Advanced Algorithms	**Lecture 4 : March 15, 1995**
Lecturer: Johan Håstad	*Scribe: Claes Thornberg*

4.1 Some preliminaries

Given a prime number p, is there a number g (*generator*) such that the series $g, g^2, g^3, \dots, g^{p-1} \pmod p$ includes all numbers $1, \dots, p-1$? If such a number g exists, we say that the multiplicative group $\mathbb{Z}_p^*$ is cyclic.

Example 1 *The multiplicative group with $p = 13$ is generated by $g = 2$.*

$$2, 4, 8, 16 \equiv 3, 6, 12, 24 \equiv 11, 22 \equiv 9, 18 \equiv 5, 10, 20 \equiv 7, 14 \equiv 1.$$

With $g = 4$, only a subgroup is generated.

$$4, 16 \equiv 3, 12, 48 \equiv 9, 36 \equiv 10, 40 \equiv 1.$$

There are many such numbers g, but there is no known way to find such a number in deterministic polynomial time. Even worse, even testing whether a given number is a generator is not known to be deterministic polynomiaal time.

The multiplicative group (mod p), $\mathbb{Z}_p^*$, is isomorphic with the additive group (mod $p-1$), $\mathbb{Z}_{p-1}$. The mapping is $g^x \in \mathbb{Z}_p^* \mapsto x \in \mathbb{Z}_{p-1}$. Furthermore, we can see that $\mathbb{Z}_{p-1}$ is generated by any number a such that $gcd(a, p-1) = 1$. Given such an $a \in \mathbb{Z}_{p-1}$, we know that g^a is a generator of $\mathbb{Z}_p^*$ if g also is a generator of $\mathbb{Z}_p^*$.

Example 2 *The generators of $\mathbb{Z}_{12}$ are $1, 5, 7$ and 11. We know, from example 1, that 2 is a generator of $\mathbb{Z}_{13}^*$. Therefore, the generators of $\mathbb{Z}_{13}^*$ are $2^1 \equiv 2, 2^5 \equiv 6, 2^7 \equiv 11$ and $2^{11} \equiv 7$. Had we known that 11 was a generator, we would also have gotten all the other generators of $\mathbb{Z}_{13}^*$, $11^1 \equiv 11, 11^5 \equiv 7, 11^7 \equiv 2$ and $11^{11} \equiv 6$.*

4.2 Introduction

The problem of finding the discrete logarithm can be formulated as follows: Given numbers y, g and p, where p is prime and g a generator of $\mathbb{Z}_p^*$, find x such that $g^x \equiv y \pmod p$. This is called the discrete logarithm of y base g modulo p.

Example 3 *The discrete logarithm of* 11 *(base* 2 *modulo* 13*) is* 7*. This can be seen in example 1.*

4.2.1 A digression

To give some clue of the usefulness of computing discrete logarithms, consider the Diffie-Hellman key-exchange algorithm from cryptography. There we have A and B who wants to communicate over an open link using a secret key K. They must use the link to agree on what key to use.

First a prime p and a number g is agreed upon and transmitted on the link. Then A transmits g^a, where a is a random number, to B, and B transmits g^b to A for some random number b. The key K is then formed by A as $K = (g^b)^a = g^{ab}$, and by B as $K = (g^a)^b = g^{ab}$. An eavesdropper having only g^a and g^b is at a disadvantage and the only known method to compute K from this information is to compute the discrete logarithm of at least one of them.

4.3 A naive algorithm

A naive algorithm for computing the discrete logarithm of y (base g, modulo p), is to generate $g, g^2, \ldots$ until we get y. However, this is very inefficient, its worst case complexity being $O(2^n)$ if $p \sim 2^n$. That is, in the worst case we have to generate all the powers $g, g^2, \ldots, g^{p-1}$.

4.4 The baby step/giant step algorithm

A somewhat better algorithm for computing the discrete logarithm is the following.

- Let $a = \lceil \sqrt{p} \rceil$
- Compute $L_1 = 1, g^a, g^{2a}, g^{3a}, \ldots, g^{a^2} \sim g^p$
- Compute $L_2 = y, yg, yg^2, yg^3, \ldots, yg^{a-1}$
- Look for a number z which appears in both L_1 and L_2. We then have

$$z = yg^k \quad = \quad g^{la}$$

$$\begin{aligned} y &= g^{la-k} = g^x \\ x &= la-k \end{aligned}$$

However, two problems remain to be solved.

1. Is there always such a number z in both lists?
2. How to find z efficiently?

The answer to the first question is yes, since we can write $x = x_1 a + x_2$ where $0 \le x_1, x_2 < a$. We then have $y = g^x = g^{x_1 a + x_2}$. We multiply both sides of the equation by g^{a-x_2}, and get $yg^{a-x_2} = g^{(x_1+1)a}$. The left hand side of this equation is found in L_2, and the right hand side in L_1.

The answer to the second question is also quite simple. Sort the two lists in time $O(a \log a)$, then proceed as follows: compare the heads of the two lists. If they are equal, we have found z and we stop, otherwise we throw away the smallest of the two front elements and continue. An alternative way is to store L_1 in a hash table, then go through L_2. With a good hash table, this can be done in time $O(a)$. Hence, the overall complexity is $O(\sqrt{p})$.

Example 4 *To solve the equation*

$$2^x \equiv 11 \pmod{13}$$

using the algorithm above, we set $a = \lceil \sqrt{13} \rceil = 4$ *and compute*

$$L_1 = 1, 2^4 = 3, 2^8 = 9, 2^{12} = 1, 2^{16} = 3$$

$$L_2 = 11, 11 \cdot 2 = 9, 11 \cdot 2^2 = 5, 11 \cdot 2^3 = 10$$

In the two lists we have $11 \cdot 2 = 2^8 = 9$*, which gives us* $11 = 2^7$*. Thus, the logarithm of* 11 *(base* 2 *modulo* 13*) is* 7*.*

4.5 A better algorithm

Suppose $p-1$ can be factorized in "small" primes, i.e.

$$p - 1 = \prod_{i=1}^{s} q_i^{\varepsilon_i}.$$

We assume that $q_1 < \ldots < q_s$ and that $\varepsilon_i = 1, 1 \leq i \leq s$. We can then find (all) q_i efficiently by Pollard's ϱ-test or by using elliptical curves. This takes $\sim \sqrt{q_s}$. To proceed, we compute $x \bmod q_i$ for $i = 1, \ldots, s$. For q_1 we have

$$(g^x)^{\prod_{i=2}^s q_i} = y^{\prod_{i=2}^s q_i} \stackrel{\text{def}}{=} y_1.$$

The right hand side of which we can compute quite easily. Furhtermore, since we know that $x = a_1 q_1 + r_1$, we have

$$\begin{aligned}
(g^x)^{\prod_{i=2}^s q_i} &= (g^{a_1 q_1 + r_1})^{\prod_{i=2}^s q_i} = \\
g^{a_1 \prod_{i=1}^s q_i + r_1 \prod_{i=2}^s q_i} &= g^{a_1 (p-1) + r_1 \prod_{i=2}^s q_i} = \\
g^{r_1 \prod_{i=2}^s q_i} &= (g^{\prod_{i=2}^s q_i})^{r_1} \stackrel{\text{def}}{=} g_1^{r_1}.
\end{aligned}$$

Since g_1 generates a subgroups of order q_1 of $\mathbb{Z}_p^*$, ${g_1}^{q_1} = (g^{\prod_{i=2}^s q_i})^{q_1} = g^{\prod_{i=1}^s q_i} = g^{p-1} = 1$, all we have to do in order to compute $x \bmod q_1$ is to solve ${g_1}^{r_1} = y_1$. We can do this in time $\sim \sqrt{q_1}$ by using the algorithm in section 4.4 with $a = \lceil \sqrt{q_1} \rceil$.

In the same way, we can determine $x \bmod q_i$ for all q_i in time $\sum_{i=1}^s \sqrt{q_i} \sim \sqrt{q_s}$. The Chinese remainder theorem, then gives us $x \bmod \prod_{i=1}^s q_i = x \bmod (p-1)$.

4.5.1 Chinese remainder theorem in practice

Given $x \bmod p_1$ and $x \bmod p_2$, find $x \bmod p_1 p_2$. We have $\gcd(p_1, p_2) = 1$. If we can do this, we can easily do it for any number of factors, all of which are relatively prime.

The problem can be seen as: given

$$\begin{aligned}
x &\equiv a_1 \bmod p_1 \\
x &\equiv a_2 \bmod p_2
\end{aligned}$$

find

$$\begin{aligned}
U_1 &\equiv 1 \bmod p_1 \\
U_1 &\equiv 0 \bmod p_2 \\
U_2 &\equiv 0 \bmod p_1 \\
U_2 &\equiv 1 \bmod p_2.
\end{aligned}$$

We then have

$$x \equiv a_1 U_1 + a_2 U_2 \bmod p_1 p_2,$$

which fulfills the equations above.

One question remains: how do we find U_1 and U_2? Euclid's algorithm on (p_1, p_2) gives d as result, Euclid's extended algorithm gives $d = ap_1 + bp_2$. With $d = 1$ we have $1 = ap_1 + bp_2$, and we identify $U_1 = bp_2$ and $U_2 = ap_1$.

Example 5 *Let's say we have the following equations:*

$$\begin{aligned} x &\equiv 4 \bmod 23 \\ x &\equiv 7 \bmod 15. \end{aligned}$$

Euclid's extended algorithm gives us

$$1 = 2 \cdot 23 - 3 \cdot 15.$$

This means that

$$\begin{aligned} U_1 &= -45 \\ U_2 &= 46 \end{aligned}$$

and that

$$x = 4 \cdot (-45) + 7 \cdot 46 \bmod 23 \cdot 15 = 142 \bmod 345.$$

That any number congruent with 142 *satisfies the above equations can be tested quite easily.*

Euclid's extended algorithm is approximately as fast as the normal one.

Conclusion: Discrete logarithms (mod p) kan be computed in time $\sim (\log p)^3 \sqrt{q_s}$, where q_s is the largest prime factor in $p - 1$.

4.6 An expected better algorithm

The expected running time of this algorithm is better than for any of the previous algorithms. The idea is to take a random r_i, compute g^{r_i} and hope that $g^{r_i} = \prod_{q<B} q^{a_{i,q}}$, where all q are prime and B an upper limit on the size of the primes. This yields the equation

$$r_i = \sum_{q<B} a_{i,q} \log q,$$

where all logarithms are base g, modulo p. Collect many such relations and solve the equation system which gives us $\log q$ for all $q < B$. Pick a new random number r, compute yg^r and hope that it can be factorized so that

$$yg^r = \prod_{q<B} q^{b_q}.$$

We now have the solution as

$$x = \log y = \sum_{q<B} b_q \log q - r.$$

The time to generate all the equation needed is

$$Pr[\text{numbers} \sim p \text{ only have factors} \leq B]^{-1} B^2.$$

To solve an equation system with $\sim B$ unknown can be done in time $\sim B^3$. The algorithm works best with $B \sim 2^{c\sqrt{\log p \log\log p}}$, which gives an overall cost of $2^{c'\sqrt{\log p \log\log p}}$.