CS 487/687 Introduction to Symbolic Computation

Keven Qiu
Instructor: Armin Jamshidpey

Contents

Chapter 1

Basic Algebraic Domains

1.1 Mathematical Domains

Most algorithms for polynomials, matrices, etc. come from

- Integers
- Rational numbers
- Integers modulo n (n is often a prime or a power of a prime)
- Algebraic extensions $(\mathbb{Q}(\sqrt{2}), \mathbb{Q}(\sqrt{2+\sqrt{3}}))$
- Complex numbers

Definition: Ring

A set with an operation $+$ and an operation $\times$ where

- $a + 0 = 0 + a = a$
- $a + (-a) = 0$
- $a + b = b + a$
- $(a + b) + c = a + (b + c)$
- $a(bc) = (ab)c$
- $a(b + c) = ab + ac$

Definition: Commutative Ring

A ring where $ab = ba$.

Definition: Ring with Unit

A ring with a special element 1 such that $a \cdot 1 = 1 \cdot a = a$.

1.2 Integers, Rationals, and Polynomials

Assume that the machine architecture has 64 bits. Therefore, integers are represented exactly in $[0, 2^{64} - 1]$. For larger integers, we can use an array of word-size numbers.

Any integer a can be expressed as

$$a = (-1)^s \sum_{i=0}^{n} a_i B^i$$

where $B = 2^{64}, s \in \{0, 1\}, 0 \leq a_i \leq B - 1$.

If $0 \leq n + 1 < 2^{63}$, then a can be encoded as an array

$$[s \cdot 2^{63} + n + 1, a_0, \ldots, a_n]$$

of 64 bit words.

Polynomials can be represented in dense (arrays) or sparse (linked lists) forms. Multivariate polynomials are typically sparse.

Definition: Field

A ring $\mathbb{F}$ with addition and multiplication such that every nonzero element has a multiplicative inverse.

Some examples of fields include rational numbers $\mathbb{Q}$, $\mathbb{Q}(\sqrt{2}) = \{a + b\sqrt{2} : a, b \in \mathbb{Q}\}$, $\mathbb{Z}_p$, $\mathbb{F}_q$ (finite field of size $q = p^k$), $\mathbb{R}$, and $\mathbb{C}$.

Given a base ring R, we can construct a polynomial ring $R[x]$ by adding a new free variable x to R. Elements will have the form $a_0 + a_1 x + \cdots + a_d x^d$, $a_i \in R$. Equality is defined by their coefficients.

Definition: Greatest Common Divisor

The greatest common divisor of $a, b \in R$, denoted $\gcd(a, b)$ is an element $c \in R$ such that c divides both a and b and if r divides both a and b, then r divides c.

gcd's do not always exist as it depends on the ring, and even if it does exist, it is not clear that an algorithm exists.

Definition: Unit

$u \in R$ is a unit if there is $v \in R$ such that $uv = 1$.

Definition: Associates

$a, b \in R$ are associates if $a = ub$ with $u \in R$ a unit.

3 and -3 are associates in $\mathbb{Z}$, 3 and 9 are associates in $\mathbb{Z}_{12}$.

Definition: Irreducible

A non-unit element $a \in R \setminus \{0\}$ is irreducible if $a = bc$ implies one of b, c is a unit.

Definition: Zero Divisor

An element $a \in R \setminus \{0\}$ such that there is a non-zero $b \in R \setminus \{0\}$ such that $a \cdot b = 0$.

Definition: Integral Domain

A ring R having no zero divisor.

Definition: Euclidean Domain

An integral domain R with a Euclidean function $|\cdot| : R \to \mathbb{N} \cup \{-\infty\}$ such that for all $a, b \in R$ with $b \neq 0$, there exists $q, r \in R$ such that

$$a = qb + r, |r| < |b|$$

E.g. $\mathbb{Z}$ is a Euclidean domain with Euclidean function absolute value, units are ± 1 and irreducibles are prime integers.

E.g. $\mathbb{F}[x]$ is a Euclidean domain with Euclidean function degree, units are constant polynomials, and irreducibles are polynomials that do not factor.

E.g. $\mathbb{Z}[i]$ is a Euclidean domain with Euclidean function $|a + bi| = a^2 + b^2$, units are $\pm 1, \pm i$.

E.g. $\mathbb{R}[x]$ is not a Euclidean domain when R is not a field, units are constants which are units in R.

Measuring cost in rings:

- $\mathbb{Z}$: The bit complexity of the integer is

$$\log a = \begin{cases} 1 & \text{if } a = 0 \\ 1 + \lfloor \log |a| \rfloor & \text{otherwise} \end{cases}$$

- $\mathbb{Q}$: The complexity of a/b is the total bit complexity of a and b.
- $\mathbb{F}_q$: The complexity is bit complexity $\log q$.
-

1.3 Basic Algebraic Operations with Cost

Addition over $\mathbb{Z}[x]$

Input: two elements $a, b \in \mathbb{Z}[x]$, $\deg(a) = m$, $\deg(b) = n$.
Output: $c = a + b$.
$c_i = a_i + b_i$ for $0 \le i \le \max(m, n)$ and the running time is $O(m + n)$.

Multiplication over $\mathbb{Z}[x]$

Input: two elements $a, b \in \mathbb{Z}[x]$.
Output: $a \cdot b$.
$c_k = \sum_{i=0}^{k} a_i b_{k-i}$. Compute all $(m + 1)(n + 1)$ multiplications of $a_i b_j$ and add the mn summands so running time is $O(mn)$.

Addition and Multiplication Over $R = \mathbb{Z}$

Input: two elements $a, b \in \mathbb{Z}$.
Output: $a + b$ and $a \cdot b$.
Use bit representation of a, b. For addition, the running time is $O(\log a + \log b)$. For multiplication, there are $\lceil \log b \rceil$ additions of multiples of a, so running time is $O(\log a \cdot \log b)$.

So over $\mathbb{Z}$ we count bit operations and over $\mathbb{Z}[x]$ we count operations in $\mathbb{Z}$.

Division with Remainder over $\mathbb{Z}[x]$

Input: two elements $a, b \in \mathbb{Z}[x]$, with b nonzero and leading coefficient of b ($LC(b)$) is unit in $\mathbb{Z}$.
Output: $q, r \in \mathbb{Z}[x]$ such that $\deg(r) < \deg(b)$ and $a = qb + r$.
Start with $r = a, q = 0$. While $\deg(r) \ge \deg(b)$, do $q = q + \frac{LC(r)}{LC(b)} x^{\deg(r) - \deg(b)}$ and $r = r - \frac{LC(r)}{LC(b)} x^{\deg(r) - \deg(b)} \cdot b$. We perform at most $\deg(a) - \deg(b) + 1$ subtractions to r so total time is $(\deg(a) - \deg(b) + 1)(\deg(b) + 1)$.

Division with Remainder over $\mathbb{Z}$

Input: two elements $a, b \in \mathbb{Z}$, with b nonzero.
Output: $q, r \in \mathbb{Z}$ such that $|r| < |b|$ and $a = qb + r$.
Start with $r = a, q = 0$. While $|r| \ge |b|$, do $q = q + 1$ and $r = r - b$. We perform $\lfloor a/b \rfloor$ subtractions to r, total time is $\frac{a \log b}{b}$.
Instead of subtracting a/b times, we can find the biggest multiple of b in r. $q = q + 2^{\log r - \log b}$, $r = r - 2^{\log r - \log b} \cdot b$. The total running time is $\log q \cdot \log b$.

$\gcd(a, b)$

Over ring $\mathbb{Z}$, the upper bound is $\log a \cdot \log b$ and over ring $\mathbb{Z}[x]$, the upper bound is $(\deg(a) + 1)(\deg(b) + 1)$.

Chapter 2

Polynomial and Integer Multiplication

Theorem (Master)

Suppose that $a \geq 1, b > 1$. Consider the recurrence

$$T(n) = aT\left(\frac{n}{b}\right) + \Theta(n^y)$$

Denote $x = \log_b a$, then

$$T(n) \in \begin{cases} \Theta(n^x) & \text{if } y < x \\ \Theta(n^y \log n) & \text{if } y = x \\ \Theta(n^y) & \text{if } y > x \end{cases}$$

Polynomial Multiplication

Input: Two polynomials $F = f_0 + f_1 x + \cdots + f_{n-1}x^{n-1}, G = g_0 + g_1 x + \cdots + g_{n-1}x^{n-1}$.
Output: Product $H = FG = h_0 + \ldots h_{2n-2}x^{2n-2}$ with $h_0 = f_0 g_0, \ldots, h_i = \sum_{j+k=i} f_j g_k, \ldots, h_{2n-2} = f_{n-1}g_{n-1}$.

Multiplication is a central problem. There are algorithms for gcd, factorization, root-finding, evaluation, interpolation, Chinese remaindering, linear algebra, polynomial system solving that rely on polynomial multiplication and their complexity can be expressed using multiplication.

Proposition

On can multiply polynomials with n terms using

- Naive algorithm with $O(n^2)$ operations.
- Karatsuba's algorithm with $O(n^{\log_2 3}) = O(n^{1.59})$ operations.
- Toom's algorithm with $O(n^{\log_3 5}) = O(n^{1.47})$ operations.
- Fast Fourier Transform with $O(n \log n)$ operations for nice cases and $O(n \log n \log \log n)$ operations in general.

Polynomials:

$$\begin{aligned}
&(3x^2 + 2x + 1)(6x^2 + 5x + 4) \\
&= (3 \cdot 6)x^4 + (3 \cdot 5 + 2 \cdot 6)x^3 + (3 \cdot 4 + 2 \cdot 5 + 1 \cdot 6)x^2 + (2 \cdot 4 + 1 \cdot 5)x + (1 \cdot 4) \\
&= 18x^4 + 27x^3 + 28x^2 + 13x + 4
\end{aligned}$$

Integers:

$$\begin{aligned}
321 \times 654 &= (3 \cdot 10^2 + 2 \cdot 10 + 1) \times (6 \cdot 10^2 + 5 \cdot 10 + 4) \\
&= 18 \cdot 10^4 + 27 \cdot 10^3 + 28 \cdot 10^2 + 13 \cdot 10 + 4 \\
&= 2 \cdot 10^5 + 9 \cdot 10^3 + 9 \cdot 10^2 + 3 \cdot 10 + 4 \\
&= 209934
\end{aligned}$$

There are similarities, but the carrying for the integer case is seemingly harder.

2.1 Karatsuba's Algorithm

A divide-and-conquer algorithm. Let $F = f_0 + f_1 x, G = g_0 + g_1 x$. Instead of computing 4 multiplications, we compute 3: $f_0 g_0, f_1 g_1, f_0 g_1 + f_1 g_0 = (f_0 + f_1)(g_0 + g_1) - f_0 g_0 - f_1 g_1$.

Suppose now that F, G have n terms with $n = 2^s$ and let

$$F = F_0 + F_1 x^{n/2}, G = G_0 + G_1 x^{n/2}$$

so F_0, F_1, G_0, G_1 have $n/2$ terms. Then

$$H = FG = F_0 G_0 + (F_0 G_1 + F_1 G_0) x^{n/2} + F_1 G_1 x^n$$

Algorithm 1 Karatsuba's Algorithm

if $n = 1$ **then**
  return $h = f_0 g_0$
Compute recursively $F_0G_0, F_1G_1, (F_0 + F_1)(G_0 + G_1)$.
Deduce $F_0G_1 + F_1G_0 = (F_0 + F_1)(G_0 + G_1) - F_0G_0 - F_1G_1$.
return H

2.2 Evaluation

Definition: Polynomial Evaluation

Assume R is a ring. Given $n \in \mathbb{N}$, find an algorithm that, on input $\alpha, a_0, \ldots, a_n \in R$, computes $f(\alpha) \in R$, where

$$f(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0 \in R[x]$$

$$P(\infty) = \lim_{x\to\infty} \frac{P(x)}{x^{\deg(P)}}$$

Definition: Horner's Evaluation

Rewrite the polynomial as

$$f(\alpha) = (\cdots((a_n\alpha + a_{n-1})\alpha + a_{n-2})\alpha + \cdots)\alpha + a_0$$

The cost the algorithm using Horner's rule is n multiplications and n additions as opposed to $2n - 1$ multiplications and n additions for the naive method.

Theorem (Uniqueness of an Interpolating Polynomial)

For any set $\{(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})\}$ pairs such that all x_i's are distinct, there is a unique polynomial $P(x)$ of degree $n-1$ such that $y_i = P(x_i)$ for $0 \le i \le n-1$.

Under assumptions of the previous theorem, we can find $P(x)$ in quadratic time, using Lagrange interpolation:

$$L_i = \frac{\prod_{j\neq i}(x - x_j)}{\prod_{j\neq i}(x_i - x_j)}, P(x) = \sum_{i=0}^{n-1} y_i L_i$$

Application of Evaluation/Interpolation: We want to share a secret between n parties such that 1. together they can discover the secret, 2. no proper subset of the parties can discover the secret.

Construct the scheme:

1. Assume the secret is $s \in \mathbb{F}_p$ where p is a large prime.

2. Choose $f_1, \ldots, f_{n-1}$ and $\alpha_0, \ldots, \alpha_{n-1} \in \mathbb{F}_p$.
3. Set $f(x) = s + f_1 x + \cdots + f_{n-1} x^{n-1} \in \mathbb{F}_p[x]$.
4. Given $(\alpha_i, f(\alpha_i))$ to player i.
5. Together they can construct the unique polynomial f and s.

2.3 Toom's Algorithm

The idea behind Karatsuba's trick:
Evaluation

$$\begin{aligned} f_0 &= F(0) & g_0 &= G(0) \\ f_0 + f_1 &= F(1) & g_0 + g_1 &= G(1) \\ f_1 &= F(\infty) & g_1 &= G(\infty) \end{aligned}$$

Multiplication

$$\begin{aligned} H(0) &= F(0)G(0) \\ H(1) &= F(1)G(1) \\ H(\infty) &= F(\infty)G(\infty) \end{aligned}$$

Interpolation

$$H = H(0) + (H(1) - H(0) - H(\infty))x + H(\infty)x^2$$

Now we work with polynomials in $\mathbb{Q}[x]$. Let $F = f_0 + f_1 x + f_2 x^2$ and $G = g_0 + g_1 x + g_2 x^2$ and

$$H = FG = h_0 + h_1 x + h_2 x^2 + h_3 x^3 + h_4 x^4$$

To get H we still need evaluation, multiplication, and interpolation. Now we need 5 values because H has 5 unknown coefficients.

Evaluation

$$\begin{aligned} F(0) &= f_0 & G(0) &= g_0 \\ F(1) &= f_0 + f_1 + f_2 & G(1) &= g_0 + g_1 + g_2 \\ F(-1) &= f_0 - f_1 + f_2 & G(-1) &= g_0 - g_1 + g_2 \\ F(2) &= f_0 + 2f_1 + 4f_2 & G(2) &= g_0 + 2g_1 + 4g_2 \\ F(\infty) &= f_2 & G(\infty) &= g_2 \end{aligned}$$

Multiplication:

$$H(0) = F(0)G(0), \ldots, H(\infty) = F(\infty)G(\infty)$$

Interpolation:

$$\begin{aligned} H(0) &= h_0 \\ H(-1) &= h_0 - h_1 + h_2 - h_3 + h_4 \\ H(1) &= h_0 + h_1 + h_2 + h_3 + h_4 \\ H(2) &= h_0 + 2h_1 + 4h_2 + 8h_3 + 16h_4 \\ H(\infty) &= h_4 \end{aligned}$$

Linear system of 5 equations in 5 unknowns.

Analysis: At each step we divide n by 3, do 5 recursive calls, and the extra operations count is ℓn for some ℓ. The recurrence is

$$T(n) = 5T\left(\frac{n}{3}\right) + \ell n$$

Master theorem:

$$T(n) = \Theta(n^{\log_3 5})$$

The constant is $\approx \ell$.

Algorithm 2 Generalized Toom's Algorithm

Write input F, G as
$F = F_0 + F_1 x^{n/k} + \cdots + F_{k-1} x^{(k-1)n/k}$
$G = G_0 + G_1 x^{n/k} + \cdots + G_{k-1} x^{(k-1)n/k}$
return $H = FG = H_0 + H_1 x^{n/k} + \cdots + H_{2k-2} x^{(2k-2)n/k}$

Analysis: At each step, we divide n by k, do $2k-1$ recursive calls, and the extra operations count is ℓn. Master theorem gives $T(n) = \Theta(n^{\log_k(2k-1)})$.

2.4 Fast Fourier Transform

Evaluation and interpolation are expensive in general. FFT gives an $O(n \log n)$ evaluation and interpolation, and so an $O(n \log n)$ multiplication.

Definition: nth Root of Unity

A complex number z such that $z^n = 1$.

Definition: Primitive nth Root of Unity

A complex number z such that z is an nth root of unity and $z^k \neq 1$ for $0 < k < n$.

$z_n = e^{2i\pi/n}$ is a primitive nth root of unity.

Proposition

The nth roots of unity are the powers

$$z_n^0 = 1, z_n, z_n^2, \ldots, z_n^{n-1}$$

Proposition

If $m = n/2$, then $z_m = z_n^2$.

Proposition

$\gcd(n, k) = 1 \implies z_n^k$ is a primitive nth root of unity.

Consider the nth roots of unity $z_n^0, \ldots, z_n^{n-1}$, then the DFT by

$$F = f_0 + \cdots + f_{n-1}x^{n-1} \mapsto (F(z_n^0), \ldots, F(z_n^{n-1}))$$

is the Discrete Fourier Transform of order n.

Definition: Discrete Fourier Transform

$$f_\ell = \sum_{k=0}^{n-1} F_k z_n^{\ell k}$$

Definition: Inverse Discrete Fourier Transform

$$F_\ell = \frac{1}{n} \sum_{k=0}^{n-1} f_k z_n^{-\ell k}$$

Fast Fourier Transform can solve this in $O(n \log n)$. This is a divide-and-conquer algorithm.

With $m = n/2$, squaring sends all nth roots of unity to mth roots, i.e. z_n^i and $z_n^{i+m} = -z_n^i$ have the same square.

Any polynomial $F = f_0 + f_1 x + \cdots + f_{n-1}x^{n-1}$ can be written as $F = F_{even}(x^2) + xF_{odd}(x^2)$ with $\deg(F_{even}) < n/2$ and $\deg(F_{odd}) < n/2$.

E.g. $F = 28 + 11x + 34x^2 - 55x^3$. $F_{even}(x^2) = 28 + 34x^2, F_{odd}(x^2) = 11 - 55x^2$, so $F_{even} = 28 + 34x$ and $F_{odd} = 11 - 55x$. We only need to evaluate at $z_n^0, \ldots, z_n^{n/2-1}$.

Decomposition and Evaluation: Given $u_0, \ldots, u_{n-1} \in \mathbb{C}$ to evaluate $F(u_0), \ldots, F(u_{n-1})$, evaluate $v_i = F_{even}(u_i^2), v_i' = F_{odd}(u_i^2)$ and deduce $F(u_i) = v_i + u_i v_i'$. If we choose $u_0, \ldots, u_{n-1}$ poorly, we have to evaluate two polynomials of degree $< n/2$ at n points. For FFT, we choose the u_i as the roots of unity.

The cost $F(n)$ of the FFT algorithm satisfies

- $F(1) = 0$

Algorithm 3 Fast Fourier Transform $FFT(F, n)$

if $n = 1$ **then**
  return f_0
$V = FFT(F_{even}, n/2)$, $V = [v_0, \dots, v_{n/2-1}]$
$V' = FFT(F_{odd}, n/2)$, $V = [v'_0, \dots, v'_{n/2-1}]$
return $[V[i \mod n/2] + z_n^i V'[i \mod n/2] : 0 \leq i < n]$

- $F(n) = 2F(n/2) + cn$

so $F(n) = \Theta(n \log n)$.

Inverse Fourier Transform: Given n, take z_n to be a primitive nth root of unity and let

$$V(z_n) = V(1, z_n, \dots, z_n^{n-1})$$

where V is the Vandermonde matrix. Recall

$$\begin{bmatrix} F(1) \\ \vdots \\ F(z_n^{n-1}) \end{bmatrix} = V(z_n) \cdot \begin{bmatrix} f_0 \\ \vdots \\ f_{n-1} \end{bmatrix}$$

Lemma

$$V(z_n) \cdot V(z_n^{-1}) = n \cdot I_n$$

Proof. $c = V(z_n) \cdot V(z_n^{-1})$. $c_{ij} = [1, z_n^{i-1}, \dots, z_n^{(i-1)(n-1)}][1, z_n^{-(j-1)}, \dots, z^{-(j-1)(n-1)}]^T$.

If $i = j$, then $c_{ij} = n$. If $i \neq j$, $c_{ij} = \sum_{k=0}^{n-1} z_n^{(i-j)k} = \frac{(z_n^{i-j})^k - 1}{z_n^{i-j} - 1} = 0$.

Proposition

Performing the inverse DFT in size n is done by performing a DFT at $z_n^0, z_n^{-1}, \dots, z_n^{-(n-1)}$ and dividing the results by n.

This new DFT is tihe same as before: $z_n^{-}i = z_n^{n-i}$ so the outputs are shuffled. Inverse DFT is $\Theta(n \log n)$.

FFT Multiplication

To multiply two polynomials $F, G \in \mathbb{C}[x]$ of degrees $< m$:

1. Find $n = 2^k$ such that $H = FG$ has degree $< m$. $(n \leq cm)$
2. Compute $DFT(F, n)$ and $DFT(G, n)$. $(O(n \log n))$
3. Multiply the values to get $DFT(H, n)$. $(O(n))$
4. Recover H by inverse DFT. $(O(n \log n))$

Cost is $O(n \log n) = O(m \log m)$.

2.5 Multivariate Polynomials

Degree is not the proper measure anymore and the shape of the set of monomials becomes more important.

One useful trick, Kronecker substitution, works for any multivariate polynomials, good for polynomials $F(x_1, \ldots, x_n)$ with $\deg(F, x_1) < d_1, \ldots, \deg(F, x_n) < d_n$ and reduces to univariate polynomial multiplication.

Kronecker's substitution on example:

$$F = (1 + 3x_1 + 4x_1^2) + (22 + x_1 - x_1^2)x_2 + (-3 - 3x_1 + 2x_1^2)x_2^2 = F_0(x_1) + F_1(x_1)x_2 + F_2(x_1)x_2^2$$

$$G = (-2 + x_1 + x_1^2) + (4 + x_1 + 3x_1^2)x_2 + (3 - x_1 + x_1^2)x_2^2 = G_0(x_1) + G_1(x_1)x_2 + G_2(x_1)x_2^2$$

Then

$$H = F_0G_0 + (F_0G_1 + F_1G_0)x_2 + (F_0G_2 + F_1G_1 + F_2G_0)x_2^2 + (F_1G_2 + F_2G_1)x_2^3 + F_2G_2x_2^4$$

Since all $F_i(x_1)G_j(x_1)$ have degree at most 4, we can replace x_2 by x_1^5, then we have

$$H = F_0G_0 + (F_0G_1 + F_1G_0)x_1^5 + (F_0G_2 + F_1G_1 + F_2G_0)x_1^{10} + (F_1G_2 + F_2G_1)x_1^{15} + F_2G_2x_1^{20}$$

Chapter 3

Extended Euclidean Algorithm

3.1 Greatest Common Divisor

Let $a, b \in R$ where R is a Euclidean domain. A greatest common divisor of a and B is a polynomial g such that g divides a, g divides b, and if c divides both a and b, then c divides g.

If c and d are GCD's of a and b, then $c = \ell d$ for some unit $\ell \neq 0$. The GCD is the one that is normalized (polynomials with leading coefficient 1).

Proposition

- $\gcd(a, b) = \gcd(b, a)$
- $\gcd(a, 0) = \text{normalized}(a)$
- $\gcd(a, c) = 1$ if c is a nonzero unit.

Let $a, b \in R$ with R a Euclidean domain. If $a = bq + r$, then we write $r = a \mod b$ and $q = a \operatorname{div} b$.

Proposition

For all $a, b \in R$,

$$\gcd(a, b) = \gcd(a, b \mod a) = \gcd(b, a \mod b)$$

Proof. Let $r = b \mod a$. Then $r = b - aq$. Let $g = \gcd(a, b)$ and $h = \gcd(a, r)$. g divides a and b, so g divides r. This implies g divides h by property of the GCD for h. h divides a and r, so h divides b. Thus, h divides g.

3.2 Euclid's Algorithm

Algorithm 4 gcd(a, b) Euclid's Algorithm

```
if deg(a) < deg(b) then
    return gcd(b, a)
else
    if b = 0 then
        return normalized(a)
    else
        return gcd(b, a mod b)
```

Towards the iterative algorithm: Let $R = \mathbb{F}[x], |a| = \deg(a)$. We rewrite $a_0 = a, a_1 = b$ and assume $\deg(a_0) \geq \deg(a_1)$, otherwise swap.

- $\gcd(a_0, a_1) = \gcd(a_1, a_2)$ where $a_1 = a_0 \mod a_1$.
- $\gcd(a_1, a_2) = \gcd(a_2, a_3)$ where $a_3 = a_1 \mod a_2$.
- $\gcd(a_i, a_{i+1}) = \gcd(a_{i+1}, a_{i+2})$ where $a_{i+2} = a_i \mod a_{i+1}$.
- $\gcd(a_N, 0) = a_N/\text{leading coefficient}(a_N)$.

Algorithm 5 gcd(a_0, a_1) Iterative Euclid's Algorithm

```
i = 1
while a_i ≠ 0 do
    a_{i+1} = a_{i-1} mod a_i
    i++
return a_{i-1}/leading coefficient(a_{i-1})
```

E.g. Over $\mathbb{Z}_3[x]$, let $a_0 = 1 + 2x + x^2 + x^3 + 2x^4, a_1 = 1 + 2x + x^2 + x^3$.

$$\begin{aligned} a_0 &= 1 + 2x + x^2 + x^3 + 2x^4 \\ a_1 &= 1 + 2x + x^2 + x^3 \\ a_2 &= 2 + 2x + x^2 \\ a_3 &= 2x \\ a_4 &= 2 \\ a_5 &= 0 \end{aligned}$$

Proposition

Given a and b, one can compute $g = \gcd(a, b)$, as well as Bezout coefficients u, v such that

$$au + bv = g$$

where $\deg(u) < \deg(b), \deg(v) < \deg(a)$.

a, b are coprime if $\gcd(a, b) = 1$, so $au + bv = 1$.

E.g. Computing with complex numbers. Complex multiplication is multiplication modulo $1 + x^2$. Complex inversion is extended gcd with $1 + x^2$.

Suppose $z = a + bi$. Compute $G = \gcd(a + bx, 1 + x^2)$ and the Bezout coefficients $U(x), V(x)$. $G = 1, \deg(U) < 2, \deg(V) < 1$, so $U = u_0 + u_1 x$ and $V = v_0$. Then $(u_0 + u_1 x)(a + bx) + v_0(1 + x^2) = 1$. Evaluating at $x = i$ gives $(u_0 + u_1 i)(a + bi) = 1$.

General example: Suppose that $p \in \mathbb{F}[x]$ is irreducible. Then for $a \in \mathbb{F}[x]$, either p divides a and so $\gcd(a, p) = p$ or $\gcd(a, p) = 1$.

Define $\mathbb{F}[x]/p$ by the set of all polynomials of degree less than $\deg(p)$ with addition and multiplication defined modulo p. Now we have inversion modulo p; for $0 \neq a \in \mathbb{F}[x]/p$, $\gcd(a, p) = 1$. So there exists u, v with $au + pv = 1$, so $au = 1$ in $\mathbb{F}[x]/p$.

3.3 Extended Euclidean Algorithm

Getting the quotients, we replace the step $a_{i+1} = a_{i-1} \mod a_i$ by $q_i = a_{i-1} \operatorname{div} a_i$ and $a_{i+1} = a_{i-1} - q_i a_i$. Additionally to a_i, we also compute u_i and v_i with

$$u_0 = 1, u_1 = 0, u_{i+1} = u_{i-1} - q_i u_i$$

$$v_0 = 0, v_1 = 1, v_{i+1} = v_{i-1} - q_i v_i$$

Proposition

For $0 \leq i \leq n$, we have $a_0 u_i + a_1 v_i = a_i$.

Proof. By induction starting with $i = 0$ and 1.

For the final step $i = N$, we have $a_0 u_N + a_1 v_N = a_N$.

E.g. $\gcd(91, 63)$. $28 = 91 \mod 63, 1 = 91 \div 63$

$$\underbrace{\begin{pmatrix} 0 & 1 \\ 1 & -1 \end{pmatrix}}_{Q_1} \begin{pmatrix} 91 \\ 63 \end{pmatrix} = \begin{pmatrix} 63 \\ 28 \end{pmatrix}$$

$7 = 63 \mod 28, 2 = 63 \div 28$

$$\underbrace{\begin{pmatrix} 0 & 1 \\ 1 & -2 \end{pmatrix}}_{Q_2} \begin{pmatrix} 63 \\ 28 \end{pmatrix} = \begin{pmatrix} 28 \\ 7 \end{pmatrix}$$

$0 = 28 \mod 7, 4 = 28 \div 7$

$$\underbrace{\begin{pmatrix} 0 & 1 \\ 1 & -4 \end{pmatrix}}_{Q_3} \begin{pmatrix} 28 \\ 7 \end{pmatrix} = \begin{pmatrix} 7 \\ 0 \end{pmatrix}$$

Algorithm 6 Extended Euclidean Algorithm $\gcd(a, b)$

Let $a_0 = a$ and $a_1 = b$ and $R_0 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$
for $i = 1, 2, \ldots$ **do**
  Compute q_i and q_{i+1} such that $a_{i-1} = q_i a_i + a_{i+1}$ where $|a_{i+1}| < |a_i|$
  $\begin{pmatrix} 0 & 1 \\ 1 & -q_i \end{pmatrix} \begin{pmatrix} a_{i-1} \\ a_i \end{pmatrix} = \begin{pmatrix} a_i \\ a_{i+1} \end{pmatrix}$
  Let $R_i := Q_i R_{i-1}$
  Stop at smallest $i = \ell$ such that $a_{\ell+1} = 0$.

$Q_3Q_2Q_1 = \begin{pmatrix} -2 & 3 \\ 9 & -13 \end{pmatrix}$ and $\begin{pmatrix} -2 & 3 \\ 9 & -13 \end{pmatrix} \begin{pmatrix} 91 \\ 63 \end{pmatrix} = \begin{pmatrix} 7 \\ 0 \end{pmatrix}$ so gcd is 7.

Because $|a_1| > \cdots > |a_\ell| > 0$ and $a_{\ell+1} = 0$,

$$R_\ell \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = Q_\ell Q_{\ell-1} \ldots Q_2 Q_1 \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = \begin{pmatrix} s_\ell & t_\ell \\ s_{\ell+1} & t_{\ell+1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = \begin{pmatrix} a_\ell \\ 0 \end{pmatrix}$$

so $s_\ell a_0 + t_\ell a_1 = a_\ell$.

Claim: a_ℓ is a GCD of a_0 and a_1.
Proof. Need to show that

(i) $a_\ell \div a_0$ and $a_\ell \div a_1$.

(ii) If $d \div a_0$ and $d \div a_1$, then $d \div a_\ell$ for all $d \in R$.

For part (i), observe that each Q_i is invertible over R

$$Q_i^{-1} = \begin{pmatrix} q_i & 1 \\ 1 & 0 \end{pmatrix}, Q_i = \begin{pmatrix} 0 & 1 \\ 1 & -q_i \end{pmatrix}$$

This implies that each R_i is invertible over R:

$$R_i^{-1} = Q_1^{-1} Q_2^{-1} \ldots Q_i^{-1}$$

and in particular

$$\begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = \underbrace{\begin{pmatrix} r_1 & r_2 \\ r_3 & r_4 \end{pmatrix}}_{R_\ell^{-1}} \begin{pmatrix} a_\ell \\ 0 \end{pmatrix}$$

This shows (i) and (ii).

Cost analysis: Consider $R = \mathbb{F}[x]$ and assume $\deg(a_0) \geq \deg(a_1)$. $\ell \leq \deg(a_1)$ since $-\infty = \deg(a_{\ell+1}) < \deg(a_\ell) < \cdots < \deg(a_1)$. Division with remainder of a_{i-1} by a_i costs $c(\deg(a_i) + 1)(\deg(q_i) + 1)$ operations from $\mathbb{F}$ for some constant c.

$$\sum_{i=1}^{\ell} \deg(q_i) = \sum_{i=1}^{\ell} (\deg(a_{i-1}) - \deg(a_i)) = \deg(a_0) - \deg(a_\ell) \leq \deg(a_0)$$

The total cost is at most

$$\begin{aligned}
\sum_{i=1}^{\ell} c(\deg(a_i)+1)(\deg(q_i)+1) &\le c(\deg(a_1)+1)\sum_{i=1}^{\ell}(\deg(q_i)+1) && (\deg(a_i)\le\deg(a_1)) \\
&\le c(\deg(a_1)+1)(\deg(a_0)+\ell) \\
&\le c(\deg(a_1)+1)(\deg(a_0)+\deg(a_1)) \\
&= O(\deg(a_0)\deg(a_1))
\end{aligned}$$

Chapter 4

Division with Remainder Using Newton Iteration

Let $a = \sum_{i=0}^{n} a_i x^i$ and $b = \sum_{i=0}^{m} b_i x^i$, $a_n, b_m \neq 0$, $m \leq n$ and $b_m = 1$. We wish to find $q \in \mathbb{F}[x]$ and $r \in \mathbb{F}[x]$ satisfying $a = qb + r$ with $r = 0$ or $\deg(r) < \deg(b)$.

Definition: Reversal of Polynomial

Let $a = \sum_{i=0}^{n} a_i x^i$, then the reversal of a is

$$\underset{n}{\operatorname{rev}}(a) = y^n a\left(\frac{1}{y}\right) = a_n + a_{n-1}y + a_{n-2}y^2 + \cdots + a_1 y^{n-1} + a_0 y^n$$

Substitute $\frac{1}{y}$ for the variable x in the expression $a(x) = q(x) + b(x) + r(x)$ and multiply both sides by y^n to get

$$y^n a\left(\frac{1}{y}\right) = \left(y^{n-m} q\left(\frac{1}{y}\right)\right)\left(y^m b\left(\frac{1}{y}\right)\right) + y^{n-m+1}\left(y^{m-1} r\left(\frac{1}{y}\right)\right)$$

or equivalently,

$$\underset{n}{\operatorname{rev}}(a) = \underset{n-m}{\operatorname{rev}}(q) \cdot \underset{m}{\operatorname{rev}}(b) + y^{n-m+1} \underset{m-1}{\operatorname{rev}}(r)$$

Suppose we could compute $\operatorname{rev}_m(b)^{-1} \in \mathbb{F}[[y]]$, then we could compute q and r as follows:

$$\underset{n}{\operatorname{rev}}(a) \equiv \underset{n-m}{\operatorname{rev}}(q) \cdot \underset{m}{\operatorname{rev}}(b) \pmod{y^{n-m+1}}$$

and

$$\underset{n}{\operatorname{rev}}(a) \cdot \underset{m}{\operatorname{rev}}(b)^{-1} \equiv \underset{n-m}{\operatorname{rev}}(q) \pmod{y^{n-m+1}}$$

We then have $q = \operatorname{rev}_{n-m}(\operatorname{rev}_{n-m}(q))$ and $r = a - qb$.

Conclusion: If $\operatorname{rev}_m(b)^{-1}$ exists and we can compute it in $O(M(n))$, then we can do division in $O(M(n))$, where $M(d)$ is the cost of polynomial multiplication in degree d.

4.1 Newton Iteration

Newton's iteration is a way to compute approximate solutions to various problems. To compute a solution of $P(z) = 0$, we use the iteration $x_0 = \text{random}$ and $x_{i+1} = x_i - \frac{P(x_i)}{P'(x_i)}$.

E.g. Computing $\sqrt{2}$. Take $P(x) = x^2 - 2$, so $P'(x) = 2x$. Newton's iteration is $x_{i+1} = x_i - \frac{x_i^2 - 2}{2x_i}$.

Definition: Power Series

A formal sum of the form
$$A = \sum_{i \geq 0} a_i x^i$$

Computationally, we handle truncated power series
$$A \mod x^d = \sum_{i<d} a_i x^i$$

Addition of power series is done term-by-term. Multiplication is done using the same formulas as polynomials. The set of all power series with coefficients in $\mathbb{F}$ is denoted by $\mathbb{F}[[x]]$ and it is a ring with above operations.

Algorithmically, you only represent truncations and the algorithms are the same as polynomials, $O(n)$ for addition and $M(n)$ for multiplication.

Many functional relations have solutions that are power series, but not polynomials. For example, the inverse. Let $P(x) = -x + 1$, there is no polynomial $Q(x)$ such that $PQ = 1$, but there is a power series $Q = 1 + x + x^2 + x^3 + \cdots$.

Proposition

For any power series P, with constant coefficients $P \neq 0$, there exists a unique power series Q with $PQ = 1$.

There is an $O(n^2)$ algorithm to compute $Q_n = Q \mod x^n$ so that $PQ_n \mod x^n = 1$, meaning
$$PQ_n = 1 + 0x + 0x^2 + \cdots + 0x^{n-1} + r_n x^n + \cdots$$

4.2 Iteration for the Inverse

Given $g \in \mathbb{F}[x]$ and $k \in \mathbb{N}$, find $h \in \mathbb{F}[x]$ of degree less than k satisfying $hg \equiv 1 \pmod{x^k}$.

We need to find a zero of a function $\Phi : \mathbb{F}[[y]] \to \mathbb{F}[[y]]$, namely
$$\Phi(X) = \frac{1}{X} - g$$

since $\Phi(\tilde{g}) = 0$ where $\tilde{g} \in \mathbb{F}[[y]]$ is such that $\tilde{g} \cdot g = 1$. Clearly

$$\Phi'(X) = -\frac{1}{X^2}$$

and our Newton iteration step is

$$h_{i+1} = h_i - \frac{\frac{1}{h_i} - g}{-1/h_i^2} = 2h_i - gh_i^2$$

Theorem

Let $g, h_0, h_1, \dots \in \mathbb{F}[x]$ with $h_0 = 1$ and

$$h_{i+1} \equiv 2h_i - gh_i^2 \pmod{x^{2^{i+1}}}$$

for all i. Assume also that $g_0 = 1$, then for all i

$$gh_i \equiv 1 \pmod{x^{2^i}}$$

Input: $g = g_0 + g_1 x + \cdots + g_n x^n$ and $k \in \mathbb{N}$
Output: $u \in \mathbb{F}[x]$ satisfying $1 - gu \equiv 0 \mod x^k$
$h_0 = 1, r = \lceil \log_2 k \rceil$
for $i = 0, \dots, r-1$ **do**
  $h_{i+1} = (2h_i - gh_i^2) \text{ rem } x^{2^i}$
return h_r

Theorem

The algorithm InversePolyMod uses $O(M(n))$ field operations to correctly compute the inverse.

Corollary

For polynomials of degree n in $\mathbb{F}[x]$, division with remainder requires $O(M(n))$ field operations.

Chapter 5

Chinese Remainder Theorem

When solving a system of linear equations, we can use integer arithmetic, but the intermediate numbers are big. We can use Cramer's rule to get numbers that are smaller.

Let $x = x_1/d, y = y_1/d, z = z_1/d$ be the solutions from the determinants from Cramer's rule. For a given domain $\mathbb{Z}_p$, we need not calculate these determinants. Rather we find the modular solutions $x \pmod p, y \pmod p, z \pmod p, d \pmod p$ via efficient Gaussian elimination and use $x_1 \equiv x \cdot d \pmod p$ and similarly for the other variables.

E.g. Working over $\mathbb{Z}_7$, we have the system

$$\begin{aligned} x + 2y - 3z &= 1 \\ x - 3z &= -2 \\ 3x - z &= -1 \end{aligned}$$

Gaussian elimination gives $x \equiv 1 \pmod 7, y \equiv -2 \pmod 7, z \equiv -2 \pmod 7, d \equiv -2 \pmod 7$. Doing this for $\mathbb{Z}_{11}, \mathbb{Z}_{13}, \mathbb{Z}_{17}, \mathbb{Z}_{19}$, we get the modular representations for x_1 and d as

$$x_1 = (2, -5, -2, 5, 9), d \equiv (-2, 1, 4, -2, -8)$$

So $x_1 = -44280, d = -7380$ and $x = \frac{-44280}{-7380} = 6$.

When do we stop evaluating over the prime fields? For linear systems $Ax = b$, we have Hadamard's bound

$$|\det(a_{ij})| \leq \prod_i \sqrt{\sum_j a_{ij}^2}$$

For the example

$$A = \begin{bmatrix} 22 & 44 & 74 \\ 15 & 14 & -10 \\ -25 & -28 & 20 \end{bmatrix}$$

The Hadamard is $\approx 6\sqrt{206719254} = 86266.40796$, so the determinant is about 5 digits.

Some modular reductions: $\mathbb{Z}$ reduced to many $\mathbb{Z}_p$, $\mathbb{F}[x]$ reduced to many $\mathbb{F}$ via evaluation.

Let R be the Euclidean domain and $m_1, \ldots, m_s \in R$ be pairwise coprime. Let $m = m_1 \ldots m_s$.

Theorem Chinese Remainder Theorem

$$R/(m) \simeq R/(m_1) \times \cdots \times R/(m_s)$$

For example, when $R = \mathbb{Z}, m = 15, m_1 = 3, m_2 = 5$, so $\mathbb{Z}_{15} \simeq \mathbb{Z}_3 \times \mathbb{Z}_5$ with homomorphisms $a \pmod{15} \to (a \pmod 3, a \pmod 5)$, $(x \pmod 3, y \pmod 5) \to 10x + 6y \pmod{15}$.

E.g. $\mathbb{Z}_{459119} \simeq \mathbb{Z}_{17} \times \mathbb{Z}_{239} \times \mathbb{Z}_{113}$. The first direction is $a \pmod{459119} \to (a \pmod{17}, a \pmod{239}, a \pmod{113})$, so $37312 \pmod{459119} \to (5, 42, 108)$.

The other direction is $(a, b, c) \to 378098a + 357306b + 182835c \pmod{459119}$. An example is

$$378098 \cdot 5 + 357306 \cdot 42 + 182835 \cdot 108 \pmod{459119} \to 37312$$

Proof. (Chinese Remainder Theorem) One homomorphism is easy:

$$a \pmod m \to (a \pmod{m_1}, \ldots, a \pmod{m_s})$$

For the other homomorphism, we need to find elements $L_i \in \mathbb{Z}_m$ such that $L_i \equiv \delta_{ij} \pmod{m_j}$. Then the homomorphism is

$$(u_1 \pmod{m_1}, \ldots, u_s \pmod{m_s}) \to u_1 L_1 + \cdots + u_s L_s \pmod m$$

Since the m_i's are pairwise coprime, $\gcd(m_i, m/m_i) = 1$. So there exists $s_i, t_i \in \mathbb{Z}$ such that

$$s_i m_i + t_i(m/m_i) = 1$$

Then $L_i = t_i \cdot m/m_i$ satisfies

$$\begin{cases} L_i = 0 \pmod{m_j} \text{ when } i \neq j \\ L_i = 1 \pmod{m_i} \end{cases}$$

5.1 Complexity

To compute the first homomorphism, we need to compute $a \pmod{m_i}$ for each m_i, which takes $O(\log m \cdot \log m_i)$.

For the second homomorphism, the input is $(u_1, \ldots, u_s) \in \mathbb{Z}_{m_1} \times \cdots \times \mathbb{Z}_{m_s}$ and the output is $a \in \mathbb{Z}_m$ such that $a = u_i \pmod{m_i}$. It is enough to just compute L_i's like previously. So, we compute m by computing $m_1 m_2$, then $m_1 m_2 m_3$, until we compute m, we have

$$c \cdot \sum_{i=2}^{s} \log(m_1 \ldots m_{i-1}) \cdot \log m_i \leq c \log(m) \cdot \sum_{i=2}^{s} \log m_i \leq c(\log m)^2$$

Computing each m/m_i by division algorithm in $O(\log m \log m_i)$.

Now we have computed $m, m/m_1, \ldots, m/m_s$ in time $O(\log^2 m)$ operations. Now we compute the interpolators L_i's. We know that $L_i = t_i \cdot m/m_i$ where $s_i m_i + t_i m/m_i = 1$. So we need the extended Euclidean algorithm to compute (s_i, t_i). We know the cost is $O(\log(m/m_i) \cdot \log(m_i))$ which gives total running time of $O(\log^2 m)$. So both homomorphisms can be computed with $O(\log^2 m)$ operations.