01204211 Discrete Mathematics
Lecture 17: Primality testing (2)

Jittat Fakcharoenphol

October 25, 2015

Efficient algorithms

This lecture proves that the two forms of the Fermat's Little Theorem are equivalent.
We then outline a primality testing algorithm based on the Fermat's Little Theorem. This algorithm requires two subroutines for

- computing powers a^n,
- finding the greatest common divisors.

We will present efficient algorithms for these two problems.

Fermat's Little Theorem

The common form of the Fermat's Little Theorem is as follows.

First form

Theorem: If p is a prime and a is an integer not divisible by p, then $a^{p-1} \mod p = 1$.

However, we prove the following version in the previous lecture:

Second form

Theorem: If p is a prime, then $p|a^p - a$, for any integer a.

We shall prove that both forms are equivalent.

Basic claim

We need the following claim to prove the equivalence.

Claim 1: If p is a prime and $p|ab$ but $p \not| a$, then $p|b$.

Proof: Factor ab into a product of primes. Since p divides ab then p must be one of the factors in the prime factorization of ab. However, since $p \not| a$, p does not appear in a's prime factorization. This implies that p must be from b's prime factorization. Hence, $p|b$. ■

Note that the assumption that p is a prime is crucial. For example, if we let $p = 6$, $a = 9$, and $b = 2$, we can see that $6|18$, but $6 \not| 9$ or $6 \not| 2$.

Another form for the Fermat's Little Theorem

To show that both forms of the Fermat's Little Theorem are equivalent, we need to prove the following two claims. (You should look at the claims and try to understand why proving both of them implies that both forms of the Fermat's Little Theorem are equivalent.)

Second form $\Rightarrow$ first form

Claim 2: If p is a prime, p does not divide an integer a, and $p|a^p - a$; then $a^{p-1} \mod p = 1$.

Proof: Write $a^p - a = a(a^{p-1} - 1)$. Since $p|a(a^{p-1} - 1)$ but $p \not| a$, from claim 1, we have that $p|a^{p-1} - 1$. Thus,

$$a^{p-1} \mod p = 1,$$

as required. ■

First form $\Rightarrow$ second form

Claim 3: Lat p be a prime. If for any integer a not divisible by p, $a^{p-1} \mod p = 1$, then for any integer b, we have that $p|b^p - b$.

Proof: There are two cases, when $p \not| b$ and $p|b$.
In the first case where $p \not| b$, the assumption implies that $p|b^{p-1} - 1$; thus, $p|b(b^{p-1} - 1)$, as required.
Now consider the case when $p|b$. Since we can write $b^p - b = b(b^{p-1} - 1)$, we know that $p|b(b^{p-1} - 1)$, because $p|b$. ■

The test: idea

If we want to check if q is a prime, from the Fermat's Little Theorem, we know that when q is a prime, for any integer a not divisible by q, $a^{q-1} \mod q = 1$. We can devise a test based on this fact:

PROCEDURE FermatTest0(q,a)
1. Find $y = a^{q-1} \mod q$
2. **if** $y \neq 1$ **then**
3. **return** "COMPOSITE"
4. **else**
5. **return** "PRIME"
6. **endif**

To implement this test efficiently, we need a way to find $a^{q-1} \mod q$ quickly. Recall that in our test, q will be a very large number and an $O(q)$-time algorithm or even $O(\sqrt{q})$-time algorithm will not be fast enough.

Computing powers

If we want to find $a^n \mod q$, a straight-forward implementation is to perform n multiplications. When a and q are m-bit numbers, this takes $O(nm^2)$ time because multiplication and division run in $O(m^2)$ time.

Can we compute the power faster?

Let's start with some example.

- How to compute a^2? We can't do anything faster then just computing directly a^2.
- What can we compute if we know a^2? Note that we can square a^2 to get $a^2 \times a^2 = a^4$.
- We can keep squaring to get $a^8 = a^4 \times a^4$, $a^{16} = a^8 \times a^8$, and so on.
- Thus, we can compute a^{2^k} with only k multiplications.

We know how to compute the power a^n when n is a power of 2.

We still need to handle the case when the exponent n is not a power of 2.

To deal with general case, we can either keep all computed powers and multiply appropriate results to get the final answer. For example, to compute a^{20}, we note that $a^{20} = a^{16} \cdot a^4$, so we first compute a^2, a^4, a^8, a^{16} and multiply a^{16} and a^4.

A recursive algorithm below takes another approach. It first finds $a^{\lfloor n/2 \rfloor}$, where $\lfloor x \rfloor$ is a floor function that returns the largest integer not greater than x (e.g., $\lfloor 1.2 \rfloor = 1$ and $\lfloor 2 \rfloor = 2$). It then squares the result and depending on the value of $n/2$, multiplies the result with a.

PROCEDURE Power(a,n)
1. **if** $n = 1$ **then return** a **endif**
2. **let** $b \leftarrow$ Power(a,$\lfloor n/2 \rfloor$)
3. **if** $2|n$ **then**
4. **return** $b \times b$
5. **else**
6. **return** $b \times b \times a$
7. **endif**

The number of multiplications

Let's analyze the number of multiplications required to compute a^n.

The running time

The greatest common divisors

The test: idea + more condition

If q is a prime and a is chosen so that $2 \leq a \leq q-1$, we know that $gcd(q, a) = 1$. We shall add this condition to the test as well. While this check might not be that helpful in itself when q is composite, it is crucial to our proof that the algorithm works (described in the next lecture).

PROCEDURE FermatTest(q,a)
1. **if** $GCD(q, a) \neq 1$ **then return** "COMPOSITE" **endif**
2. Find $y = a^{q-1} \mod q$
3. **if** $y \neq 1$ **then**
4. **return** "COMPOSITE"
5. **else**
6. **return** "PRIME"
7. **endif**

Finding the greatest common divisors

The Euclidean algorithm

An example

The running time