来表示以前 k 个素数为基的强伪素数，Riesel 在 1994 年给出下表：

$\Psi_1 = 2{,}047$

$\Psi_2 = 1{,}373{,}653$

$\Psi_3 = 25{,}326{,}001$

$\Psi_4 = 3{,}215{,}031{,}751$

$\Psi_5 = 2{,}152{,}302{,}898{,}747$

$\Psi_6 = 3{,}474{,}749{,}660{,}383$

$\Psi_7 = 341{,}550{,}071{,}728{,}321$

$\Psi_8 = 341{,}550{,}071{,}728{,}321$

$\Psi_9 \le 41{,}234{,}316{,}135{,}705{,}689{,}041$

$\Psi_{10} \le 1{,}553{,}360{,}566{,}073{,}143{,}205{,}541{,}002{,}401$

$\Psi_{11} \le 56{,}897{,}193{,}526{,}942{,}024{,}370{,}326{,}972{,}321$

考虑到 64 位二进制数能表示的范围，只需取前 9 个素数为基，则对小于 Ψ_8 的所有大于 1 的整数测试都是正确的；对大于或等于 Ψ_8 并小于 2^{64} 的整数测试错误的概率不超过 $\frac{1}{2^{18}}$。

Rabin-Miller 强伪素数测试本身的形式稍有一些复杂，在实现时可以下面的

简单形式代替：

对 $n>1$，如果 $a^{n-1}\equiv 1(\bmod n)$ 则认为 n 通过测试。

代替的理由可简单证明如下：

仍然记 $n-1=2^s d$，其中 d 是奇数而 s 非负。若 n 是素数，由 $a^{n-1}\equiv 1(\bmod n)$ 可以推出 $a^{2^{s-1}d}\equiv a^{\frac{n-1}{2}}\equiv -1(\bmod n)$ 或 $a^{2^{s-1}d}\equiv 1(\bmod n)$。若为前者，显然取 $r=s-1$ 即可使 n 通过测试。若为后者，则继续取平方根，直到对某个 $1\le r<s$ 有 $a^{2^r d}\equiv -1(\bmod n)$，或 $a^{2d}\equiv 1(\bmod n)$。无论 $a^d\equiv 1(\bmod n)$ 还是 $a^d\equiv -1(\bmod n)$，n 都通过测试。

Rabin-Miller 强伪素数测试的核心是幂取模（即计算 $a^s(\bmod n)$）。计算幂取模有以下的 $O(\log n)$ 算法（以 Sprache伪代码语言描述）：

```
function powermod(a, s, n)
{
	p := 1
	b := a
	while s > 0
	{
		if (s & 1) == 1 then p := p * b % n
		b := b * b % n
		s := s >> 1
	}
	return p
}
```

这个算法在 32 位计算机上实现的难点在于指令集和绝大部分编程语言的编译器都只提供了 32 位相乘结果为 64 位的整数乘法，浮点运算由于精度的问题不能应用于这里的乘法。唯一解决办法是模仿一些编译器内建的 64 位整数乘法来实现两个无符号 64 位相乘结果为 128 位的乘法。这个乘法可以将两个乘数分别分割成两个 32 位数来实现。为方便乘法之后的取模运算，运算结果应当用连续的 128 个二进制位来表示。以下是其伪代码：

```
function mul64to128(a, b)
{
	{ah, al} := {a >> 32, a & 0xffffffff}
	{bh, bl} := {b >> 32, b & 0xffffffff}
```

```
        rl := al * bl
        c := al * bh
        rh := c >> 32
        c := c << 32
        rl := rl + c

        if rl < c then rh++                    //    处理进位

        c := ah * bl
        rh := rh + (c >> 32)
        c := c << 32
        rl := rl + c

        if rl < c then rh++                    //    处理进位

        rh := rh + ah * bh
        return {rh, rl}
}
```

乘法之后的取模运算可以用浮点运算快速完成。具体做法是乘积的高 64 位和低 64 位分别先对除数取模，然后再利用浮点单元合并取模。这里的浮点运算要求浮点单元以最高精度运算， 计算前应先将浮点单元控制字中的精度控制位设置为 64 位精度。为保证精度，应当用 80 位浮点数实现此运算。伪代码如下：

```
function mod64(rh, rl, n)
{
        rh := rh % n
        rl := rl % n
        r := fmodl(rh * 2 ** 64, n)
        r := r + rl

        if r < n then r := r - n                    //    处理进位

        r := fmodl(r, n)
        return r
}
```

至此， Rabin-Miller 强伪素数测试的核心已经全部实现。

二、 Pollard ρ因数分解算法

Pollard ρ因数分解算法又称为 Pollard Monte Carlo 因数分解算法。它的核心思想是：选取一个随机数 a。再选一个随机数 b。检查 $\gcd(a-b,n)$是否大于 1。

若大于 1， $\gcd(a-b,n)$就是 n 的一个因子。若不大于 1，再选取随机数 c，检查 $\gcd(c-b,n)$和 $\gcd(c-a,n)$。如此继续，直到找到 n 的一个非平凡因子。

它的主要实现方法是从某个初值 $0<x_1<n$ 出发，通过一个适当的多项式 $f(x)$进行迭代，产生一个伪随机迭代序列 $\{x_1,x_2,x_3,\cdots\}=\{x_1,f(x_1)\ f(f(x_1)),\cdots\}$直到其对模 n 出现循环。多项式 $f(x)$的选择直接影响着迭代序列的长度。经典的选择是选取 $f(x)=x^2+a$，其中 $a\neq 0,-2(\mathrm{mod}n)$。不选择 0 和 -2的原因是避免当序列中某一项 $x_k\equiv\pm1(\mathrm{mod}n)$时后续各项全部为 1 的情况。迭代序列出现循环的期望时间和期望长度都与 $\sqrt{n}$成正比。设 n 有一个非平凡因子 p。由前面可知，迭代序列关于模 p 出现循环的期望时间和期望长度与 $\sqrt{p}$成正比。当循环出现时，设 $x_j\equiv x_k(\mathrm{mod}\ p)$，记 $d=\gcd(x_j-x_k,n)$，则 d 是 p 的倍数。当 $d\neq n$时，d 就是 n 的一个非平凡因子。

但是在分解成功之前，p 是未知的，也就无从直接判断循环是否已经出现。仍然设迭代序列关于模 p 出现循环，$x_j\equiv x_k(\mathrm{mod}\ p)$并设 $j<k$。由取模运算的性质可知 $f(x_j)\equiv f(x_k)(\mathrm{mod}p)$，即 $x_{j+1}\equiv x_{k+1}(\mathrm{mod}p)$。故对任意 $c\geq 0$，总有 $x_{j+c}\equiv x_{k+c}(\mathrm{mod}p)$。记循环部分长度为 t，若 $t|k$，则 $t|2k$，那么 $x_{2k}\equiv x_k(\mathrm{mod}n)$。因此，只要对迭代序列中的偶数项 x_k 和对应的 $x_{\frac{k}{2}}$ 计算 $d=\gcd(x_k-x_{\frac{k}{2}},n)$。只要 $d\neq n$，d 就是 n 的一个非平凡因子。以上就是 Pollard 原来建议使用的 Floyd 循环判断算法。由此得到 Pollard ρ因数分解算法的第一个伪代码：

```
function pollard_floyd(n)
{
    x := x0
    y := x0
    d := 1
    while d == 1
    {
        x := f(x)
        y := f(f(y))
```

```
            d := gcd(x          − y, n)
            if 1 < d AND d < n then return d
            if d == n then return FAILURE
        }
}
```

Floyd 循环判断算法对储存整个迭代序列的空间要求很高，一般实现时都使用时间换空间的办法，同时计算 x_k 和 x_{2k} 来进行判断（如上面的伪代码）。Brent 提出了另外一种效率更高的循环判断算法：每步只计算 x_k，当 k 是 2 的方幂时，令 $y = x_k$；每一步都拿当前的 x_k 和 y 计算 $d = \gcd(x_k - y, n)$。伪代码如下：

```
function pollard_brent(n)
{
        x := x0
        y := x0
        k := 0
        i := 1
        d := 1
        while d == 1
        {
            k := k + 1
            x := f(x)
            d := gcd(x          − y, n)
            if 1 < d AND d < n then return d
            if d == n then return FAILURE
            if k == i then
            {
                y := x
                i := i << 1
            }
        }
}
```

由于循环出现的时空期望都是 $O(\sqrt{p})$，Pollard ρ因数分解算法的总体时间复杂度也就是 $O(\sqrt{p})$。在最坏情况下，其时间复杂度可能接近 $O(\sqrt{n})$，但在一般条件下，时间复杂度都可以认为是 $O(\sqrt[4]{n})$。

参考资料：

Chris Caldwell “Fermat, probable-primality and pseudoprimes.”

Chris Caldwell “Strong probable-primality and a practical test.”

Eric W. Weisstein “Brent’s Factorization Method.”

Eric W. Weisstein “Pollard Rho Factorization Method.”

Eric W. Weisstein “Rabin-Miller Strong Pseudoprime Test.”

Eric W. Weisstein “Strong Pseudoprime.”

Unknown Author “Pollard’s ρ.”