# 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

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

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 $\pm 1$ 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 $0 \leq i \leq \max(m, n)$ 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 $\deg(r) \geq \deg(b)$, 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 $|r| \geq |b|$, 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 $\alpha, a_0, \dots, a_n \in R$, 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 $\{(x_0, y_0), \dots, (x_{n-1}, y_{n-1})\}$ 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 $f_1, \dots, f_{n-1}$ and $\alpha_0, \dots, \alpha_{n-1} \in \mathbb{F}_p$.

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

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

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

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

Multiplication:

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

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 $\ell n$ for some $\ell$. 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

1: Write input $F, G$ as
2: $F = F_0 + F_1 x^{n/k} + \cdots + F_{k-1} x^{(k-1)n/k}$
3: $G = G_0 + G_1 x^{n/k} + \cdots + G_{k-1} x^{(k-1)n/k}$
4: **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 $\ell n$. 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: $n$th Root of Unity**

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

**Definition: Primitive $n$th Root of Unity**

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

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

**Proposition**

The $n$th 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 $n$th root of unity.

Consider the $n$th 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 $n$th roots of unity to $m$th 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, ..., v_{n/2-1}]
V' = FFT(F_odd, n/2), V = [v'_0, ..., v'_{n/2-1}]
return [V[i mod n/2] + z_n^i V'[i mod n/2] : 0 ≤ 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 $n$th 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 $\gcd(a, b)$

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

$Q_3 Q_2 Q_1 = \begin{pmatrix} -2 & 3 \\ 9 & -13 \end{pmatrix}$ 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 $a_{\ell+1} = 0$,

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

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

Claim: $a_\ell$ 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 $R$:

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

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

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

# 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

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

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,

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

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

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

and

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

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

Conclusion: If $\mathrm{rev}_m(b)^{-1}$ 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.

# Chapter 6

# Modular Composition

## 6.1 Fast Exponentiation

**Fast Exponentiation**

Input: $a \in R, n \in \mathbb{N}$ where $R$ is a ring.
Output: $a^n \in R$

The naive way is to do this using $n$ mulitplications. We can do better

**Algorithm 7** RepeatedSquaring

Compute the binary representation of $n$:

$$n = 2^k + n_{k-1}2^{k-1} + \cdots + n_1 \cdot 2 + n_0$$

with $n_i \in \{0, 1\}$.

**for** $i = 0$ to $k$ **do**

  $a_i \leftarrow a^{2^i}$

Compute $a^n = a^{2k+n_{k-1}2^{k-1}+\cdots+n_1\cdot 2+n_0} = a_k \prod_{i=0}^{k-1} a_i^{n_i}$.

**Modular Inverse**

Input: $0 \neq a \in \mathbb{Z}_p$.
Output: $a^{-1} \in \mathbb{Z}_p$.

**Proposition**

If $p \in \mathbb{N}$ is prime and $a, b \in \mathbb{Z}$, then

$$(a+b)^p = a^p + b^p \pmod{p}$$

and more generally

$$(a+b)^{p^i} = a^{p^i} + b^{p^i} \pmod{p}$$

**Theorem (Fermat's Little Theorem)**

If $p \in \mathbb{N}$ is prime and $a \in \mathbb{Z}$, then $a^p \equiv a \pmod{p}$ and if $\gcd(a, p) = 1$, then $a^{p-1} \equiv 1 \pmod{p}$.

Application (Diffie-Hellman Key Exchange): A trusted party chooses and publishes a large prime $p$ and $g \in \mathbb{F}_p^*$.

- Alice chooses a secret integer $a$ and computes $A \equiv g^a \pmod{p}$.
- Bob chooses a secret integer $b$ and computes $B \equiv g^b \pmod{p}$.
- Alice sends $A$ to Bob.
- Bob sends $B$ to Alice.
- Alice computes $B^a \pmod{p}$ and Bob computes $A^b \pmod{p}$.

Then $B^a \equiv A^b \pmod{p}$.

## 6.2 Shanks' Babystep-Giantstep Algortihm

**Discrete Logarithm Problem (DLP)**

Given $g, h \in \mathbb{F}_p^*$, find $x$ such that $g^x \equiv h \pmod{p}$.

Observation: Let $n = \lceil \sqrt{p-1} \rceil$. We have $g^x = g^{nq+r}$ for some $0 \leq r < n$. Then

$$g^x = h \implies g^r = hg^{-nq}$$

1: Create list $[g^0, g^1, g^2, \ldots, g^n]$ (babysteps).
2: Create list $[h, h \cdot g^{-n}, h \cdot g^{-2n}, \ldots, h \cdot g^{-n^2}]$. (giantsteps)
3: Find a match between the two lists, say $g^i = hg^{-jn}$.
4: **return** $x = jn + i$

## 6.3 Modular Composition

**Modular Composition Problem**

Input: $f, g, h \in R[x]$ with $\deg(g), \deg(h) < \deg(f) = n$, $R$ is a ring and $f$ is monic.
Output: $g(h) \pmod f$

The idea is to follow the Babystep-Giantstep idea:

- Let $m = \lceil \sqrt{n} \rceil$.
- Similar to what we have seen, we can write

$$g = \sum_{i=0}^{m-1} g_i x^{mi}$$

  with $g_i \in R[x]$ of degree less than $m$.
- For $i < m$, let

$$g_i = \sum_{j=0}^{m-1} g_{ij} x^j$$

  with $g_{ij} \in R$.
- Then

$$g_i(h) \pmod f = \underbrace{\sum_{j=0}^{m-1} g_{ij} \cdot (h^j \pmod f)}_{r_i}$$

$$\begin{bmatrix} \text{Coefficients of } g_0 \\ \hline \text{Coefficients of } g_1 \\ \hline \vdots \\ \hline \text{Coefficients of } g_{m-1} \end{bmatrix} \cdot \begin{bmatrix} 1 \\ \hline \text{Coefficients of } h \mod f \\ \hline \vdots \\ \hline \text{Coefficients of } h^{m-1} \mod f \end{bmatrix} = \begin{bmatrix} \text{Coefficients of } r_0 \\ \hline \text{Coefficients of } r_1 \\ \hline \vdots \\ \hline \text{Coefficients of } r_{m-1} \end{bmatrix}$$

---

**Input**: $f, g, h \in R[x]$ with $\deg(g), \deg(h) < \deg(f) = n$ where $f$ is monic.
**Output**: $g(h) \pmod f \in R[x]$.
$m \leftarrow \lceil \sqrt{n} \rceil$
$g = \sum_{i=0}^{m-1} g_i x^{mi}$ with $g_i \in R[x]$ of degree less than $m$
Compute $r_i$'s by forming matrices in image and multiply them
**return** $b = \sum_{0 \le i < m} r_i (h^m)^i$ using Horner's Rule

---

The cost of matrix multiplication is currently $O(n^{2.37})$ by Coppersmith-Winograd.

# Chapter 7

# Linearly Recurrent Sequences

## 7.1 Rational Reconstruction

With Newton iteration, given polynomials $N(x)$ and $D(x)$, we can expand

$$S(x) = \frac{N(x)}{D(x)} = s_0 + s_1 x + s_2 x^2 + \cdots$$

Assuming you know sufficiently many terms, can we recover $N(x)/D(x)$.

Suppose that

- $\deg(N) \leq n$ and $\deg(D) \leq d$.
- We know $s_0, \ldots, s_{n+d}$.

$S = \frac{N}{D} \pmod{x^{n+d+1}} \implies DS = N \pmod{x^{n+d+1}} \implies Rx^{n+d+1} + DS = N$.

We run the extended Euclidean algorithm with input $A_0 = x^{n+d+1}$ and $A_1 = s_0 + \cdots + s_{n+d}x^{n+d}$.

1. For $i = 0$, let $U_0 = 1, V_0 = 0$.
2. For $i = 1$, let $U_1 = 0, V_1 = 1$.
3. For $i \geq 2$
    - $Q_i = A_{i-1} \text{ div } A_i$
    - $A_{i+1} = A_{i-1} - Q_i A_i$
    - $U_{i+1} = U_{i-1} - Q_i U_i$
    - $V_{i+1} = V_{i-1} - Q_i V_i$

At each step, we maintain the invariant

$$U_i x^{n+d+1} + V_i(s_0 + s_1 x + \cdots + s_{n+d}x^{n+d}) = A_i$$

The sequence of degrees of $A_i$ decrease. The sequence of degrees of $V_i$ increase.

Since $V_{i+1} = V_{i-1} - Q_i V_i$, $\deg(V_2) = \deg(Q_1)$ and $\deg(V_3) = \deg(Q_1) + \deg(Q_2)$, and $\deg(V_i) = \sum_{j=1}^{i-1} \deg(Q_j) = \sum_{j=1}^{i-1}(\deg(A_{j-1}) - \deg(A_j)) = \deg(A_0) - \deg(A_{i-1}) = n+d+1-(n+1) \leq d$.

**Proposition**

Let $i$ be the first index with $\deg(A_i) \leq n$. Then $\deg(V_i) = n+d+1-\deg(A_{i-1}) \leq d$. Hence $A_i/V_i = N/D$.

**E.g.** Find the next term:

$$\begin{aligned} U &: 1,1,1,1,1,1,1,1 \\ V &: 0,1,1,2,3,5,8,13 \\ W &: 12, 134, 222, 21, -3898, -40039, -347154, -2929918, -24657854 \end{aligned}$$

The next terms are 1, 21, and -207605083. They satisfy the following linear recurrences

$$\begin{aligned} U_{n+1} &= U_n \\ V_{n+2} &= V_{n+1} + V_n \\ W_{n+4} &= 12W_{n+3} - 33W_{n+2} + 22W_{n+1} + 19W_n \end{aligned}$$

**Definition: Generating Series**

Given a sequence $(s_0, s_1, \dots)$, we can construct the generating series

$$S = \sum_{i \geq 0} s_i x^i$$

**E.g.** Let $s_n = 2^n$, i.e. $s_0 = 1$ and $s_{n+1} = 2s_n$. Then the generating series

$$S = \sum_i 2^i x^i = \frac{1}{1-2x}$$

**Proposition**

The generating series $S = \frac{N(x)}{D(x)}$ is rational with $D(x) = 1 + a_{k-1}x + \cdots + a_1 x^{k-1} + a_0 x^k$ and $\deg(N) < \deg(D)$ if and only if the sequence $(s_n)_{n\geq 0}$ satisfies the recurrence

$$s_{n+k} + a_{k-1}s_{n+k-1} + \cdots + a_1 s_{n+1} + a_0 s_n = 0, a_0 \neq 0$$

Basically we have a rational series if and only if recurrence with constant coefficients.

Check with recurrences of order 2:

$$s_0 = \alpha, s_1 = \beta, s_{n+2} + as_{n+1} + bs_n = 0$$

and $S = \sum_{i \geq 0} s_i x^i$. Multiply the relation by $x^{n+2}$:

$$s_{n+2}x^{n+2} + as_{n+1}x^{n+2} + bs_n x^{n+2} = 0$$

And sum for $n \geq 0$:

$$S - (\alpha + \beta x) + ax(S - \alpha) + bx^2 S = 0$$

Rearranging

$$S = \frac{\alpha + (\beta + \alpha a)x}{1 + ax + bx^2}$$

Consequence: Suppose that you know a sequence $(s_t)_{t \geq 0}$ satisfies a recurrence of order $k$:

- Set $n = k - 1, d = k$.
- Need $s_0, \dots, s_{n+d}$ up to $s_{2k-1}$.
- Apply the Extended Euclidean Algorithm to

$$A_0 = x^{2k}, A_1 = s_0 + \cdots + s_{2k-1}x^{2k-1}$$

- Stop at the first $i$ with $\deg(A_i) \leq k - 1$.

**E.g.** Fibonacci Numbers $(s_n)_{n \geq 0} = (1, 1, 2, 3, 5, 8, \dots)$ and suppose we know it satisfies a recurrence of order $k = 2$. We apply the XGCD algorithm to $A_0 = x^4$ and $A_1 = 1 + x + 2x^2 + 3x^3$. So we find

| $i$ | $A_i$ | $U_i$ | $V_i$ |
|---|---|---|---|
| 0 | $x^4$ | ? | 0 |
| 1 | $1 + x + 2x^2 + 3x^3$ | ? | 1 |
| 2 | $\frac{1}{9}(2 - x - x^2)$ | ? | $\frac{1}{9}(2 - 3x)$ |
| 3 | $-9$ | ? | $9(-1 + x + x^2)$ |

$$1 + x + 2x^2 + 3x^3 + 5x^4 + \cdots = \frac{-9}{9(-1 + x + x^2)} = \frac{1}{1 - x - x^2}$$

and the recurrence

$$s_{n+2} - s_{n+1}s_n = 0$$

# Chapter 8

# Sparse Linear Systems

Dense matrices have mostly entries nonzero and usually not big. Gauss' algorithm takes $O(n^3)$. Sparse matrices have mostly zero entries and can be very big. Iterative algorithms aim at $O(n^2)$.

Matrix-vector product takes $O(n^2)$, which is optimal in general. Matrix-matrix product takes $O(n^3)$ naively.

Special case: When $A$ is very sparse, only $O(1)$ nonzero entry per row. Then 1 vector product in $O(1)$ operations. 1 matrix-vector product $Ab$ in $O(n)$ operations. 1 matrix-matrix product $AB$ in $O(n^2)$ operations.

## 8.1 Polynomials of Matrices

**Definition: Polynomial of $A$**

Given a univariate polynomial $P = p_0 + p_1 x + \cdots + p_d x^d$ and a matrix $A$,

$$P(A) = p_0 I + p_1 A + \cdots + p_d A^d$$

**Proposition**

For any matrix $A$ of size $n$, there exists one (or more) polynomial(s) $P$ of degree at most $n$ such that $P(A) = 0$.

**Definition: Minimal Polynomial of $A$**

The monic polynomial $P$ of smallest degree such that $P(A) = 0$, written $m_A$.

Diagonal matrices: Let $A = \begin{bmatrix} c & 0 \\ 0 & c \end{bmatrix}$, then $m_A = x - c$.

**E.g.** Let $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$. $A^0 = I, A^1 = A, A^2 = \begin{bmatrix} 7 & 10 \\ 15 & 22 \end{bmatrix}$. We look for $m_A = p_0 + p_1 x + x^2$ so that $p_0 A^0 + p_1 A + A^2 = 0$. This matrix is

$$\begin{bmatrix} p_0 + p_1 + 7 & 2p_1 + 10 \\ 3p_1 + 15 & p_0 + 4p_1 + 22 \end{bmatrix}$$

This gives $p_0 = -2, p_1 = -5$ and $m_A = -2 - 5x + x^2$.

## 8.2 Linear Recurrence for Matrices

**Definition: Linear Recurrence with Constant Coefficients**

Let $m_A = p_0 + p_1 x + \cdots + p_{d-1} x^{d-1} + x^d$ be the minimal polynomial of $A$. Then, the sequence $I, A, A^2, \ldots$ satisfies

$$\begin{aligned} p_0 I + p_1 A + \cdots + A^d &= 0 \\ p_0 A + p_1 A^2 + \cdots + A^{d+1} &= 0 \\ &\vdots \\ p_0 A^m + p_1 A^{m+1} + \cdots + A^{d+m} &= 0 \end{aligned}$$

Given a matrix $A$ of size $n$, choose (at random) two vectors

$$u = \begin{bmatrix} u_1 & \ldots & u_n \end{bmatrix}, v = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix}$$

Then define

$$a_0 = uA^0 v, a_1 = uA^1 v, \ldots, a_i = uA^i v, \ldots$$

**E.g.** With $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ and $u = \begin{bmatrix} 1 & 2 \end{bmatrix}$ and $v = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$, we get

$$a_0 = -3, a_1 = -13, a_2 = -71, a_3 = -381, a_4 = -2047, a_5 = -10997$$

Starting from

$$p_0 I + p_1 A + \cdots + A^d = 0$$

we can multiply on the left by $u$ and on the right by $v$, and get 0. For any choice $u, v$,

$$p_0 a_0 + p_1 a_1 + \cdots + a_d = 0$$

More generally, we can pre-multiply by $A^m$

$$p_0 A^m + p_1 A^{m+1} + \cdots + A^{m+d} = 0$$

and multiply left by $u$ and right by $v$ to get

$$p_0 a_m + p_1 a_{m+1} + \cdots + a_{m+d} = 0$$

The sequence $(a_m)_{m \geq 0}$ satisfies a linear recurrence with constant coefficients.

## 8.3 Finding the Minimal Polynomial

Compute $2n$ values $a_0, \dots, a_{2n-1}$.
Find the recurrence for $(a_m)$.
Hope it is the minimal polynomial of $A$.

**Proposition**

For most choices of $u$ and $v$, we find $m_A$. In unlucky situations, we may find a factor only.
Precisely, there is a polynomial $\Delta(U_1, \dots, U_n, V_1, \dots, V_n)$ such that if

$$\Delta(u_1, \dots, u_n, v_1, \dots, v_n) \neq 0$$

we are good.

**E.g.** For $A = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$, $u = \begin{bmatrix} 1 & 0 \end{bmatrix}$ and any $v$, we are unlucky.

## 8.4 Solving Systems

Let $m_A = p_0 + p_1 x + \cdots + p_{d-1}x^{d-1} + x^d$ and assume that $p_0 \neq 0$. Given a vector $c$, we can use $m_A$ to solve the system $Ab = c$.

**Proposition**

Define

$$b = -\frac{1}{p_0}(p_1 I + \cdots + p_{d-1}A^{d-2} + A^{d-1})c$$

Then $Ab = c$.

Complexity:

1. Computing $a_i = uA^i v$ up to $i = 2n-1$ includes computing $v, Av, A^v, \dots$ and deducing $uv, uAv, uA^2v, \dots$. Total is $O(n)$ matrix-vector products by $A$, $O(n^2)$ other operations.

2. Computing $m_A$ is $O(n^2)$ by Euclid's algorithm.

3. Computing $b$ includes computing $c, Ac, A^2c, \dots,$ and deducing $p_1 c + p_2 Ac + \cdots$. Total is $O(n)$ matrix-vector products by $A$ and $O(n^2)$ other operations.

The total cost is $O(n)$ matrix-vector products by $A$ and $O(n^2)$ other operations.

When $A$ is dense, a matrix-vector product by $A$ takes $O(n^2)$ operations, so the total is $O(n^3)$. When $A$ is sparse, and has $O(1)$ nonzero entries per row, a matrix-vector product takes $O(n)$ time, so total is $O(n^2)$.

**E.g.** Factor an integer $N$.

Basic idea: $A^2 \mod N = B^2 \mod N \Leftrightarrow N$ divides $(A+B)(A-B)$, hope that $\gcd(N, A+B)$ and $\gcd(N, A-B)$ are not trivial. Take $N = 2183$ and suppose we have guessed that $96002478^2 \mod N = 21^2 \mod N$. Then we get the gcds $\gcd(96002478 + 21, N) = 59$ and $\gcd(96002478 - 21, N) = 37$.

# Chapter 9

# Matrix Multiplication

## 9.1 Matrix Multiplication

Let $A = \begin{bmatrix} a_{1,1} & \dots & a_{1,n} \\ \vdots & & \vdots \\ a_{n,1} & \dots & a_{n,n} \end{bmatrix}$ and $B = \begin{bmatrix} b_{1,1} & \dots & b_{1,n} \\ \vdots & & \vdots \\ b_{n,1} & \dots & b_{n,n} \end{bmatrix}$, then

$$AB = \begin{bmatrix} \dots & a_{1,1}b_{1,j} + \dots + a_{1,n}b_{n,j} & \dots \\ & \dots & \\ \dots & a_{n,1}b_{1,j} + \dots + a_{n,n}b_{n,j} & \dots \end{bmatrix}$$

**Algorithm 8** NaiveProduct$(A, B)$

```
for i = 1, ..., n do
    for j = 1, ..., n do
        c_ij = 0
        for k = 1, ..., n do
            c_ij = c_ij + a_{i,k} b_{k,j}
```

Total of $n^3$ multiplications, $n^3 - n^2$ additions.

**Theorem (Multiplication of Matrices)**

$O(n^3)$ for naive algorithm, $O(n^{\log_2 7})$ for Strassen's algorithm, $O(n^{2.38})$ using Coopersmith and Winograd's algorithm.

We let $\omega$ be the exponent of the runtime $O(n^\omega)$ for matrix multiplication. Inverse of matrices can be done in $O(n^\omega)$.

### 9.1.1 Pre-Strassen

Winograd's algorithm for dot-product: $A = \begin{bmatrix} a_1 & a_2 & \cdots \end{bmatrix}, B = \begin{bmatrix} b_1 & b_2 & \cdots \end{bmatrix}^T$,

$$\begin{aligned} (a_1 + b_2)(a_2 + b_1) - a_1 a_2 - b_2 b_1 &= a_1 b_1 + a_2 b_2 \\ (a_3 + b_4)(a_4 + b_3) - a_3 a_4 - b_4 b_3 &= a_3 b_3 + a_4 b_4 \\ &\cdots \end{aligned}$$

$n/2$ multiplications depending on $(A, B)$. $n/2$ depending only on $A$, $n/2$ depending only on $B$.

Winograd's algorithm for matrix product does a dot product between all rows $A_i$ and all columns $B_j$. $n^2 \cdot n/2$ multiplications depending on pairs $(A_i, B_j)$. $n \cdot n/2$ depending on $A_i$'s, $n \cdot n/2$ depending on $B_j$'s. Total is $\frac{1}{2}n^3 + n^2$.

We cannot make this into a recursive algorithm if the entries are matrices since $a_2 b_2 \neq b_2 a_2$ in general for matrices.

### 9.1.2 Strassen

Find an improvement to the $2 \times 2$ case. We can compute 7 linear combinations, multiply pairwise and recombine.

$$\begin{aligned} q_1 &= (a_{1,1} - a_{1,2}) b_{2,2} \\ q_2 &= (a_{2,1} - a_{2,2}) b_{1,1} \\ q_3 &= a_{2,2}(b_{1,1} + b_{2,1}) \\ q_4 &= a_{1,1}(b_{1,2} + b_{2,2}) \\ q_5 &= (a_{1,1} + a_{2,2})(b_{2,2} - b_{1,1}) \\ q_6 &= (a_{1,1} + a_{2,1})(b_{1,1} - b_{1,2}) \\ q_7 &= (a_{1,2} + a_{2,2})(b_{2,1} - b_{2,2}) \end{aligned}$$

So the entries are

$$\begin{aligned} c_{1,1} &= q_1 - q_3 - q_5 + q_7 \\ c_{1,2} &= q_4 - q_1 \\ c_{2,1} &= q_2 + q_3 \\ c_{2,2} &= -q_2 - q_4 + q_5 + q_6 \end{aligned}$$

Now let $A$ and $B$ have size $n \times n$ with $n = 2^k$. We can break them into blocks:

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix}, B = \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix}$$

where each $A_{i,j}, B_{i,j}$ have size $n/2 \times n/2$. The formulas for the $2 \times 2$ case still work. There are 7 products in size $n/2$ and $O(n^2)$ extra operations (from $O(1)$ additions/subtractions of matrices of size $n/2$).

Complexity: Let $MM(n)$ be the cost of matrix multiplication of size $n$. Then we have

$$MM(n) \leq 7MM(n/2) + \lambda n^2$$

so

$$M(n) \leq Cn^{\frac{\log 7}{\log 2}} \leq Cn^{2.81}$$

***Proof.*** Master theorem.

More generally, if you find an algorithm that does $k$ multiplications in size $n$, then you can take $\omega = \log k / \log n$.

## 9.2 Rectangular Matrices

**Definition: Minimum Number of Multiplications**

Let $\langle n, m, p\rangle$ be the minimal number of multiplications it takes to multiply matrices of size $(n, m) \times (m, p)$.

$\langle 2, 2, 2\rangle = 7$. $\langle 3, 3, 3\rangle$ we do not know, but it is $\leq 23$.

**Proposition**

If $\langle n, n, n\rangle \leq k$, then we can take

$$\omega = \frac{\log k}{\log n}$$

**Proposition**

If $\langle m, n, p\rangle \leq k$, then we can take

$$\omega = \frac{3 \log k}{\log(nmp)}$$

***Proof.*** (Block Matrices) Show $\langle mm', nn', pp'\rangle \leq \langle m, n, p\rangle \langle m', n', p'\rangle$.

Suppose we have to multiply $A$ of size $(mm', nn')$ by $B$ of size $(nn', pp')$. We can decompose them into blocks of size $(m', n')$ in $A$ and size $(n', p')$ in $B$. So then $A$ has size $(m, n)$ of these $(m', n')$ blocks and $B$ has size $(n, p)$ of these $(n', p')$ blocks. The product is $C$ with each block of size $(m', p')$.

To compute $AB$, we apply the algorithm in size $(m, n, p)$ so $\langle m, n, p\rangle$ products of blocks. Then each of the products is done on blocks of size $(m', n', p')$, so it costs $\langle m', n', p'\rangle$ products in the base field. So the total number of multiplications is $\langle m, n, p\rangle \langle m', n', p'\rangle$.

(Permutations) Show $\langle m, n, p\rangle = \langle n, p, m\rangle = \langle p, m, n\rangle$.

Proposition: $B^T A^T = (AB)^T$. The consequence of this is $\langle m, n, p\rangle = \langle p, n, m\rangle$.

(Conclusion)

$$\langle mnp, mnp, mnp\rangle \leq \langle m,n,p\rangle \langle np, mp, mn\rangle \leq \langle m,n,p\rangle \langle n.p.m\rangle \langle p,m,n\rangle \leq k^3$$

so $\omega = \frac{\log k^3}{\log(mnp)} = \frac{3\log k}{\log(mnp)}$.

## 9.3 Polynomial Notation

We want to describe the multiplication

$$(a_0 + a_1X)(b_0 + b_1X) = a_0b_0 + (a_1b_0 + a_0b_1)X + a_1b_1X^2 = C_0 + C_1X + C_2X^2$$

We can describe this operation using a polynomial in variables $(A_0, A_1), (B_0, B_1), (C_0, C_1, C_2)$:

$$P = A_0B_0C_0 + A_0B_1C_1 + A_1B_0C_1 + A_1B_1C_2$$

**Polynomial notation for Karatsuba**: Recall Karatsuba

$$(a_0 + a_1x)(b_0 + b_1x) = a_0b_0 + ((a_0 + a_1)(b_0 + b_1) - a_0b_0 - a_1b_1)x + a_1b_1x^2$$

So the polynomial notation is

$$P = A_0B_0(C_0 - C_1) + (A_0 + A_1)(B_0 + B_1)C_1 + A_1B_1(C_2 - C_1)$$

We compute $A_0B_0$, add result to $C_0$ and subtract from $C_1$. Compute $(A_0 + A_1)(B_0 + B_1)$, add result to $C_1$. Compute $A_1B_1$, add result to $C_2$ and subtract from $C_1$.

**Polynomial notation for $2 \times 2$ matrices**:

$$\begin{aligned} P_{mat2} = {} & A_{1,1}B_{1,1}C_{1,1} + A_{1,2}B_{2,1}C_{1,1} + A_{1,1}B_{1,2}C_{1,2} \\ & + A_{2,1}B_{1,1}C_{2,1} + A_{2,2}B_{2,1}C_{2,1} + A_{2,1}B_{1,2}C_{2,2} + A_{2,2}B_{2,2}C_{2,2} \end{aligned}$$

Compute $A_{1,1}B_{1,1}$ and add it to $C_{1,1}$, compute $A_{1,2}B_{2,1}$ and add it to $C_{1,1}$, etc.

**Polynomial notation for Strassen's Algorithm**:

$$\begin{aligned} P_{mat2} = {} & (A_{1,1} - A_{1,2})B_{2,2}(C_{1,1} - C_{1,2}) + \\ & (A_{2,1} - A_{2,2})B_{1,1}C_{2,1} + \\ & A_{2,2}(B_{1,1} + B_{2,1})(-C_{1,1} + C_{2,1}) + \\ & A_{1,1}(B_{1,2} + B_{2,2})(C_{1,2} - C_{2,2}) + \\ & \cdots \end{aligned}$$

Compute $(A_{1,1} - A_{1,2})B_{2,2}$ and add it to $C_{1,1}$, subtract it from $C_{1,2}$, etc.

**Polynomial notation for $(1,2) \times (2,3)$**:

$$\begin{bmatrix} A_{1,1} & A_{1,2} \end{bmatrix} \begin{bmatrix} B_{1,1} & B_{1,2} & B_{1,3} \\ B_{2,1} & B_{2,2} & B_{2,3} \end{bmatrix} = \begin{bmatrix} C_{1,1} & C_{1,2} & C_{1,3} \end{bmatrix}$$

The polynomial notation is

$$\begin{aligned} P_{mat123} = &A_{1,1}B_{1,1}C_{1,1} + A_{1,2}B_{2,1}C_{1,1}+ \\ &A_{1,1}B_{1,2}C_{1,2} + A_{1,2}B_{2,2}C_{1,2}+ \\ &A_{1,1}B_{1,3}C_{1,3} + A_{1,2}B_{2,3}C_{1,3} \end{aligned}$$

**Polynomial notation for** $(2,3) \times (3,1)$:

$$\begin{bmatrix} A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,1} & A_{2,2} & A_{2,3} \end{bmatrix} \begin{bmatrix} B_{1,1} \\ B_{2,1} \\ B_{3,1} \end{bmatrix} = \begin{bmatrix} C_{1,1} \\ C_{2,1} \end{bmatrix}$$

The polynomial notation is

$$\begin{aligned} P_{mat231} = &A_{1,1}B_{1,1}C_{1,1} + A_{1,2}B_{2,1}C_{1,1} + A_{1,3}B_{3,1}C_{1,1}+ \\ &A_{2,1}B_{1,1}C_{2,1} + A_{2,2}B_{2,1}C_{2,1} + A_{2,3}B_{3,1}C_{2,1} \end{aligned}$$

As we see, up to replacing $A_{i,j}$ by $C_{j,i}$, $B_{i,j}$ by $A_{i,j}$ by $A_{i,j}$, and $C_{i,j}$ by $B_{j,i}$, $P_{mat123}$ and $P_{mat231}$ are the same polynomials, so an algorithm for $P_{mat231}$ gives an algorithm for $P_{mat123}$, i.e. $\langle 2,3,1 \rangle = \langle 1,2,3 \rangle$.

# Chapter 10

# Fast Evaluation/Interpolation

## 10.1 Evaluation and Interpolation of Polynomials

### 10.1.1 Evaluation

**Fast Evaluation**

Given $n$ points $a_0, \ldots, a_{n-1}$ in a field $\mathbb{F}$, $F$ in $\mathbb{F}[x]$ of degree less than $n$, what is the complexity of computing $F(a_0), \ldots, F(a_{n-1})$.

Naive algorithm: Compute each $F(a_i)$, the cost is $O(n \times n) = O(n^2)$.

Quasi-linear algorithm: Can solve in $O(n \log n)$ if $n$ is a power of 2 and $a_i$'s are roots of unity of order $n$ by FFT. This is $O(M(n) \log n)$ in general.

**Proposition (Key Property 1)**

$F(a) = F \mod (x - a)$.

**Proposition (Key Property 2)**

Let $A, B, C \in \mathbb{F}[x]$ be such that $B$ divides $C$. Then

$$A \mod B = (A \mod C) \mod B$$

***Proof.*** $C = DB$ by assumption, so if we write

$$A = QC + R, R = A \mod C$$

we get

$$A = QDB + R$$

So $A \mod B = 0 + R \mod B$.

Divide-and-conquer algorithm: With $n = 4$,

- Compute $F_0 = F \mod (x-a_0)(x-a_1)$
  - Compute $F_{00} = F_0 \mod (x-a_0) = F(a_0)$
  - Compute $F_{01} = F_0 \mod (x-a_1) = F(a_1)$
- Compute $F_1 = F \mod (x-a_2)(x-a_3)$
  - Compute $F_{10} = F_1 \mod (x-a_2) = F(a_2)$
  - Compute $F_{11} = F_1 \mod (x-a_3) = F(a_3)$

**Definition: Subproduct Tree**

A tree $\mathcal{T}$ of polynomials built bottom-up by its factors.

There are $\log n$ levels. Building the tree takes

$$T(n) = 2T(n/2) + M(n) \implies T(n) \in O(M(n)\log n)$$

**Algorithm 9** Evaluate($F, \mathcal{T}$)

```
F = F mod root(T)
if T is a leaf then
    return F
return Evaluate(F, left(T)), Evaluate(F, right(T))
```

Runtime:

$$E(n) = 2E(n/2) + O(M(n)) \implies E(n) \in O(M(n)\log n)$$

### 10.1.2 Lagrange Interpolation

**Interpolation**

Given pairwise distinct points $a_0, \ldots, a_{n-1}$ in a field $\mathbb{F}$ and values $v_0, \ldots, v_{n-1}$, there is a unique polynomial $P$ of degree less than $n$ such that

$$P(a_i) = v_i$$

This is given by

$$P = \sum -i = 0^{n-1} v_i \prod_{j\neq i} \frac{x-a_j}{a_i-a_j} = \sum_{i=1}^{n-1} \left( \underbrace{\frac{v_i}{\prod_{j\neq i}(a_i-a_j)}}_{u_i} \prod_{j\neq i}(x-a_j) \right)$$

Given $v_0, \ldots, v_{n-1}$, we first compute

$$u_i = \frac{v_i}{\prod_{j\neq i}(a_i-a_j)}$$

If we define

$$M = (x - a_0) \dots (x - a_{n-1})$$

The formal derivative is

$$M' = \sum_{i=0}^{n-1} \frac{M}{(x - a_i)}$$

We have

$$u_i = \frac{v_i}{\prod_{j \neq i}(a_i - a_j)} = \frac{v_i}{M'(a_i)}$$

Runtime: $O(M(n) \log n)$.

**Going up the tree**: With $n = 4$,

$$\begin{aligned}
P &= u_0(x - a_1)(x - a_2)(x - a_3) + u_1(x - a_0)(x - a_2)(x - a_3) \\
&\quad + u_2(x - a_0)(x - a_1)(x - a_3) + u_3(x - a_0)(x - a_1)(x - a_2) \\
&= (u_0(x - a_1) + u_1(x - a_0))(x - a_2)(x - a_3) \\
&\quad + (u_2(x - a_3) + u_3(x - a_2))(x - a_0)(x - a_1)
\end{aligned}$$

Runtime: $O(M(n) \log n)$.

**Algorithm 10** Combine($u_0, \dots, u_{n-1}, \mathcal{T}$)

```
if T is a leaf then
    return u_0
F_0 = Combine(u_0, ..., u_{n/2-1}, left(T))
F_1 = Combine(u_{n/2}, ..., u_{n-1}, right(T))
return F_0 × root(right(T)) + F_1 × root(left(T))
```

# Chapter 11

# Reed-Solomon Codes

## 11.1 Finite Fields and Reed-Solomon Codes

Error-correcting codes allow to detect and correct errors in digital messages. Our messages will be made of symbols from a finite alphabet.

**E.g.** Our alphabet can be the hexadecimal digits, message we send is $m$ and we receive $r$.

We can view the problem in an algebraic context. Reed-Solomon codes are defined in terms of polynomials over finite fields. Encoding is the evaluation of polynomials and error correction is rational reconstruction.

Recall rings and fields (any elements $x \neq 0$ can be inverted). Some field examples ($\mathbb{Q}, \mathbb{R}, \mathbb{C}, \mathbb{Z}/2\mathbb{Z} = \mathbb{F}_2 = \{0, 1\}$ with operations mod 2). $\mathbb{Z}, \mathbb{Z}/4\mathbb{Z}$ are not fields.

**Complex Numbers**: $\mathbb{C} = \mathbb{R}[x]/(x^2 + 1)$. A complex number can be seen as a polynomial $a + bx$ with operations done modulo $x^2 + 1$.

**Binary Fields**: Take an irreducible polynomial $P$ of degree $d$ in $\mathbb{F}_2[x]$. Then

$$\mathbb{F}_{2^d} = \mathbb{F}_2[x]/P$$

is a field, with $2^d$ elements.

**Claim**

For any $d$, there exists an irreducible polynomial of degree $d$ over $\mathbb{F}_2$.

For $\mathbb{F}_4$, the polynomial $P = x^2 + x + 1 \in \mathbb{F}_2[x]$ is irreducible.

***Proof.*** The polynomials of degree 1 are $x$ and $x + 1$ and they do not divide $P$.

**Corollary**

$\mathbb{F}_2[x]/P$ is a field.

The elements are $0, 1, x, x+1$. Can look at multiplication table and see every nonzero polynomial has an element that it can multiply by to get 1.

$Q = x^2 + 1 \in \mathbb{F}_2[x]$ is not irreducible, since $x^2 + 1 = (x+1)^2$. Thus, $\mathbb{F}_2[x]/Q$ is not a field. (Can look at multiplication table for $0, 1, x, x+1$.

**Proposition**

- One addition in $\mathbb{F}_{2^d}$ takes $d$ operations in $\mathbb{F}_2$.
- One multiplication in $\mathbb{F}_{2^d}$ takes $O(M(d))$ operations in $\mathbb{F}_2$ using Fast Euclidean division.
- One inversion in $\mathbb{F}_{2^d}$ takes $O(M(d) \log d)$ operations in $\mathbb{F}_2$ using Fast xGCD.

For small fields, say $2^{16}$ elements, you can use a table look-up instead.

### 11.1.1 Zech Logarithms

In any finite field $\mathbb{F}$ of size $q$, there is at least one element $\alpha$ such that

$$F - \{0\} = \{1, \alpha, \alpha^2, \ldots, \alpha^{q-2}\}$$

in particular $\alpha^{q-1} = 1$.

**E.g.** $\mathbb{F}_4 = \{0, 1, x, x+1\}, q = 4, \alpha = x$. $\alpha^0 = 1, \alpha^1 = x, \alpha^2 = x + 1$.

So we can represent a nonzero element $\alpha^i$ using its logarithm $i$.

Multiplication: $\alpha^i \alpha^j = \alpha^{i+j \mod (q-1)}$.

Addition: $\alpha^i + \alpha^j = \alpha^i(1 + \alpha^{j-i})$, so we store $z_0, \ldots, z_{q-2}$ such that

$$1 + \alpha^n = \alpha^{z_n}$$

## 11.2 Sparse Interpolation

Points $(a_i)$ and values $(v_i)$, but now assume

- $P$ has at most $s$ terms.
- Do not know their degrees.

We can still find it, if

- There is $\alpha$ such that for all $i$
$$a_i = \alpha^i$$

- We change the bound to $s/2$ terms, say

$$P = \sum_{i=1}^{\ell} c_i x^{m_i}, \ell \leq s/2$$

- All $m_i$ are less than the order of $\alpha$ = the smallest integer $e > 0$ such that $\alpha^e = 1$.

We write

$$P = \sum_{i=1}^{\ell} c_i x^{m_i}$$

all $c_i$ nonzero and $\ell \leq s/2$. Then

$$\begin{aligned} S := \sum_{k \geq 0} P(\alpha^k) x^k &= \sum_{k \geq 0} \sum_{i=1}^{\ell} c_i (\alpha^k)^{m_i} x^k \\ &= \sum_{i=1}^{\ell} c_i \sum_{k \geq 0} (\alpha^{m_i})^k x^k \\ &= \sum_{i=1}^{\ell} \frac{c_i}{1 - \alpha^{m_i} x} \end{aligned}$$

**Step 1 (Rational Reconstruction)**: We can rewrite $S$ as

$$S = \frac{N(x)}{D(x)} = \frac{\sum_i c_i \prod_{j \neq i} (1 - \alpha_j^m x)}{\prod_i (1 - \alpha^{m_i} x)}$$

By assumption, all $\alpha^{m_i}$ are pairwise distinct, so that $N(x)$ and $D(x)$ have no common factor. Their degrees are at most $\ell - 1$ and $\ell$.

In this case, rational reconstruction applied to

$$P(1), P(\alpha), \ldots, P(\alpha^{s-1})$$

gives us $N(x)$ and $D(x)$.

Given

$$N(x) = \sum_i c_i \prod_{j \neq i} (1 - \alpha^{m_j} x)$$

and

$$D(x) = \prod_i (1 - \alpha^{m_i} x)$$

we deduce

$$\tilde{N}(x) = \sum_i c_i \prod_{j \neq i} (x - \alpha^{m_j}), \tilde{D}(x) = \prod_i (x - \alpha^{m_i})$$

Explicitly,

$$\tilde{N}(x) = x^{\ell-1} N(1/x), \tilde{D}(x) = x^{\ell} D(1/x)$$

**Step 2 (Finding the $m_i$'s)**:

Option 1: assume we know $M$ is not too large such that $m_i \leq M$ for all $i$. Then, compute

$$\tilde{D}(1), \tilde{D}(\alpha), \ldots, \tilde{D}(\alpha^M)$$

and record the entries for which we find a zero.

Option 2: Compute all the roots of $\tilde{D}(x)$, say $r_1, \ldots, r_\ell$. Then for all $i$, we have to find $m_i$ such that

$$\alpha^{m_i} = r_i$$

## 11.3 Reed-Solomon Codes

**Definition: Reed-Solomon** $RS(n, k)$

The alphabet is a binary field $\mathbb{F} = \mathbb{F}_{2^d}$, the message to encode is a polynomial $P$ of degree $< k$, and the encoded message is a polynomial $PG$ for some $G$ of degree $n - k$.

There are $k$ symbols before encoding and $n$ symbols after encoding. We add $n - k$ symbols.

We write $t = (n - k)/2$ to say we will be able to fix $t$ errors. DVDs use $RS(208, 192)$ over $\mathbb{F}_{256}$.

**Choosing G**: We take

$$G = (x - 1)(x - a) \ldots (x - a^{n-k-1})$$

where $a$ is chosen such that all powers are pairwise distinct.

Do not take $a = 0$ or $a = 1$. We want that $1, a, a^2, \ldots, a^{n-1}$ are pairwise distinct.

**Transmission Errors**: We may not receive $S$, but another polynomial $R$ of degree $< n$:

$$R = S + E = PG + E$$

with

- $R$ = received polynomial (known)
- $S = PG$ = message (unknown)
- $E$ = error (unknown)

**Key Result**

If there are at most $t$ errors, we can reconstruct $S$ from $R$.

**Step 1 (Get Values of $E$)**: We compute

$$R(a^i), i = 0, \ldots, n - k - 1$$

Because

$$G(a^i) = 0, i = 0, \ldots, n-k-1$$

and $R = PG + E$ gives us

$$E(a^i), i = 0, \ldots, n-k-1$$

**Step 2 (Get $E$)**:

1. Polynomial $E$ has the form

$$E = \sum_{i=0}^{\ell-1} c_i x^{m_i}$$

for some $\ell \leq (n-k)/2$. The $m_i$'s are all less than $n$.

2. We know the values of $E$ at $1, a, \ldots, a^{n-k-1}$.

3. By assumption, $1, a, \ldots, a^{n-1}$ are all pairwise distinct.

We can reconstruct $E$ by sparse interpolation. Knowing $E$ gives us $PG = R - E$ and thus $P$.

Example: $RS(8,4)$ over $\mathbb{F}_{16}$.

The alphabet is $\mathbb{F}_{16} = \mathbb{F}_2[x]/(z^4 + z + 1)$. So the symbols are

| | | | |
|---|---|---|---|
| $0$ | $1 = \alpha^0$ | $z = \alpha$ | $z + 1 = \alpha^4$ |
| $z^2 = \alpha^2$ | $z^2 + 1 = \alpha^8$ | $z^2 + z = \alpha^5$ | $z^2 + z + 1 = \alpha^{10}$ |
| $z^3 = \alpha^3$ | $z^3 + 1 = \alpha^{14}$ | $z^3 + z = \alpha^9$ | $z^3 + z + 1 = \alpha^7$ |
| $z^3 + z^2 = \alpha^6$ | $z^3 + z^2 + 1 = \alpha^{13}$ | $z^3 + z^2 + z = \alpha^{11}$ | $z^3 + z^2 + z + 1 = \alpha^{12}$ |

Our messages have length 4, so they are polynomials

$$u_0 + u_1 x + u_2 x^2 + u_3 x^3$$

with all $u_i$'s in $\mathbb{F}_{16}$. We add $8 - 4 = 4$ symbols.

We choose $a = \alpha$. It is such that $\{a^0, \ldots, a^{14}\}$ are all pairwise distinct.

Polynomial $G$ is

$$G = (x-1)(x-\alpha)(x-\alpha^2)(x-\alpha^3) = \alpha^6 + x + \alpha^4 x^2 + \alpha^{12} x^3 + x^4$$

Example (Corrupted message): Input is 4 symbols written as a polynomial

$$P = \alpha^6 + \alpha^8 x + \alpha^{14} x^2 + \alpha^{12} x^3$$

Encoding $S = PG$:

$$S = \alpha^{12} + \alpha^8 x + \boldsymbol{\alpha^2 x^2} + \alpha^5 x^3 + \alpha^{13} x^4 + \boldsymbol{\alpha^{14} x^5} + \alpha^4 x^6 + \alpha^{12} x^7$$

Received:

$$R = \alpha^{12} + \alpha^8 x + \boldsymbol{\alpha^5 x^2} + \alpha^5 x^3 + \alpha^{13} x^4 + \boldsymbol{\alpha^7 x^5} + \alpha^4 x^6 + \alpha^{12} x^7$$

1. Compute $R(1), R(\alpha), R(\alpha^2), R(\alpha^3)$:

$$R(1) = 0, R(\alpha) = \alpha^2, R(\alpha^2) = \alpha^3, R(\alpha^3) = \alpha^{14}$$

2. Their generating series is

$$0 + \alpha^2 x + \alpha^3 x^2 + \alpha^{14} x^3 = \frac{\alpha^2 x}{1 + \alpha x + \alpha^7 x^2} \mod x^4$$

3. Revert and find the roots of denominator

$$D(x) = 1 + \alpha x + \alpha^7 x^2$$

so

$$\tilde{D}(x) = x^2 + \alpha x + \alpha^7 = (x - \alpha^2)(x - \alpha^5)$$

Finding $E$, then $S$, then $P$: We know that errors happened at coefficients of $x^2$ and $x^5$:

$$E = c_0 x^2 + c_1 x^5$$

The numerator is $N(x) = \alpha^2 x$, so $\tilde{N}(x) = \alpha^2$. We have

$$c_0 = \frac{\tilde{N}(\alpha^2)}{\tilde{D}'(\alpha^2)} = \alpha, c_1 = \frac{\tilde{N}(\alpha^5)}{\tilde{D}'(\alpha^5)} = \alpha$$

so $E = \alpha x^2 + \alpha x^5$. Finally,

$$\begin{aligned} R - E &= \alpha^{12} + \alpha^8 x + (\alpha^{\mathbf{5}} - \alpha)x^2 + \alpha^5 x^3 + \alpha^{13} x^4 + (\alpha^{\mathbf{7}} - \alpha)x^5 + \alpha^4 x^6 + \alpha^{12} x^7 \\ &= \alpha^{12} + \alpha^8 x + \alpha^{\mathbf{2}} x^2 + \alpha^5 x^3 + \alpha^{13} x^4 + \alpha^{\mathbf{14}} x^5 + \alpha^4 x^6 + \alpha^{12} x^7 = S \end{aligned}$$

and

$$P = S/G = \alpha^6 + \alpha^8 x + \alpha^{14} x^2 + \alpha^{12} x^3$$

# Chapter 12

# Finite Field Algorithms

Error correcting algorithms over finite fields require

- Root finding: given $P$ in $\mathbb{F}[x]$, find all roots of $P$ in $\mathbb{F}$.
- Discrete logarithm: given $\alpha, \beta \in \mathbb{F}$, find $k$ such that $\alpha^k = \beta$.

Let $q = |\mathbb{F}|$. The naive approach is to try all elements in $\mathbb{F}$ to find the roots of $P$. Runtime is $q \times O(d) = O(dq)$ operations $+, \times$ in $\mathbb{F}$ with $d = \deg(P)$. Runtime of discrete algorithm is $O(q)$ multiplications to compute all powers of $\alpha$.

## 12.1 Squarefree Part

Suppose first that $P$ is in $\mathbb{Q}[x]$ and it factors as

$$P = P_1^{e_1} \dots P_s^{e_s}$$

with $P_1, \dots, P_s$ as irreducible factors of $P$ and $e_1, \dots, e_s$ multiplicities. We want

$$\overline{P} = P_1 \dots P_s$$

which has the same roots but no multiplicities. We do not know the $P_i$'s.

**Proposition**

$$\overline{P} = \frac{P}{\gcd(P', P)}$$

**E.g.** Let $P = (x - r_1)^{31}(x - r_2)^{62}$, then its derivative is

$$P' = 31(x-r_1)^{30}(x-r_2)^{62}+62(x-r_1)^{31}(x-r_2)^{61} = 31(x-r_1)^{30}(x-r_2)^{61}((x-r_2)+2(x-r_1))$$

$\gcd(P, P') = (x - r_1)^{s_1}(x - r_2)^{s_2}$ where $s_1 = 30, s_2 = 61$. Thus,

$$P/\gcd(P, P') = (x - r_1)(x - r_2)$$

The complexity is the cost of computing GCD's and divisions, so $O(M(d) \log d)$ if $\deg(P) = d$.

Over finite fields, the algorithm becomes complicated since in the previous example, it fails over $\mathbb{Z}/31\mathbb{Z}$ because $P' = 0$ as $31 = 62 = 0 \mod 31$.

## 12.2 Squares

Suppose that $\mathbb{F}$ is a finite field where $2 \neq 0$ (iff $1 \neq -1$), so this excludes $\mathbb{F}_2$ and $\mathbb{F}_2[x]/P(x)$.

**Proposition**

If $\mathbb{F}$ has $q$ elements,

- There are $\frac{q-1}{2}$ squares (excluding 0).
- $\alpha$ is a square if and only if $\alpha^{(q-1)/2} = 1$ (or $\alpha = 0$).

***Proof.*** (1) If $b$ is nonzero, $b^2$ is nonzero. Every square has at least two square roots (if $a = b^2$, then $a = (-b)^2$, and $b \neq -b$). It cannot have more than two square roots since $b^2 = c^2 \implies (b+c)(b-c) = 0$ and if $b - c \neq 0, b + c = 0$. So the $(q-1)$ nonzero elements in $\mathbb{F}$ map to $(q-1)/2$ squares.

(2) For any $a$ nonzero in $\mathbb{F}$, $a^{q-1} = 1$. So $(a^{(q-1)/2})^2 = 1 \implies a^{(q-1)/2} = \pm 1$. If $a = b^2$, then

$$a^{(q-1)/2} = b^{q-1} = 1$$

There are $(q-1)/2$ squares, so that gives us $(q-1)/2$ solutions to $a^{(q-1)/2} = 1$. For the non-squares, $a^{(q-1)/2} = -1$.

**E.g.** In $\mathbb{F} = \mathbb{Z}/31\mathbb{Z} = \{0, \ldots, 30\}$, we have $0^2 = 0$ and

| | | |
|---|---|---|
| $1^2 = 30^2 = 1$ | $6^2 = 25^2 = 5$ | $11^2 = 20^2 = 28$ |
| $2^2 = 29^2 = 4$ | $7^2 = 24^2 = 18$ | $12^2 = 19^2 = 20$ |
| $3^2 = 28^2 = 9$ | $8^2 = 23^2 = 2$ | $13^2 = 18^2 = 14$ |
| $4^2 = 27^2 = 16$ | $9^2 = 22^2 = 19$ | $14^2 = 17^2 = 10$ |
| $5^2 = 26^2 = 25$ | $10^2 = 21^2 = 7$ | $15^2 = 16^2 = 8$ |

So the nonzero squares are $\{1, 2, 4, 5, 7, 8, 9, 10, 14, 16, 18, 19, 20, 25, 28\}$.

## 12.3 Part 1: Keeping Only Linear Factors

Something we in the proof is

$$x^q - x = \prod_{a \in \mathbb{F}} (x - a)$$

**E.g.** Over $\mathbb{Z}/31\mathbb{Z}$, $x(x-1)\dots(x-30) = x^{31} - x$.

Preparing $P$: Make $P$ squarefree (optional), replace $P$ by $Q = \gcd(P, x^q - x)$, so

$$Q = \prod_{a \in \mathbb{F}, P(a)=0} (x - a)$$

**E.g.** $P = x^4 + 10x^3 + 9x^2 + 20x + 14$ gives

$$Q = \gcd(P, x^{31} - x) = x^2 + 10x + 7$$

Then, $Q = (x + 12)(x + 29)$. (We do not know it)

Complexity: $\deg(x^q - x) = q$. We saw that

$$\gcd(P, x^q - x) = \gcd(P, (x^q - x) \mod P)$$

To compute $x^q - x \mod P$, we compute $x^q \mod P$. There are $O(\log q)$ multiplication mod $P$, so $O(M(d) \log q)$. GCD is $O(M(d) \log d)$.

## 12.4 Part 2: Using Squares

At this stage, suppose $Q = (x - r_1)\dots(x - r_s)$, for some unknown $r_i$'s, all distinct. Take $A \in \mathbb{F}[x]$ of degree less than $s$ and compute

$$B = \gcd(A^{(q-1)/2} - 1, Q)$$

The roots of $B$ have some $r_i$ since they are roots of $Q$ and those for which $A(r_i)^{(q-1)/2} - 1 = 0$, that is, $A(r_i)$ is a nonzero square. So $B$ is a factor of $Q$.

**E.g.** Take $Q = x^2 + 10x + 7 = (x - 19)(x - 2)$. We try some $A$'s:

- With $A = x + 2$, we have $B = x + 29$. $(A(19) = 21, A(2) = 4)$
- With $A = x$, we have $B = x^2 + 10x + 7$ $(A(19) = 19, A(2) = 2)$
- With $A = 3x$, we have $B = 1$ $(A(19) = 26, A(2) = 6)$

If $B \neq 1$ and $B \neq Q$, we are done because here $\deg(Q) = 2$. Otherwise, it is a recursive call.

Probability of splitting: The sample space is choosing $s$ coefficients of $A$ which is equivalent to choosing $A(r_1), \dots, A(r_s)$. There are $q^s$ choices.

We lose if

- either all $A(r_i)$'s are nonzero squares $(B = Q)$. There are $\left(\frac{q-1}{2}\right)^2 \simeq \frac{q^s}{2^s}$ choices.
- or all are non-squares $B = 1$. There are $\left(\frac{q+1}{2}\right)^s \simeq \frac{q^s}{2^s}$ choices.

The probability of losing is

$$\frac{q^s/2^s + q^s/2^s}{q^s} = \frac{1}{2^{s-1}} \leq \frac{1}{2}$$

So the probability of winning is $\geq \frac{1}{2}$, we we expect to win in $O(1)$ trials.

$$E(\text{trials}) = 1 \cdot p + 2p(1-p) + 3p(1-p)^2 + \cdots = \frac{1}{p} \leq 2$$

Complexity: Let $s := \deg(Q) \leq d$. We have to be careful since $A^{(q-1)/2}$ has degree $O(sq)$. Again

$$\gcd(A^{(q-1)/2} - 1, Q) = \gcd(A^{(q-1)/2} - 1 \mod Q, Q)$$

It takes $O(\log q)$ multiplications mod $Q$, so $O(M(s)\log q)$. After that, GCD is $O(M(s)\log s)$.

The cost to split $Q$ into factors is $O_E(M(s)\log(qs))$ where $O_E$ is the expected runtime.

## 12.5 Factoring Polynomials Over Finite Fields

There are three factorization methods:

1. Squarefree factorization: Find the largest factor of $f \in \mathbb{F}_q[x]$ which is squarefree, generally referred to as the squarefree part of $f$.
2. Distinct degree factorization: Given a squarefree $f \in \mathbb{F}_q[x]$, find $g_1, \ldots, g_n \in \mathbb{F}_q[x]$ such that $g_d$ is the product of all factors of $f$ which have degree $d$ for $1 \leq d \leq n$.
3. Equal degree factorization: Given a squarefree $g \in \mathbb{F}_q[x]$, all of whose irreducible factors have some fixed degree $d$, find $h_1, \ldots, h_k \in \mathbb{F}_q[x]$ of degree $d$ such that $g = h_1 \ldots h_k$.

**Theorem (Distinct Degree Factorization)**

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

**Algorithm 11** DistinctDegreeFactorization

**Input**: $f \in \mathbb{F}_q[x]$ monic, squarefree of degree $n > 0$.
**Output**: $g_1, \ldots, g_n \in \mathbb{F}_q[x]$ such that $g_d$ is the product of all irreducible factors of $f$ of degree $d$.
$h_0 = x, f_0 = f$
**for** $i = 1$ to $n$ while $f_i \neq 1$ **do**
    Compute $h_i = h_{i-1}^q \mod f$ by repeated squaring
    $g_i = \gcd(h_i - x, f_{i-1})$
    $f_i = f_{i-1}/g_i$
**return** $g_1, \ldots, g_n$

# Chapter 13

# Gröbner Bases

Assume $f_1, \ldots, f_s \in \mathbb{F}[x_1, \ldots, x_n]$. We want to solve the system

$$f_1(x_1, \ldots, x_n) = 0$$
$$f_2(x_1, \ldots, x_n) = 0$$
$$\vdots f_s(x_1, \ldots, x_n) \qquad = 0$$

i.e. find $\{(a_1, \ldots, a_n) \in \mathbb{F}^n : f_i(a_1, \ldots, a_n) = 0, \forall 0 \le i \le s\}$.

## 13.1 Algebraic Sets and Ideals

**Definition: Algebraic Set**

Given a collection of polynomials $\mathcal{F} \subset \mathbb{F}[x_1, \ldots, x_n]$, the set

$$V(\mathcal{F}) := \{(a_1, \ldots, a_n) \in \mathbb{F}^n : f(a_1, \ldots, a_n) = 0, \forall f \in \mathcal{F}\}$$

Examples: Circle $V(x^2+y^2-1)$, Lorenz cone $V(z^2-x^2-y^2)$, Twisted cubic $V(y-x^2, z-x^3)$, Line and Hyperplane $V(xz, yz)$.

**Definition: Ideal**

Given a ring $R$, an ideal $I \subset R$ such that

- $(I, +)$ is a subgroup of $(R, +)$.
- $I$ is closed under multiplication by elements of $R$

$$a \in I, s \in R \implies s \cdot a \in I$$

**E.g.**

- $(0)$ is ideal generated by the 0 element of the ring.

- $R$ is an ideal of $R$.
- In $\mathbb{Z}$, the set of even numbers is the ideal generated by 2, denoted (2).
- In $\mathbb{Q}[x]$, the set of all polynomials whose constant coefficient is 0 is the ideal $(x)$ generated by $x$.
- In $\mathbb{Q}[x, y]$, the set of all polynomials whose constant coefficient is 0 is the ideal $(x, y)$ generated by $x$ and $y$.

**Definition: Ideal Generation and Basis**

Let $f_1, \ldots, f_s$ be polynomials in $\mathbb{F}[x_1, \ldots, x_n]$. Then

$$I = (f_1, \ldots, f_s) = \left\{ \sum_{i=1}^{s} p_i f_i : p_i \in \mathbb{F}[x_1, \ldots, x_n], 1 \leq i \leq s \right\}$$

is an ideal of $\mathbb{F}[x_1, \ldots, x_n]$ and is the ideal generated by $f_1, \ldots, f_s$.
Moreover, $f_1, \ldots, f_s$ are a basis of $I$.

**Theorem (Hilbert Basis)**

Every ideal $I \subset \mathbb{F}[x_1, \ldots, x_n]$ has a finite generating set.

Relationship between algebraic sets and ideals: For a given polynomial system of equations, defined by $f_1, \ldots, f_s \in \mathbb{F}[x_1, \ldots, x_n]$, we consider $I = (f_1, \ldots, f_s)$. Then we can use the following proposition.

**Proposition**

If $f_1, \ldots, f_s$ and $g_1, \ldots, g_t$ are bases of the same ideal in $\mathbb{F}[x_1, \ldots, x_n]$, then

$$V(f_1, \ldots, f_s) = V(g_1, \ldots, g_t)$$

**Gaussian Elimination**: Given a matrix $A \in \mathbb{F}^{n \times d}$, vector $b \in \mathbb{F}^n$, we want a solution $x \in \mathbb{F}^d$ to $Ax = b$.

1. Put $C = \begin{pmatrix} A & b \end{pmatrix}$ in reduced row-echelon form.
2. From bottom-up along rows of $A$, if the equation has a solution then set it properly.
3. So long as there are no inconsistencies, we found a solution.

## 13.2 Monomial Ordering

In the division algorithm over $\mathbb{F}[x]$, we assumed $c \leq x \leq x^2 \leq x^3 \leq \cdots$. In our linear system solving algorithm, we implicitly assumed that $y_1 \geq y_2 \geq \cdots \geq y_d$. We can assume a similar ordering for monomials $\mathbb{F}[x_1, \ldots, x_n]$.

We will denote $x^{\mathbf{a}} = x_1^{e_1} \dots x_n^{e_n}$ where $\mathbf{a} = (e_1, \dots, e_n) \in \mathbb{N}^n$ and $\mathbb{F}[x] = \mathbb{F}[x_1, \dots, x_n]$.

**E.g.** Given two monomials $x^{\mathbf{a}}, x^{\mathbf{b}} \in \mathbb{F}[x_1, \dots, x_n]$, we say $x^{\mathbf{a}} \succeq x^{\mathbf{b}}$ if $\mathbf{a} \geq \mathbf{b}$ in lexicographic order over $\mathbb{N}^n$. In general, a good monomial order has

- Total order: any two elements can be compared.
- Transitive: $x^{\mathbf{a}} \succeq x^{\mathbf{b}}$ and $x^{\mathbf{b}} \succeq x^{\mathbf{c}}$, then $x^{\mathbf{a}} \succeq x^{\mathbf{c}}$.
- Well-behaved under multiplication: $x^{\mathbf{a}} \succeq x^{\mathbf{b}} \implies x^{\mathbf{a}+\mathbf{c}} \succeq x^{\mathbf{b}+\mathbf{c}}$.
- Well-ordering: every non-empty subset has a smallest element.

Define the polynomial

$$f(x) = \sum_{\alpha} f_\alpha x^\alpha$$

**Definition: Support**

$$\text{Supp}(f) := \{\alpha \in \mathbb{N}^n : f_\alpha \neq 0\}$$

**Definition: Multidegree**

The maximum monomial in the support of $f$ according to $\succeq$.

**Definition: Leading Monomial**

$$LM(f) := x^{\text{multideg}(f)}$$

**Definition: Leading Coefficient**

$$LC(f) := f_{\text{multideg}(f)}$$

**Definition: Leading Term**

$$LC(f) \cdot LM(f)$$

## 13.3 Division Algorithm

**Division**

**Input**: polynomials $G, F_1, \ldots, F_s \in \mathbb{F}[x]$ and a monomial order $\succeq$.
**Output**: $Q_1, \ldots, Q_s, R \in \mathbb{F}[x]$ such that

$$G = F_1 \cdot Q_1 + \cdots + F_s \cdot Q_s + R$$

where $\text{multideg}(R) < \text{multideg}(F_i)$ for all $i$.

### 13.3.1 First Attempt

Idea is the same as in the one-variable case. Cancel the leading term of $G$ by using $F_i$.

**E.g.** $G = xy^2 + 1, F_1 = xy + 1, F_2 = y + 1$, we have

$$xy^2 + 1 = y \cdot (xy + 1) + (-1) \cdot (y + 1) + 2$$

The quotients are not unique as we can have

$$xy^2 + 1 = xy \cdot (xy + 1) + (-1) \cdot (y + 1) + 2$$

**E.g.** $G = x^2y + xy^2 + y^2, F_1 = xy - 1, F_2 = y^2 - 1$ with lexicographic order.

$$x^2y + xy^2 + y^2 = (x + y)(xy - 1) + 1(y^2 - 1) + (x + y + 1)$$

So instead of requiring that the leading term of remainder be smaller than leading term of divisors, it is better to require that no monomial of $R$ is divisible by any leading monomial of the $F_i$'s

### 13.3.2 Second Attempt

Redefine the problem

**Division**

**Input**: polynomials $G, F_1, \ldots, F_s \in \mathbb{F}[x]$ and a monomial order $\succeq$.
**Output**: $Q_1, \ldots, Q_s, R \in \mathbb{F}[x]$ such that

$$G = F_1 \cdot Q_1 + \cdots + F_s \cdot Q_s + R$$

no monomial of $R$ be divisible by any leading term of the $F_i$'s.
Furthermore, if $F_iQ_i \neq 0$, we also want $LM(G) \succeq LM(F_iQ_i)$.

**Algorithm 12** Division Algorithm

1: While $LM(G)$ is divisible by some $LM(F_i)$, divide appropriately (respecting the order preference of $F_i$'s)

2: If no $LM(F_i) \mid LM(G)$, add $LT(G)$ to the remainder and go back to step 1.

The algorithm always terminates. Proof is by well-ordering principle of the monomial order and fact that each step of division algorithm decreases leading term of $G$.

Properties we want from a division algorithm:

- Remainder should be uniquely determined.
- Ordering should not matter.
- Univariate division algorithm solves ideal membership problem.

**E.g.** $G = x^2y + xy^2 + y^2, F_1 = y^2 - 1, F_2 = xy - 1$. The remainder here is $2x + 1$, which is different from the remainder in the previous example $(x + y + 1)$.

**Definition: Monomial Ideal**

Any ideal generated by a family $\mathcal{F}$ of monomials.

Question: Does every ideal of $\mathbb{F}[x_1, \ldots, x_n]$ have a finite description.

**Theorem (Dickson's Lemma)**

Let $I = (x^\alpha : \alpha \in \mathcal{F}) \subset \mathbb{F}[x_1, \ldots, x_n]$ be a monomial ideal. Then $I$ can be written as

$$I = (x^{\alpha(1)}, \ldots, x^{\alpha(s)})$$

where $\alpha(1), \ldots, \alpha(s) \in \mathcal{F}$.

Dickson's Lemma helps us decide if a monomial relation is a proper monomial ordering.

**Corollary (Monomial Order Criterion)**

If $>$ is a relation on $\mathbb{N}^n$ satisfying

- $>$ is a total ordering on $\mathbb{N}^n$
- $\alpha > \beta$ and $\gamma \in \mathbb{N}^n$, then $\alpha + \gamma > \beta + \gamma$

Then $>$ is a well-ordering if and only if $\alpha \geq 0$ for all $\alpha \in \mathbb{N}^n$.

**Definition: Minimal Basis of a Monomial Ideal**

One where none of the generators is divisible by another generator.

Some issues with the division algorithm

- Our division algorithm only gives sufficient condition for ideal membership problem: if $G$ has zero remainder when divided by $(F_1, \dots, F_s)$, then we know $G \in (F_1, \dots, F_s)$.
- Main problem is due to the fact that for some generators of an ideal, we are missing important leading monomials.

The fix for this division algorithm is to find a good basis for the ideal generated by $F_1, \dots, F_s$ called the Gröbner basis.

A Gröbner basis is one which contains all the important leading monomials.

Given ideal $I \subseteq \mathbb{F}[x]$ and a monomial ordering $>$, let

- $LT(I)$ be the set of all leading terms of nonzero elements of $I$.
- $LM(I)$ be the monomial ideal generated by $LT(I)$.

By Dickson's lemma, we know that $LM(I)$ is *finitely generated*. From previously, we know that given a generating set for $I$, it could be the case that the leading terms of the generators are *strictly contained* in $LT(I)$.

## 13.4 Hilbert's Basis Theorem

Proof of Hilbert Basis Theorem:

Let $I \subseteq \mathbb{F}[x]$ be an ideal. By Dickson's lemma, $LM(I)$ is finitely generated. Let $g_1, \dots, g_s \in I$ such that $LM(I) = (LM(g_1), \dots, LM(g_s))$. The division algorithm shows that $I \subseteq (g_1, \dots, g_s)$.

Note that for any $f \in I$ we have that

$$LM(f) \in LM(I) = (LM(g_1), \dots, LM(g_s))$$

So long as $f$ is nonzero and in $I$ we will be able to divide, and remainder will be zero. Since the division algorithm always terminates, we will end with remainder zero.

From this proof, we see the existence of a very special generating set of our ideal.

**Definition: Gröbner Basis of an Ideal**

A generating set which has the property that the special generating set was that the leading monomials of generating set generate the ideal $LM(I)$.

A property of Gröbner bases is the uniqueness of the remainder in the division algorithm. Precisely, if $G = \{g_1, \dots, g_s\}$ is a Gröbner basis for $I$, then given $f \in \mathbb{F}[x]$, there is a unique $r \in \mathbb{F}[x]$ with the following properties:

- no term of $r$ is divisible by any $LM(g_i)$.

- there is $g \in I$ such that $f = g + r$.

The division algorithm gives existence of $r$ and uniqueness comes from the fact that if $r, r'$ are remainders, then $r - r' \in I \implies r = r'$ by the division algorithm.

**Definition: $S$-Polynomial**

Given two polynomials $f, g \in \mathbb{F}[x]$, let $x^\gamma = LCM(LM(f), LM(g))$. Then the $S$-polynomial of $f, g$ is

$$S(f,g) := \frac{x^\gamma}{LT(f)} \cdot f - \frac{x^\gamma}{LT(g)} \cdot g$$

$S$-polynomials are designed to produce cancellations of leading terms.

**Lemma**

If we have a sum $p_1 + \cdots + p_s$ where $\text{multideg}(p_i) = \delta \in \mathbb{N}^n$ for all $i \in [s]$ such that $\text{multideg}(p_1 + \cdots + p_s) < \delta$, then $p_1 + \cdots + p_s$ is a linear combination, with coefficients in $\mathbb{F}$, of the $S$-polynomials $S(p_i, p_j)$, where $i, j \in [s]$.

**Definition: Buchberger's Criterion**

Let $I \subseteq \mathbb{F}[x]$ be an ideal. Then a basis $G = \{g_1, \ldots, g_s\}$ of $I$ is a Gröbner basis of $I$ if and only if for all pairs $i \neq j$, the remainder on division of $S(g_i, g_j)$ by $G$ is 0.

We denote $S(g_i, g_j)^G = 0$ as short-hand notation to say remainder of division by $G$ is 0.

**Algorithm 13** Buchberger's Algorithm

```
1: Input: I = (f_1, ..., f_s)
2: Output: Gröbner basis G for I
3: G = {f_1, ..., f_s}
4: while there is S_ij := S(f_i, f_j) such that S_ij^G ≠ 0 do
5:     G = G ∪ {S_ij}
6: Once all S_ij^G = 0, return G
```

The criterion ensures the algorithm always returns a Gröbner basis and terminates because of Dickson's lemma.

Of all Gröbnmer bases for an ideal $I$, one is special by the following:

- $LC(p) = 1$ for all $p \in G$.
- For all $p \in G$, no monomial of $p$ lies in $(LT(G) \setminus \{p\})$.

These are called **reduced Gröbner bases**.

Applications:

- The solution to the Ideal Membership Problem: Given $f, I$, simply compute Gröbner basis $G$ of $I$ and $f \in I \Leftrightarrow f^G = 0$.
- Solving system of polynomial equations:
  1. Compute Gröbner basis using lex order $x_1 > \cdots > x_n$.
  2. Solve system just like a linear system.

  Example: $I = (x^2 + y^2 + z^1 - 1, x^2 + z^2 - y, x - z)$, the Gröbner basis for $I$ is

  $$G = \left\{x - z, y - 2z^2, z^4 + \frac{1}{2}z^2 - \frac{1}{4}\right\}$$