# Fermat's "Little" Theorem

Fermat's little theorem almost characterizes primes.

Theorem: Let $p$ be prime and $a$ be an integer that is not a multiple of $p$. Then $a^{p-1} \equiv 1 \pmod{p}$.

It is easy to evaluate $a^{p-1}$ mod $p$ because of
Fast Exponentiation

Input: A prime $p$ and integers $n \geq 0$ and $a$.
Output: The value $a^n$ mod $p$.

```
e = n
y = 1
z = a
while (e>0) {
       if (e is odd) y = (y * z) mod p
       z = (z * z) mod p
       e = e/2
       }
return y
```

In fast exponentiation, $a$ does not have to be an integer. In fact, the algorithm works when $a$ is anything that can be multiplied associatively, such as a matrix.

Fermat's little theorem can almost be used to find large primes. The theorem says that if $p$ is prime and $p$ does not divide $a$, then $a^{p-1} \equiv 1 \pmod{p}$. Thus, this theorem gives a test for *compositeness*: If $p$ is odd and $p$ does not divide $a$, and $a^{p-1} \not\equiv 1 \pmod{p}$, then $p$ is not prime.

If the converse of Fermat's theorem were true, it would give a fast test for *primality*. The converse would say, if $p$ is odd and $p$ does not divide $a$, and $a^{p-1} \equiv 1 \pmod{p}$, then $p$ is prime. This converse is not a true statement, although it is true for most $p$ and most $a$. If $p$ is a large random odd integer and $a$ is a random integer in $2 \le a \le p-2$, then the congruence $a^{p-1} \equiv 1 \pmod{p}$ almost certainly implies that $p$ is prime. However, there are more reliable tests for primality having the same complexity.

Definition: An odd positive integer $p > 2$ is called a **probable prime to base** $a$ if $a^{p-1} \equiv 1 \pmod{p}$. A composite probable prime to base $a$ is called a **pseudoprime to base** $a$.

If we knew all base $a$ pseudoprimes $< L$, then the following would form a correct primality test for odd integers $p < L$:

1. Compute $r = a^{p-1} \bmod p$.

2. If $r \neq 1$, then $p$ is composite.

3. If $p$ appears on the list of pseudoprimes $< L$, then $p$ is composite.

4. Otherwise, $p$ is prime.

Although this algorithm has occasionally been used, there are much better tests, some having the same complexity.

There are only three pseudoprimes to base 2 below 1000. The first one is $p = 341 = 11 \cdot 31$. By fast exponentiation or otherwise, one finds $2^{340} \equiv 1 \pmod{341}$. (Or check $2^{340} \equiv 1 \pmod{11}$ and $2^{340} \equiv 1 \pmod{31}$ and use the CRT.)

One difficulty with this test is that lists of pseudoprimes, to base 2, say, do not reach high enough to encompass the range of primes of cryptographic interest. A second problem is that there are too many pseudoprimes to any particular base; the list of all of them would be too long.

## A true converse of Fermat's little theorem

Theorem: Let $m > 1$ and $a$ be integers such that $a^{m-1} \equiv 1 \pmod{m}$, but $a^{(m-1)/p} \not\equiv 1 \pmod{m}$ for every prime $p$ dividing $m - 1$. Then $m$ is prime.

This theorem can be used to prove primeness of almost any prime $m$ for which we know the factorization of $m - 1$. If $m$ is an odd prime, then usually a small prime $a$ can be found quickly which will satisfy all the conditions. The principal difficulty in using the theorem to prove that a prime $m$ is prime is not the search for $a$, but rather finding the factorization of $m - 1$. If $m - 1$ has been factored, then one can use this simple algorithm to try to prove it is prime.

Lucas-Lehmer $m-1$ primality test

1. Choose $a = 2$ or choose a random $a$ in $2 \leq a \leq m-1$.

2. Compute $r = a^{m-1} \bmod m$.

3. If $r \neq 1$, then $m$ is composite.

4. Check that $a^{(m-1)/p} \not\equiv 1 \pmod{m}$ for each prime $p$ dividing $m-1$.

5. If all these incongruences are true, then $m$ has been proved prime.

6. If they are not satisfied, then either choose another $a$ (either the next small prime or a new random $2 \leq a \leq m-1$) and go back to Step 2, or else give up if many $a$ have already been tried.

If $m$ is a large prime, then the expected number of $a$ this algorithm must try before finding one that works is known to be $< 2 \ln \ln m$. If $m$ is a large composite, then the algorithm will almost certainly stop in Step 3.

If $m$ is proved prime by this algorithm, than $a$ is primitive root modulo $m$. That is, the smallest integer $e > 0$ with $a^e \equiv 1 \pmod{m}$ is $e = p - 1$. This is a good way to find a primitive root $a$ modulo a prime.

Many cryptographic algorithms require prime numbers of a certain size. If the prime need not be secret, then one can get one from a book or web site.

Every prime has a short, simple proof of its primality, but it is usually difficult to discover such a proof when the prime is large.

There are three ways to find large secret primes for cryptographic use.

1. Test random large numbers and choose the first probable prime. In other words, use "industrial-grade primes."

2. Test random large numbers for being probably prime. When you find one, prove rigorously that it is prime.

3. Use random numbers to construct a large prime having special form which permits an easy rigorous proof of its primality.

## Stronger Probable Prime Tests

Definition: An odd positive integer $n$, with $n-1 = 2^s d$, where $d$ is odd, is a **strong probable prime to base** $a$ if either $a^d \equiv 1 \pmod{n}$ or $a^{d\cdot 2^r} \equiv -1 \pmod{n}$ for some $0 \le r < s$. A **strong pseudoprime to base** $a$ is a composite strong probable prime to base $a$.

Every prime $p$ is a strong probable prime to every base $a$ it does not divide.

It is easy to see that every strong probable prime is a probable prime to the same base, because the definition says that we will get $\pm 1$ at some step before the last step in computing $a^{n-1} \bmod n$ by fast exponentiation, and this number will be squared at least once.

## Fibonacci Probable Prime Tests

Definition: The Fibonacci numbers are defined by $u_0 = 0$, $u_1 = 1$ and $u_{n+1} = u_n + u_{n-1}$ for $n \geq 1$.

The Fibonacci numbers are $u_2 = 1$, $u_3 = 2$, $u_4 = 3$, $u_5 = 5$, $u_6 = 8$, $u_7 = 13$, $u_8 = 21$, $u_9 = 34$, $u_{10} = 55$, ....

Theorem: If $n$ is prime, then $n$ divides $u_{n\pm1}$. Specifically, if $n \equiv 1$ or $9 \pmod{10}$, then $n$ divides $u_{n-1}$ and if $n \equiv 3$ or $7 \pmod{10}$, then $n$ divides $u_{n+1}$.

Examples:

Since $3 \equiv 3 \pmod{10}$, 3 divides $u_4 = 3$.

Since $7 \equiv 7 \pmod{10}$, 7 divides $u_8 = 21$.

Since $11 \equiv 1 \pmod{10}$, 11 divides $u_{10} = 55$.

There is a simple way to compute Fibonacci numbers using $2\times 2$ matrices. Define the Lucas numbers by $v_0 = 2$, $v_1 = 1$ and $v_{n+1} = v_n + v_{n-1}$ for $n \geq 1$. The first Lucas numbers are $v_2 = 3$, $v_3 = 4$, $v_4 = 7$, $v_5 = 11$, ....

Define $L = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$ and, for $n \geq 0$, $A_n = \begin{bmatrix} u_{n+1} & v_{n+1} \\ u_n & v_n \end{bmatrix}$. Then $A_0 = \begin{bmatrix} 1 & 1 \\ 0 & 2 \end{bmatrix}$. A simple induction shows that $A_n = L^n A_0$ for $n \geq 0$, where $L^0$ means the $2 \times 2$ identity matrix $I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$.

The formula $A_n = L^n A_0$ for $n \geq 0$, that is,

$$\begin{bmatrix} u_{n+1} & v_{n+1} \\ u_n & v_n \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n \begin{bmatrix} 1 & 1 \\ 0 & 2 \end{bmatrix},$$

is not just a pretty formula. It provides a quick way to compute $u_n$ and $v_n$ when $n$ is huge. The fast exponentiation algorithm given above applies to matrices. Thus we can compute $L^n$ in our formula with only $\mathrm{O}(\log n)$ matrix multiplications. If we wish to compute $u_n$ mod $m$ or $v_n$ mod $m$, we should reduce each matrix entry modulo $m$ as it is computed. This will keep the numbers small ($< m^2$) even if $n$ has hundreds of digits.

To evaluate $u_{n+1}$ mod $m$.

Input: Integers $n \geq 0$ and $m > 1$.
Output: The value $u_{n+1}$ mod $m$.

```
e = n
y = the matrix A_0 = [ 1 1 ; 0 2 ]
z = L, the matrix [ 1 1 ; 1 0 ]
while (e>0) {
       if (e is odd) y = (y * z) mod m
       z = (z * z) mod m
       e = e/2
       }
return y(1,1)
```

In 1980, Baillie and Wagstaff found that when $m$ is a composite number congruent to 3 or 7 (mod 10), then it seldom happens that $m$ divides $u_{m+1}$. A composite $m$ that divides $u_{m+1}$ is called a **Fibonacci pseudoprime**.

Pinch has computed the pseudoprimes to base 2 up to $10^{13}$. Not a single known strong pseudoprime to base 2 is also a Fibonacci pseudoprime. This may be because pseudoprimes to base 2 often have the form

$$(na+1)(nb+1)(nc+1)\cdots$$

while Fibonacci pseudoprimes $\equiv$ 3 or 7 (mod 10) often have the form

$$(na-1)(nb-1)(nc-1)\cdots.$$

Pomerance, Selfridge and Wagstaff made this conjecture.

**CONJECTURE** *An odd positive integer* $\equiv 3$ *or* $7 \pmod{10}$ *is prime if and only if it is both a strong probable prime to base 2 and a Fibonacci probable prime.*

In 1980, they offered \$30 for a proof or disproof of the conjecture, and have since raised this reward to \$620.

Cryptographers satisfied with "industrial-grade primes" should select strong probable primes to base 2 which are also Fibonacci probable primes, as in the Conjecture. The tests are simple, elegant and provide the added benefit that if you are the first to detect a failure of the conjecture, then you will collect \$620.

In 2002, the American National Standards Institute selected this algorithm for choosing industrial-grade primes for cryptography as ANSI Standard X9-80. Many implementations of the Secure Sockets Layer (SSL) choose large primes by the Baillie-Wagstaff method.

In 2003, M. Agrawal, N. Kayal and N. Saxena found a deterministic polynomial-time primality test for arbitrary positive integers. Although it runs in polynomial time, it is slower than a probable prime test like that of Baillie and Wagstaff.

Here is an example of the third way to construct a large prime for cryptographic use, namely, construct a prime with special form to permit an easy proof of its primeness.

**Theorem**. (Pocklington) Let $n$ be odd and $n-1 = FR$, where the complete factorization of $F$ is known. Suppose that for every prime $p$ dividing $F$ there is an integer $a$ such that $a^{n-1} \equiv 1 \pmod{n}$ and $\gcd(a^{(n-1)/p}-1, n) = 1$. Then every prime factor of $n$ is $\equiv 1 \pmod{F}$.

If also $F \geq \sqrt{n}$, then $n$ is prime.

This theorem allows us to construct a new prime with about twice as many digits as the previous one.

[Doubling the size of a random prime]

Input: A prime $p$.

Output: A prime $n$ near $p^2$.

repeat $\{$
  Let $k$ be a random integer between $p/2$ and $p$.
  $n = 2kp + 1$
  If $2^{n-1} \not\equiv 1 \pmod{n}$ restart this loop.
  Try to prove $n$ is prime via Pocklington's Thm.
  If you succeed, end the loop.
  $\}$ until $n$ is prime

By the prime number theorem, the expected number of iterations of the loop needed to find a prime $n$ is about $\ln p$.

In applying Pocklington's Theorem in the algorithm above, let $F = p$ and $R = 2k$. It may seem strange to put the known factor 2 into $R$, but it would take longer to check the hypotheses of Pocklington's Theorem if we put the 2 in $F$. For the integer $a$ of the theorem, try the ten primes $< 30$.

To construct a large prime near $X$, begin with a known prime near the $2^i$-th root of $X$, for some convenient $i$, and apply the algorithm $i$ times with the known prime as the first input, and each subsequent input equal to the previous output. Adjust $k$ in the final iteration of the loop to make the last $n$ just the right size.

The prime $p$ constructed by this algorithm will have a special form that permits an easy proof that it is prime. Specifically, $p-1$ will have a large prime factor $q \approx \sqrt{p}$ which will serve as the $F$ in Pocklington's Theorem and permit a quick application of the theorem. Also, $q-1$ will have a prime factor near its square root that lets us prove quickly that $q$ is prime, and so on down to a prime small enough to have an easy prime proof.