Computational Number Theory and Algebra May 7, 2012

Lecture 6

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

In the last class, we saw that polynomial division has the same asymptotic complexity as polynomial multiplication. In today's lecture, we will see two more problems that have (nearly) the same complexity as polynomial multiplication, namely - multipoint evaluation and interpolation of a polynomial. The topics of discussion for today's class are:

- Multipoint evaluation of a polynomial,
- Polynomial interpolation,
- Resultant of two polynomials.

1 Multipoint evaluation of a polynomial

Let $f(x) \in \mathcal{R}[x]$ be a given polynomial of degree less than $n = 2^k$, where $\mathcal{R}$ is a commutative ring with unity. The multipoint evaluation problem is the task of evaluating f at n distinct points $u_0, \ldots, u_{n-1}$ of $\mathcal{R}$. (We have seen an application of multipoint evaluations in the Reed-Solomon encoding procedure where $\mathcal{R}$ is a finite field.) It is easy to design an algorithm that uses $O(n^2)$ addition and mutiplication operations in $\mathcal{R}$ - but, we can do better. The idea is to use recursion.

Let $P_0 = \prod_{\ell=0}^{n/2-1} (x - u_\ell)$ and $P_1 = \prod_{\ell=0}^{n/2-1} (x - u_{n/2+\ell})$. Define, $r_0 = f \mod P_0$ and $r_1 = f \mod P_1$. Now, notice that $r_0(u_i) = f(u_i)$ for all $0 \leq i \leq n/2 - 1$, and similarly $r_1(u_i) = f(u_i)$ for all $n/2 \leq i \leq n - 1$. This immediately suggests a recursive algorithm: Compute P_0, P_1, r_0, r_1 and reduce the problem to multipoint evaluation problems on r_0 and r_1 that have degree bounded by $n/2 = 2^{k-1}$.

We are almost done, except that we would need the polynomials $P_{i,j} \stackrel{\text{def}}{=} \prod_{\ell=0}^{2^i-1} (x - u_{j \cdot 2^i + \ell})$ for $0 \leq i \leq k$ and $0 \leq j < 2^{k-i}$, at deeper levels of the recursion. So, we (pre)compute all these $P_{i,j}$'s using the relations:

$$P_{0,j} = (x - u_j) \text{ and } P_{i+1,j} = P_{i,2j} \cdot P_{i,2j+1}. \tag{1}$$

Since $\deg(P_{i,j}) = 2^i$, we can compute $P_{i+1,j}$ from $P_{i,2j}$ and $P_{i,2j+1}$ using $O(\mathsf{M}(2^i))$ operations. Summing over all $0 \leq j < 2^{k-i-1}$ for a fixed $i+1$, we spend $2^{k-i-1} \cdot O(\mathsf{M}(2^i)) = O(\mathsf{M}(n))$ operations (by Observation 1). Now summing over all $0 \leq i \leq k-1$, we can bound the complexity of this (pre)computation step by $O(\mathsf{M}(n) \log n)$ operations over $\mathcal{R}$.

Algorithm 1 Multipoint evaluation

(Pre)computation step: Compute $P_{i,j}$ for all $0 \leq i \leq k$ and $0 \leq j < 2^{k-i}$.
1. If $n = 1$ return f.
2. Let $r_0 = f \mod P_{k-1,0}$ and $r_1 = f \mod P_{k-1,1}$.
3. Recursively, evaluate r_0 at $u_0, \ldots, u_{n/2-1}$.
4. Recursively, evaluate r_1 at $u_{n/2}, \ldots, u_{n-1}$.
5. Output $r_0(u_0), \ldots, r_0(u_{n/2-1}), r_1(u_{n/2}), \ldots, r_1(u_{n-1})$.

Time complexity - As discussed before, the (pre)computation step takes $O(\mathsf{M}(n) \log n)$ operations in $\mathcal{R}$. In step 2, r_0 and r_1 can be computed by using the divison algorithm (discussed in the previous lecture), which takes $O(\mathsf{M}(n))$ operations. Hence, the time taken by the recursive steps of the algorithm is,

$$\begin{aligned} T(n) &= 2 \cdot T(n/2) + O(\mathsf{M}(n)) \\ &= O(\mathsf{M}(n) \log n). \end{aligned}$$

Therefore, the total time taken by the algorithm (including the (pre)computation step) is $O(\mathsf{M}(n) \log n)$.

2 Polynomial interpolation

Polynomial interpolation is, in a way, the converse task of multipoint evaluation: Given a set of $n = 2^k$ tuples $(u_0, v_0), \ldots, (u_{n-1}, v_{n-1})$, where u_i, v_i belong to a field $\mathbb{F}$ and u_i's are distinct, find the unique polynomial $f \in \mathbb{F}[x]$ of degree less than n such that $f(u_i) = v_i$ for all $0 \leq i < n$. Just like multipoint evaluation, there is a divide and conquer strategy for interpolating f using Lagrange's formula,

$$f(x) = \sum_{i=0}^{n-1} v_i \cdot \prod_{j=0, j\neq i}^{n-1} \frac{x - u_j}{u_i - u_j}$$

(Note that the interpolation problem makes sense even over a ring $\mathcal{R}$ if $(u_i - u_j)$'s are units in $\mathcal{R}$.) We would like to compute $f(x)$ using the above formula. Let us denote the product $\prod_{j=0, j\neq i}^{n-1} \frac{1}{u_i - u_j}$ by s_i. We can rewrite the Lagrange's formula as,

$$f(x) = \sum_{i=0}^{n-1} v_i s_i \cdot \prod_{j=0, j\neq i}^{n-1} (x - u_j)$$

At first, let us see how to compute all s_i, $0 \leq i \leq n-1$, efficiently. Recall the definition of the polynomials $P_{i,j}$'s from the previous section. Notice that, ${s_i}^{-1} = P'_{k,0}(u_i)$ for every $0 \leq i \leq n-1$, where $P'_{k,0}$ is the *formal derivative* of the polynomial $P_{k,0} = \prod_{j=0}^{n-1} (x - u_j)$ (see excercise 1). Which means, we can evaluate $P'_{k,0}$ at points $u_0, \ldots, u_{n-1}$ to obtain $s_0^{-1}, \ldots, s_{n-1}^{-1}$ - which is just a multipoint evaluation problem. To find $P'_{k,0}$, we first compute $P_{k,0}$ using the recursive formula (equation 1), and then compute its formal derivative. Therefore, $s_0, \ldots, s_{n-1}$ can be computed using $O(\mathsf{M}(n) \log n)$ operations over $\mathbb{F}$ (assume that computing inverse in $\mathbb{F}$ is a unit $\mathbb{F}$-operation). Once we compute the s_i's, we can also compute all the $v'_i = v_i s_i$. With this (pre)computation of the $v'_i s$, the interpolation problem reduces to the following problem: Given $(u_0, v'_0), \ldots, (u_{n-1}, v'_{n-1})$, compute the polynomial,

$$f(x) = \sum_{i=0}^{n-1} v'_i \cdot \prod_{j=0, j\neq i}^{n-1} (x - u_j)$$

Now, observe that f has the following structure: $f = r_0(x) P_{k-1,1} + r_1(x) P_{k-1,0}$, where

$$r_0(x) = \sum_{i=0}^{n/2-1} v'_i \cdot \prod_{j=0, j\neq i}^{n/2-1} (x - u_j) \quad \text{and} \quad r_1(x) = \sum_{i=n/2}^{n-1} v'_i \cdot \prod_{j=n/2, j\neq i}^{n-1} (x - u_j)$$

This suggests the following recursive procedure for polynomial interpolation.

Algorithm 2 Polynomial interpolation

(Pre)computation: Compute $P_{i,j}$ for $0 \leq i \leq k$ and $0 \leq j < 2^{k-i}$. Compute v'_i for $0 \leq i < n$.
1. If $n = 1$ return v'_0.
2. Recursively, compute $r_0 = \sum_{i=0}^{n/2-1} v'_i \cdot \prod_{j=0, j\neq i}^{n/2-1} (x - u_j)$.
3. Recursively, compute $r_1 = \sum_{i=n/2}^{n-1} v'_i \cdot \prod_{j=n/2, j\neq i}^{n-1} (x - u_j)$.
4. Output $r_0 P_{k-1,1} + r_1 P_{k-1,0}$.

Time complexity: As discussed before, the (pre)computation step takes $O(\mathsf{M}(n) \log n)$ operations over $\mathbb{F}$. The recursive steps of the algorithm compute r_0 from $(u_0, v'_0), \ldots, (u_{n/2-1}, v'_{n/2-1})$ and r_1 from $(u_{n/2}, v'_{n/2}), \ldots, (u_{n-1}, v'_{n-1})$ respectively. Finally, step 4 does two multiplications of polynomials whose degrees are bounded by $n/2$. Hence, the overall complexity of the recursive steps is

$$T(n) = 2 \cdot T(n/2) + O(\mathsf{M}(n)) = O(\mathsf{M}(n) \log n).$$

Therefore, the total time taken by the algorithm (including the (pre)computation step) is $O(\mathsf{M}(n) \log n)$.

3 The Resultant

Let $\mathcal{R}$ be an integral domain and $\mathbb{F}$ be its field of fractions. (If you are not familiar with integral domain, assume that $\mathcal{R}$ is $\mathbb{Z}$, and $\mathbb{F}$ is $\mathbb{Q}$, the field of rational numbers.) Let f and g be two polynomials in $\mathcal{R}[x]$ of degree n and m, respectively. Let $\gcd(f,g)$ denote the unique, *monic* largest common divisor of f and g over $\mathbb{F}$.

Lemma 1 *The* $\gcd(f,g)$ *is nontrivial (meaning,* $\gcd(f,g) \neq 1$*) if and only if there exists polynomials* $s,t \in \mathbb{F}[x]$*, with* $\deg(s) < m$ *and* $\deg(t) < n$*, such that* $sf + tg = 0$*.*

Proof Suppose $h = \gcd(f,g)$. If $\gcd(f,g) \neq 1$ then $\deg(h) > 1$. Now, if we take $s = \frac{g}{h}$ and $t = -\frac{f}{h}$ then $\deg(s) < m, \deg(t) < n$ and $sf + tg = 0$.

To show the other direction, suppose that there exist s and t with $\deg(s) < m, \deg(t) < n$ and $sf+tg = 0$. If $\gcd(f,g) = 1$ then by "unique factorization" over $\mathbb{F}$, g should divide s. But, this is not possible as $\deg(s) < \deg(g)$. Hence $\gcd(f,g)$ is nontrivial. ■

Let $f = \sum_{i=0}^{n} f_i x^i$ and $g = \sum_{j=0}^{m} g_j x^j$, where $f_i, g_j \in \mathbb{F}$ for $0 \leq i \leq n$ and $0 \leq j \leq m$. Since, $\deg(f) = n$ and $\deg(g) = m$, f_n and g_m are nonzero. Suppose $s = \sum_{k=0}^{m-1} \alpha_k x^k$ and $t = \sum_{\ell=0}^{n-1} \beta_\ell x^\ell$. Treat the coefficients $\alpha_0, \ldots, \alpha_{m-1}$ and $\beta_0, \ldots, \beta_{n-1}$ as variables. Now, consider the relation $sf + tg = 0$. By multiplying s, f and t, g, and then equating the coefficients of x^i to zero for all $0 \leq i \leq n+m-1$, we get a system of $n+m$ homogeneous linear equations in the variables $\alpha_{m-1}, \ldots, \alpha_0, \beta_{n-1}, \ldots, \beta_0$. The coefficient matrix of this linear system is called the Sylvester matrix of f and g, and is denoted by $S(f,g)$ (which means, $S(f,g) \cdot (\alpha_{m-1}, \ldots, \alpha_0, \beta_{n-1}, \ldots, \beta_0)^T = 0$.) Verify that $S(f,g)$ is the following $(n+m) \times (n+m)$ matrix.

$$S(f,g) = \begin{pmatrix}
f_n & & & & g_m & & & & \\
f_{n-1} & f_n & & & g_{m-1} & g_m & & & \\
\vdots & \vdots & \ddots & & \vdots & \vdots & \ddots & & \\
\vdots & \vdots & & f_n & g_1 & \vdots & & \ddots & \\
\vdots & \vdots & & f_{n-1} & g_0 & \vdots & & & \ddots \\
f_1 & \vdots & & \vdots & & g_0 & & & g_m \\
f_0 & \vdots & & \vdots & & & \ddots & & \vdots \\
 & f_0 & & \vdots & & & & \ddots & \vdots \\
 & & \ddots & \vdots & & & & & \ddots \\
 & & & f_0 & & & & & g_0
\end{pmatrix}_{(n+m)\times(n+m)}.$$

The resultant of f and g is defined as, $\mathsf{Res}_x(f,g) = \det(S(f,g))$. By Lemma 1, the above linear system has a nonzero solution if and only if $\gcd(f,g)$ is nontrivial. This has the following implication.

Lemma 2 *The* $\gcd(f,g)$ *is nontrivial if and only if* $\mathsf{Res}_x(f,g) = \det(S(f,g)) = 0$*.*

Another useful fact about the resultant is the following.

Lemma 3 *There exist* $s,t \in \mathcal{R}[x]$*, with* $\deg(s) < m$ *and* $\deg(t) < n$*, such that* $sf + tg = \mathsf{Res}_x(f,g)$*.*

Proof If $\gcd(f,g) \neq 1$, then from Lemma 1 and 2 it follows that, there exist $s', t' \in \mathbb{F}[x]$, with $\deg(s') < m$ and $\deg(t') < n$, such that $s'f + t'g = 0 = \mathsf{Res}_x(f,g)$. Since a coefficient of s' or t' is of the form $\frac{a}{b}$, where $a, b \in \mathcal{R}$ and $b \neq 0$, by clearing out the denominators of the coefficients of s' and t' we get $s,t \in \mathcal{R}[x]$ such that $sf + tg = 0$. Clearly, $\deg(s) = \deg(s') < m$ and $\deg(t) = \deg(t') < n$.

Suppose $\gcd(f,g) = 1$. By the extended Euclidean algorithm (see Appendix), there exist $s', t' \in \mathbb{F}[x]$, with $\deg(s') < m$ and $\deg(t') < n$, such that $s'f + t'g = 1$. Let $s' = \sum_{k=0}^{m-1} \alpha_k x^k$ and $t' = \sum_{\ell=0}^{n-1} \beta_\ell x^\ell$. Once again, by multiplying s', f and t', g, and then equating the coefficients of x^i for $0 \leq i \leq n+m-1$, we get a linear system in $\alpha_{m-1}, \ldots, \alpha_0, \beta_{n-1}, \ldots, \beta_0$ with $S(f,g)$ as the coefficient matrix. By Cramer's rule, every α_k (similarly, β_ℓ) is of the form $a/\mathsf{Res}_x(f,g)$, where $a \in \mathcal{R}$. Hence, the polynomials $s = \mathsf{Res}_x(f,g) \cdot s'$ and $t = \mathsf{Res}_x(f,g) \cdot t'$ both belong to $\mathcal{R}[x]$. Therefore, $sf + tg = \mathsf{Res}_x(f,g)$. ■

Exercises:

1. Let $p(x) = \sum_{i=0}^{n-1} p_i x^i$, where $p_i \in \mathbb{F}$, be a polynomial over a field $\mathbb{F}$. Define the *formal derivative* of $p(x)$ as, $p'(x) = \sum_{i=1}^{n-1} i p_i x^{i-1}$. Show that, for any $p, q \in \mathbb{F}[x]$, $(p+q)' = p' + q'$ and $(pq)' = pq' + p'q$. Infer that, in section 2 indeed ${s_i}^{-1} = P'_{k,0}(u_i)$.

Reading home work: Basics of finite fields

1. Read the first two chapters of the book [LN94] (till pg-63). You can also read Chapter 19 of [Sho09], or Chapter 25 (section 25.4) of [GG03].

Appendix

GCD of polynomials: Extended Euclidean algorithm

The classical Euclidean algorithm computes $r = f \bmod g$ and then recursively computes the gcd of r and g. The extended Euclidean algorithm, given below, computes two polynomials s and t such that $sf + tg = gcd(f,g) = d$, with $\deg(s) < \deg(g)$ and $\deg(t) < \deg(f)$. We can use this algorithm to compute modular inverse. For instance, if f and g are relatively coprime then $s = f^{-1} \bmod g$.

Algorithm 3 Extended Euclidean algorithm

1. Let $(r, r') \leftarrow (f, g)$, $(s, s') \leftarrow (1, 0)$ and $(t, t') \leftarrow (0, 1)$.
2. while $r' \neq 0$ do
   $q \leftarrow \lfloor \frac{r}{r'} \rfloor$, $r'' \leftarrow r \mod r'$
   $(r, s, t, r', s', t') \leftarrow (r', s', t', r'', s - s'q, t - t'q)$
3. Let $d \leftarrow r$.
4. Output d, s, t.

It is not difficult to analyse this algorithm and show that the time complexity is bounded by $O(n^2)$ operations over the underlying field $\mathbb{F}$, where n is the bound on the degree of f and g (refer to chapter 4 of [Sho09]). But then, there is also a faster version of the extended Euclidean algorithm that uses $O(\mathsf{M}(n) \log n)$ operations over $\mathbb{F}$ (refer to chapter 11 of [GG03]). **GCD of integers -** Just like polynomials, gcd of two N-bit integers f, g can be computed using the extended Euclidean algorithm, which outputs two integers s and t such that $sf + tg = \gcd(f,g)$. The time taken by a faster version of the extended Euclidean algorithm over integers is $O(\mathsf{M_I}(N) \log N)$ bit operations (refer to chapter 3 and 11 of [GG03]).

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.

[Sho09] Victor Shoup. *A Computational Introduction to Number Theory and Algebra*. Cambridge University Press, New York, 2009. Available from `http://shoup.net/ntb/`.