6.885 Algebra and Computation September 26, 2005

Lecture 5

Lecturer: Madhu Sudan *Scribe: Elena Grigorescu*

In today's lecture we will first go through a brief response to the comments on the previous lecture. The comments and the brief responses will be posted soon. We will continue by introducing methods of polynomial factorization, in particular root finding algorithms in finite fields.

1 A review of last lecture

In the last lecture we defined Strong Generating Sets (SGS) in a permutation group and we showed an algorithm for building a small-size ($O(n^2)$) SGS of a group $G =< S >$. We then demonstrated an efficient algorithm for testing membership in a permutation group, which was based on the idea that if $\sigma \in G$ then σ is the product of polynomially many elements of a SGS of G.

The motivation for looking at these rather non-intuitive generating sets is that they give a good representation of a group, which in turn leads to good and sometimes unexpected algorithms (membership seems rather exponential but it turns out not be). Generally, algebraic algorithms exhibit a couple of paradigms that we have already encountered in the concept of SGSs:

1. non-intuitive definition
2. given an object meeting the definition the problem is easy to solve
3. an object meeting the definition exists
4. objects meeting the definition can be constructed efficiently.

It is often the case that for the last two paradigms the proofs are very different.

1.1 Summary of the membership testing algorithm

Let π be a permutation and let $G =< S >$ and $G \leq S_n$.

BUILD SGS(S):
$T \leftarrow \emptyset$;
while $\sigma \in S \cup T \cdot T$ s.t. not MEM-CLOSURE(σ, T)
    $T \leftarrow$ ADD-ELEM(σ, T) $\cup T$
return T.

ADD-ELEM(σ, T):
if $\sigma = 1$ then return σ
else let $k = k(\sigma); j = \sigma^{-1}(k)$;
    if $\exists \sigma' \in T$ s.t $k = k(\sigma'); \sigma'(j) = k$
    return ADD-ELEM$(\sigma \cdot \sigma'^{-1}, T)$
    else return σ.

MEM-CLOSURE$(\sigma, T$:)
if ADD-ELEM$(\sigma, T) = 1$ return TRUE
else return FALSE.

MEM(π, S:)
$T \leftarrow$BUILD-SGS(S)
return MEM-CLOSURE(π, T).

We can now move on to a new topic.

2 Factorization of polynomials - a very simple scenario

INPUT: Integers a, p, with p prime;
GOAL: Compute $\alpha \in \mathbb{Z}_p$ st $\alpha^2 = a \bmod p$, if α exists.

We are first interested in whether α exists, and we will show that the following test decides the existence of a root.

Test: If $a^{\frac{p-1}{2}} \equiv 1$ then a has a root; else it does not.

Proof: Suppose a has a root, and $\alpha^2 = a$. By Fermat's Little Theorem we have that $\forall \alpha \in \mathbb{Z}_p^*$, $\alpha^{p-1} \equiv 1 \bmod p$. Therefore, $\alpha^{p-1} \equiv a^{\frac{p-1}{2}} \equiv 1$.
In proving the converse, we introduce ideas that will be useful in finding roots of polynomials in more general settings. Suppose that $a^{\frac{p-1}{2}} \equiv 1 \bmod p$. We want to show that a has a root.
Let $p(x) = x^{p-1} - 1$, $p_1(x) = x^{\frac{p-1}{2}} - 1$ and $p_2(x) = x^{\frac{p-1}{2}} + 1$. Thus $p(x) = p_1(x)p_2(x)$. We know that each $\alpha \in \mathbb{Z}_p^*$ is a unique root of $p(x)$ and thus it is a unique root of either p_1 or p_2. Since for any square $b \in Z_p^*$, $x^2 - b$ has exactly 2 roots in $\mathbb{Z}_p^*$, which are distinct, it follows that exactly $\frac{p-1}{2}$ elements in $\mathbb{Z}_p^*$ are squares. We also know that each of these squares are roots of $p_1(x)$, and since the degree of $p_1(x)$ is $\frac{p-1}{2}$, it follows that the squares in $\mathbb{Z}_p^*$ are the only roots of $p_1(x)$. This concludes the proof.

We can now try and find the square roots of a when they exist. When p is an odd prime we distinguish two cases, namely $p \equiv 3 \bmod 4$ for which we will give a deterministic algorithm, and $p \equiv 1 \bmod 4$ for which we give a randomized algorithm that determines the roots.

Case 1: $p \equiv 3 \bmod 4$
As before, let $p(x) = x^{p-1} - 1 = p_1(x)p_2(x)$, $p_1(x) = x^{\frac{p-1}{2}} - 1$ and $p_2(x) = x^{\frac{p-1}{2}} + 1$. We claim that
$x - \alpha \mid p_1(x)$, and
$x + \alpha \mid p_2(x)$.
To see this, notice that since $\frac{p-1}{2}$ is odd, we have that $\alpha^{\frac{p-1}{2}}$ and $(-\alpha)^{\frac{p-1}{2}}$ have different signs. Thus, each of them are roots of only one of $p_1(x)$ and $p_2(x)$, which proves the claim.
From the argument above we can deduce that $gcd(x^2 - a, x^{\frac{p-1}{2}} - 1) = x - \alpha$. Therefore, in order to compute α we only need to compute the above gcd, which only takes polynomial time in the smallest degree $(= 2)$.

Case 2: $p \equiv 1 \bmod 4$
Consider the more general setting of finding roots of a polynomial of the form $g(x) = x^2 - Ax - B = (x-\beta)(x-\gamma)$. If β and γ are random and independent elements of the field, then the $gcd(g(x), x^{\frac{p-1}{2}} - 1)$ is of degree 1 when only one of β and γ are roots of $p_1(x)$. Since the size of the field is p we can conclude that the probability of finding a linear factor of $g(x)$ is about $1/2$.

We would now like to be able to map α and $-\alpha$ into β and γ respectively, such that we could apply this approach to finding the roots of $a \in \mathbb{Z}_p^*$. Notice that there exist c and d s.t. $c\alpha + d = \beta$ and $c(-\alpha) + d = \gamma$. To make β and γ random and independent, we only need to pick $c \neq 0$ and d randomly and independently of each other. We can deduce that $x^2 - Ax - B = (x - \beta)(x - \gamma) = (x - (c\alpha + d))(x - (c(-\alpha) + d)) = (x - d)^2 - c^2 a$.

The promised algorithm follows.

SQ-ROOT(p, a):
1. If $a = 0$ return 0; If $a^{\frac{p-1}{2}} \equiv -1$ return 'none exists';
2. Pick $c \in \mathbb{Z}_p^*$ and $d \in \mathbb{Z}_p$ at random
3. Compute $q(x) = gcd((x-d)^2 - c^2 a, x^{\frac{p-1}{2}} - 1)$
4. If $q(x) = x - \beta$ return $\pm\frac{\beta - d}{c}$;
5. Else go to 2.

As before,

$$Pr_{c\in\mathbb{Z}_p^*,d\in\mathbb{Z}_p}[gcd((x-d)^2-c^2a, x^{\frac{p-1}{2}}-1) \text{ is linear }] =$$
$$Pr_{c\in\mathbb{Z}_p^*,d\in\mathbb{Z}_p}[gcd((x-(c\alpha+d))(x-(c(-\alpha)+d), x^{\frac{p-1}{2}}-1) \text{ is linear }] \approx 1/2,$$

since $c\alpha+d$ and $c(-\alpha)+d$ are random and distinct elements of $\mathbb{Z}_p$. By repeating the algorithm, the success probability can be made arbitrarily large, which concludes the analysis.

3 Generalization - Roots modulo p of higher degree polynomials

INPUT: p prime, $c_0, c_1, \ldots, c_n \in \mathbb{Z}_p$, $f(x) = \sum c_i x^i$.
GOAL: Find all α s.t. $x-\alpha \mid f(x)$.

We can assume W.l.o.g. that $f(x)$ has distinct linear factors. Otherwise, given a root α one could check in various ways if $(x-\alpha)^2 \mid f(x)$. As an exercise, show that if $(x-\alpha^2) \mid f(x)$ then $x-\alpha \mid gcd(f, f')$ (note that the derivative $f' = \sum i c_i x^{i-1} \bmod p$ is a formal definition).

From now on assume $f(x) = \sum c_i x^i = \Pi(x-\alpha_i)$ is st. $\alpha_i's$ are distinct.

Notice that if α is a root of $f(x)$ then it is also a root of $x^p - x$, and thus, w.p. $\approx 1/2$, α is a root of $p_1(x) = x^{\frac{p-1}{2}} - 1$. The goal of splitting $f(x)$ leads us into considering $q(x) = gcd(f(x), x^{\frac{p-1}{2}} - 1)$. However, this approach will work only if not all of the roots of $x^{\frac{p-1}{2}} - 1$ are also roots of f. The solution to this is that we can use randomization as we did in the square-root algorithm. Thus, map α_i into random β_i. We might not be able to get the independence of the β_i's, but we are only interested in splitting the polynomial once. Therefore, it is enough to focus only on random β_1, and β_2. With random c, d we could map $f(x) = \Pi(x-\alpha_i)$ into $\tilde{f}(x) = \Pi(x-(c\alpha_i+d))$. Then $\tilde{f}(x) = \Pi(x-(c\alpha_i+d)) = c^n \Pi(\frac{x-d}{c} - \alpha_i) = c^n f(\frac{x-d}{c})$. If we found a root of $\tilde{f}$ we can therefore find a root of f.

We are now ready to present the algorithm.

ROOT-FIND($f(x) = \sum c_i x^i$**):**
1. $f(x) \leftarrow gcd(x^p - x, f(x))$
2. Pick $c \in \mathbb{Z}_p^*, d \in \mathbb{Z}_p$ at random
3. $\tilde{f}(x) \leftarrow c^n f(\frac{x-d}{c})$
4. If $\tilde{q}(x) = gcd(\tilde{f}(x), x^{\frac{p-1}{2}} - 1)$ is non-trivial (not n)
   return ROOT-FIND($q(x) = \frac{1}{c^n}\tilde{q}(cx+d)$), ROOT-FIND ($\frac{f(x)}{q(x)}$).

Notice that the above algorithm only uses the fact that p is odd, so p does not necessarily need to be prime. The algorithm is due to Berlekamp, 1972 and it is probably the first algorithm in the literature that uses randomization. No deterministic such algorithm is known of.

In future lectures we will be dealing with the factorization of polynomials over even fields and with the factorization of polynomials into irreducible polynomials of degree higher than linear.

We conclude the lecture with the following exercise.
Given $\beta_1, \ldots, \beta_n \in \mathbb{Z}_p$, define the k'th symmetric function in $\{\beta_1, \ldots, \beta_n\}$ as

$$\sigma(\beta_1, \ldots, \beta_n) = \sum_{S\subset[n],|S|=k} \Pi_{i\in S}\beta_i.$$

Compute $\sigma(\beta_1, \ldots, \beta_n)$ efficiently.