Computational Number Theory and Algebra May 9, 2012

Lecture 7

Lecturers: Markus Bläser, Chandan Saha *Scribe: Chandan Saha*

In the last class, we mentioned that the extended Euclidean gcd algorithm for polynomials uses $O(\mathsf{M}(n)\log n)$ operations over the underlying field $\mathbb{F}$. When $\mathbb{F}$ is a finite field, say $\mathbb{F}_p$, then any operation in $\mathbb{F}_p$ takes at most $O(\log^2 p)$ bit operations - so, overall the gcd algorithm takes polynomially many bit operations. But, what if the underlying field is $\mathbb{Q}$? How do we show that the intermediate rational numbers generated during gcd computation have 'small' numerators and denominators? Well, we need not have to show this... and this is where the resultant comes to our rescue: The idea is to do the gcd computations modulo small primes (which is essentially like gcd computation over finite fields), and then finally compose the modular gcds using Chinese remaindering to obtain the gcd over $\mathbb{Q}$. The topics of discussion for today's class are:

- Modular gcd computation - an application of the resultant,
- Introduction to polynomial factoring over finite fields.

1 Modular gcd computation

We will show how to compute gcd of two polynomials in $\mathbb{Z}[x]$ in polynomial time (without proving that the intermediate rational coefficients have small numerators and denominators). Let $f = \sum_{i=0}^{n} f_i x^i$ and $g = \sum_{i=0}^{m} g_i x^i$ be two polynomials in $\mathbb{Z}[x]$ with $\deg(f) = n$ and $\deg(g) = m$, i.e. $f_n, g_m \neq 0$. For simplicity of exposition, we will assume that the polynomials f and g are monic, i.e., $f_n = g_m = 1$. We want to compute $h = \gcd(f, g)$, which is also a monic polynomial in $\mathbb{Z}[x]$ (why? see exercise 1). (The assumption that f_n and g_m are monic is not particularly binding, but it will simplify our discussion in a way. You may refer to chapter 6 of [GG03] to see how to remove this assumption.)

First, we need to show that any factor $\hat{f} \in \mathbb{Z}[x]$ of f has coefficients whose absolute values are 'polynomially' bounded in the input size. For any polynomial $s \in \mathbb{Z}[x]$, we denote the maximum among the absolute values of the coefficients of s by A_s.

Lemma 1 *$A_{\hat{f}} \leq (2nA_f)^n$.*

Proof By the fundamental theorem of algebra, f factorizes completely over complex numbers. Let, $f = \prod_{i=1}^{n}(x - a_i)$, where $a_i \in \mathbb{C}$. For any root a_i, since $a_i^n = -\sum_{j=0}^{n-1} f_j a_i^j$, hence $|a_i| \leq nA_f$. Any factor $\hat{f}$ of f is a product of the form $\prod_{i \in S}(x - a_i)$ for some subset $S \subseteq \{1, \ldots n\}$. Therefore, $A_{\hat{f}} \leq 2^{|S|}(nA_f)^{|S|}$. ■

Remark - Although, the value of $A_{\hat{f}}$ can be exponentially larger than A_f, its bit length is bounded by $O(n \log nA_f)$, which is polynomial in the input size $\Omega(n + \log A_f)$. (In our problem, the input is n coefficients of the polynomial f and m coefficients of the polynomial g.)

Observe that $\gcd(f/h, g/h) = 1$. Hence, $r \stackrel{\text{def}}{=} \mathsf{Res}(f/h, g/h) \neq 0$. We need the following bound on the absolute value of $\mathsf{Res}(f/h, g/h)$ (the proof follows easily from Lemma 1).

Observation 2 *If $A = \max_{i,j}\{|f_i|, |g_j|\}$ then $|\mathsf{Res}(f/h, g/h)| \leq (n+m)^{n+m}(2nA)^{n+m}$.*

In fact, using Hadamard inequality (Lemma 3 in Lecture 3), one can prove a better bound (see exercise 2).

Let $B = \lceil \log((n+m)^{n+m}(2nA)^{n+m}) \rceil$. Therefore, the number of primes dividing $r = \mathsf{Res}(f/h, g/h)$ is at most $B = \mathsf{poly}(n, m, \log A)$. For any polynomial $s \in \mathbb{Z}[x]$ and a fixed prime p, denote by $\bar{s}$, the polynomial $s \mod p$ (basically, reducing every coefficient of s modulo p). Naturally, $\bar{s}$ is a polynomial in $\mathbb{F}_p[x]$.

Lemma 3 *If the prime $p \nmid r$ then* $\gcd(\bar{f}, \bar{g})$ *in* $\mathbb{F}_p[x]$ *is equal to* $\bar{h}$. *If* $p|r$ *then* $\deg(\gcd(\bar{f}, \bar{g}))$ *is greater than* $\deg(\bar{h})$.

Proof Let $f_1 = f/h$ and $g_1 = g/h$ where $f_1, g_1 \in \mathbb{Z}[x]$ (see excercise 1). Then, $f = hf_1 \Rightarrow \bar{f} = \bar{h} \cdot \bar{f}_1 \mod p$. Similarly, $\bar{g} = \bar{h} \cdot \bar{g_1} \mod p$. Therefore, $\bar{h} | \gcd(\bar{f}, \bar{g})$ over $\mathbb{F}_p$. The polynomial $\bar{h}$ would be equal to $\gcd(\bar{f}, \bar{g})$ if the polynomials $\bar{f_1}$ and $\bar{g_1}$ do not share any factor over $\mathbb{F}_p$. Polynomials $\bar{f_1}$ and $\bar{g_1}$ share a factor over $\mathbb{F}_p$ if and only if $\mathsf{Res}(\bar{f_1}, \bar{g_1}) = 0$ over $\mathbb{F}_p$. Now observe that since f, g are monic, f_1, g_1 are also monic, and hence $\mathsf{Res}(\bar{f_1}, \bar{g_1})$ over $\mathbb{F}_p$ is equal to $r \mod p$. If $p \nmid r$ then $\mathsf{Res}(\bar{f_1}, \bar{g_1}) \neq 0$ over $\mathbb{F}_p$, and therefore $\gcd(\bar{f}, \bar{g}) = \bar{h}$. On the other hand, if $p|r$ then $\mathsf{Res}(\bar{f_1}, \bar{g_1}) = 0$ over $\mathbb{F}_p$, implying that $\bar{f_1}$ and $\bar{g_1}$ share a common factor over $\mathbb{F}_p$, and therefore $\deg(\gcd(\bar{f}, \bar{g})) > \deg(\bar{h})$. ■

We call a prime p, a *good* prime if $p \nmid r$, otherwise it's a *bad* prime. We have argued before that the number of bad primes is bounded by $B = \mathsf{poly}(n, m, \log A)$. By lemma 3, if p is a good prime then $\gcd(\bar{f}, \bar{g})$ over $\mathbb{F}_p$ is exactly $\bar{h} = h \mod p$. Further, by lemma 1, the absolute values of the integer coefficients of h is bounded by $C = (2nA)^n$. Hence, by choosing $\lceil \log(2C + 1) \rceil$ many good primes, we should be able to reconstruct $h \in \mathbb{Z}[x]$ using Chinese remaindering theorem (CRT) - (this argument is similar in spirit to the determinant computation algorithm using CRT from lecture 3). This means, if we choose $\ell = B + \lceil \log(2C + 1) \rceil$ primes $p_1, \ldots, p_\ell$ then at least $\lceil \log(2C + 1) \rceil$ of them are good primes. Denote by $\bar{h}_i$, the polynomial $h \mod p_i$. From lemma 3, if p_i is a good prime and p_j is a bad prime then $\deg(\bar{h}_i) < \deg(\bar{h}_j)$. This suggests the following algorithm, where $A = \max_{i,j}\{|f_i|, |g_j|\}$, $B = \mathsf{poly}(n, m, \log A)$ and $C = (2nA)^n$.

Algorithm 1 Modular gcd computation

1. `Let` $\ell = B + \lceil \log(2C + 1) \rceil$ `, and` $p_1, \ldots, p_\ell$ `be the first` ℓ `primes.`
2. `Let` $\bar{f}_i = f \mod p_i$ `and` $\bar{g}_i = g \mod p_i$ `, for` $1 \leq i \leq \ell$.
3. `Compute` $\bar{h}_i = \gcd(\bar{f}_i, \bar{g}_i)$ `over` $\mathbb{F}_{p_i}$ `for all` $1 \leq i \leq \ell$.
4. `Let` $d = \min_{1 \leq i \leq \ell}\{\deg(\bar{h}_i)\}$. `Let` $S = \{p_j : \deg(\bar{h}_j) = d\}$.
5. `Construct` $h = \gcd(f, g)$ `from the polynomials` $\{\bar{h}_j : j \in S\}$ `using CRT.`

Time complexity - It follows from the above discussion, and from the fact that CRT runs in polynomial time (see analysis of Algorithm 3 in lecture 3), that the bit complexity of the above algorithm is bounded by a 'certain' polynomial in n, m and $\log A$. We leave it as an exercise to find a precise expression for this 'certain' polynomial in order notation.

2 Polynomial factoring over finite fields

The problem of computing all the irreducible factors of a given polynomial over a finite field, is one of the most fundamental problems is algebraic computation. We have seen an application of bi-variate polynomial factoring in the list decoding algorithm of Reed-Solomon codes in lecture 2. It turns out that the problem of bi-variate factoring reduces in polynomial time to the univariate factoring problem - we will see this reduction in a later lecture. At first, we need to understand the complexity of the univariate factoring problem:

Problem 4 (Polynomial factoring) *Given the $n + 1$ coefficients of a degree n univariate polynomial $f(x) \in \mathbb{F}_q[x]$, where $\mathbb{F}_q$ is the finite field with q elements, find all the irreducible factors of f over $\mathbb{F}_q$.*

Representation of a finite field $\mathbb{F}_q$- Let p be the *characteristic* [1] of the finite field $\mathbb{F}_q$, i.e. $q = p^m$ for a prime p and $m \in \mathbb{N}$. A finite field $\mathbb{F}_q$ is represented by picking an irreducible polynomial $h(x) \in \mathbb{F}_p[x]$ of degree m. The ring $\mathbb{F}_p[x]/(h(x))$ is the finite field $\mathbb{F}_q$. (For two different degree-m irreducible polynomials h_1 and h_2 in $\mathbb{F}_p[x]$, the rings (actually fields) $\mathbb{F}_p[x]/(h_1(x))$ and $\mathbb{F}_p[x]/(h_2(x))$ are isomorphic.) An element of the ring $\mathbb{F}_p[x]/(h(x)) \cong \mathbb{F}_q$ is a polynomial over $\mathbb{F}_p$ of degree less than m.

[1] the characteristic of a field $\mathbb{F}$ is the unique prime number p such that $1 + 1 + \ldots p$ times $= 0$ in $\mathbb{F}$. We denote it by $\mathrm{char}(\mathbb{F})$.

Cost of an operation in $\mathbb{F}_q$ - An operation (addition and multiplication) in $\mathbf{F}_q$ is like doing the same operation (addition and multiplication) in the ring $\mathbb{F}_p[x]/(h(x))$. Hence, an operation in $\mathbb{F}_q$ takes $O(\log^2 q)$ bit operations (why?).

Note that the input size in Problem 4 is about $(n+1)\cdot \log q$ bits - since an element of $\mathbf{F}_q$ can be expressed using $\lceil \log q \rceil + 1$ bits. Naturally, we would like to design an algorithm that uses polynomial in n and $\log q$ operations in $\mathbb{F}_q$. Before we proceed, there's a piece of suggestion for you.

A piece of suggestion - Become familiar with some of the basics of finite fields and polynomials over finite fields by reading the first two chapters of the book [LN94] (till page-63). This is because, we will be using several properties of finite fields while describing the polynomial factoring algorithm.

The factoring algorithm is divided into three phases:

- Square-free factoring,
- Distinct degree factoring, and
- Equal degree factoring.

We will discuss each of these phases seperately.

2.1 Square-free factoring

A polynomial f is said to be *square-free* if there is no polynomial f_1 and $e \in \mathbb{N}_{>1}$ such that f_1^e divides f. In other words, f factorizes as $f = \prod_{j=1}^{k} f_j$, where every f_j is irreducible over the underlying field (which is $\mathbb{F}_q$ in our case), and $f_i \nmid f_j$ for every $i \neq j$. We show that if $f \in \mathbb{F}_q[x]$ is *not* square-free then a nontrivial factor of f can be found efficiently - this is called the square-free factoring phase.

Recall the definition of the formal derivate f' of a polynomial f from the previous lecture. Let $f = \sum_{i=0}^{n} c_i x^i$, where $c_i \in \mathbb{F}_q$. Then, $f' = \sum_{i=1}^{n} i c_i x^{i-1}$, which means that $f' = 0$ if and only if for every coefficient $c_i \neq 0$, p divides i. If $f' = 0$, then f is a polynomial of the form $f = \sum_j c_j x^{pj} = (\sum_j c_j^{p^{m-1}} x^j)^p$, as $c_j^q = c_j$ (c_j being an element of $\mathbb{F}_q$) [2]. Thus, $g = \sum_j c_j^{p^{m-1}} x^j$ is a factor of f. To compute g, we just need to compute $c_j^{p^{m-1}}$ for every nonzero coefficients c_j of f. This takes a total of $O(nm \log p) = O(n \log q)$ operations over $\mathbb{F}_q$ by *repeated squaring* (see exercise 1 in lecture 1).

Suppose, $f' \neq 0$. Let $f = f_1^e \cdot h$, where $e > 1$. Then, $f' = f_1^{e-1}(eh + f_1 h')$, which means that $\gcd(f, f')$ yields a nontrivial factor of f, as f_1 divides both f and f'. Computing $\gcd(f, f')$ takes $O(\mathsf{M}(n) \log n)$ operations over $\mathbb{F}_q$. Therefore, if f is not square-free then we can find a nontrivial factor of f using polynomial in n and $\log q$ many $\mathbb{F}_q$-operations. (Note: Computing f' from f takes $O(n)$ operations over $\mathbb{F}_q$.)

2.2 Distinct degree factoring

At the end of the square-free factoring phase, we can assume that the polynomial at hand, say f, factorizes as $f = \prod_{j=1}^{k} f_j$, where f_j is irreducible over $\mathbb{F}_q$ and $f_i \nmid f_j$, for $i \neq j$. Suppose that f has two factors f_1 and f_2 such that $\deg(f_1) \neq \deg(f_2)$. Let $\deg(f_1) = d_1 < \deg(f_2) = d_2$. If this is the case, then we will show that a non-trivial factor of f can be retrieved, efficiently - this is called distinct degree factoring. At the heart of the argument lies the following lemma.

Lemma 5 *For any $d \geq 1$, $x^{q^d} - x$ is the product of all monic irreducible polynomials in $\mathbb{F}_q[x]$ whose degree divides d.*

[2] Here, we are using the property: for any $f_1, f_2 \in \mathbb{F}_q[x]$, $(f_1 + f_2)^p = f_1^p + f_2^p$, where $p = \mathrm{char}(\mathbb{F}_q)$.

We leave the proof of this lemma as an excercise. This means that $\gcd(x^{q^{d_1}} - x, f)$ yields a nontrivial factor of f, as f_1 divides $x^{q^{d_1}} - x$ whereas f_2 doesn't (by lemma 5). Apparently, it seems that computing the $\gcd(x^{q^{d_1}} - x, f)$ is infeasible as degree of $x^{q^{d_1}} - x$ is q^{d_1}, which can be much larger than n. However, this is not the case because of the following observation.

Observation 6 *Let* $r = (x^{q^{d_1}} - x) \mod f$. *Then* $\gcd(x^{q^{d_1}} - x, f) = \gcd(r, f)$.

Degree of r is less than n, so computing $\gcd(r, f)$ takes $O(\mathsf{M}(n) \log n)$ operations over $\mathbb{F}_q$. But, how do we compute r? Use repeated squaring - first, find $r' = x^{q^{d_1}} \mod f$ using repeated squaring in the ring $\mathcal{R} = \mathbb{F}_q[x]/(f)$, and then compute $r = r' - x$. Computing r' takes $O(n \log q)$ operations in the ring $\mathcal{R}$ (why?). An operation in $\mathcal{R}$ takes $O(\mathsf{M}(n))$ operations over $\mathbb{F}_q$ (why?). Therefore, we can find r and compute $\gcd(r, f)$ to obtain a nontrivial factor of f using polynomial in n and $\log q$ operations over $\mathbb{F}_q$.

At the end of the distinct degree factoring step, we can therefore assume that the polynomial f at hand is of the form: $f = \prod_{j=1}^{k} f_j$, where all f_j's have the same degree. We will see how to handle this case in the next class.

Exercises:

1. A polynomial $f = \sum_{i=0}^{n} f_i x^i \in \mathbb{Z}[x]$ is called a *primitive* polynomial if $\gcd(f_0, \ldots, f_n) = 1$. Show that if f and g are primitive polynomials then fg is also a primitive polynomial. Using this, prove Gauss' lemma: If a primitive polynomial $f \in \mathbb{Z}[x]$ factorizes into two polynomials having rational coefficients, then f can also be factored into two polynomials having integer coefficients. Infer that if $f, g \in \mathbb{Z}[x]$ are monic, then $h = \gcd(f, g)$ is also a monic polynomial in $\mathbb{Z}[x]$.

2. Prove that, if $A_f = \max_i\{|f_i|\}$ and $A_g = \max_j\{g_j\}$ then $|\mathsf{Res}(f, g)| \leq (n+1)^{m/2}(m+1)^{n/2} A_f^m A_g^n$.

3. In the time complexity analysis of Algorithm 1, find a precise expression for the polynomial time bit complexity in 'big-Oh' notation.

4. Prove lemma 5. (Hint: Use induction and the fact that $h(x^q) = (h(x))^q$ over $\mathbb{F}_q$ for any polynomial $h \in \mathbb{F}_q[x]$.)

5. While explaining the square-free factoring and the distinct-degree factoring phases, we have hand-waved to some extent and concluded that these two phases take polynomial in n and $\log q$ operations over $\mathbb{F}_q$. Your job is to find a precise complexity expression (in terms of $\mathbb{F}_q$ operations) for these two phases after which we can assume that f has equal-degree irreducible factors.

References

[GG03] Joachim Von Zur Gathen and Jurgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, New York, NY, USA, 2003.

[LN94] Rudolf Lidl and Harald Neiderreiter. *Introduction to finite fields and their applications*. Cambridge University Press, 1994.